



Classification

What is classification?

- Goal: Learn a classification model $f: \mathcal{X} \rightarrow \mathcal{Y}$
- A training set $\mathcal{X} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$
- The feature vector $\mathbf{x}_i \in \mathcal{X}$ is a d dimensional vector, where each feature can be
 - Numeric: Real, Integer
 - Categorical (Nominal): Binary, Polynominal

The label $y_i \in \mathcal{Y}$ can be

- Binary Classification: $\mathcal{Y} = -1, +1$ or $\{0, 1\}$
- Multiclass Classification: $\mathcal{Y} = \{1, 2, \dots, C\}$
- Multilabel Classification

- The Identical Independent Distribution (i.i.d.) assumption
 - Each training example is drawn independently from the identical source

Basic Classification Methods

- We have three basic classification methods:
 - Instance based learning
 - E.g. K-NN
 - Discriminative methods
 - E.g. Logistic-Regression
 - Generative methods
 - E.g. Naïve Bayes

Instance based learning

- Key idea
 - Just store all training examples
 - Upon querying, calculate a prediction using the stored data
- IBL is also known as
 - Memory-based learning
 - Lazy Learning
- K Nearest Neighbors (kNN)
 - Most prominent representative of IBL

Instance-Based Classifiers

Set of Stored Cases			
Atr1	AtrN	Class
			A
			B
			B
			C
			A
			C
			B

- Store the training records
- Use training records to predict the class label of unseen cases

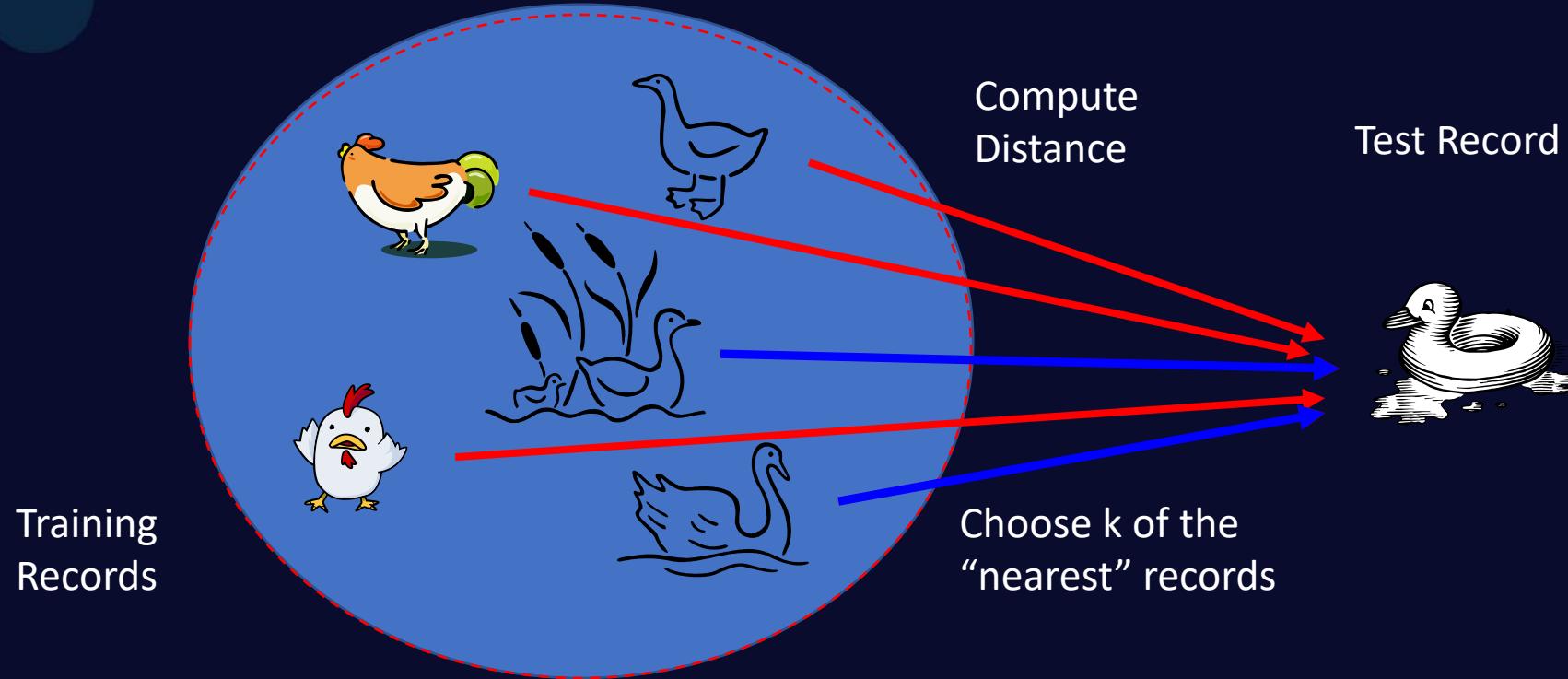
Unseen Case		
Atr1	AtrN

Instance Based Classifiers

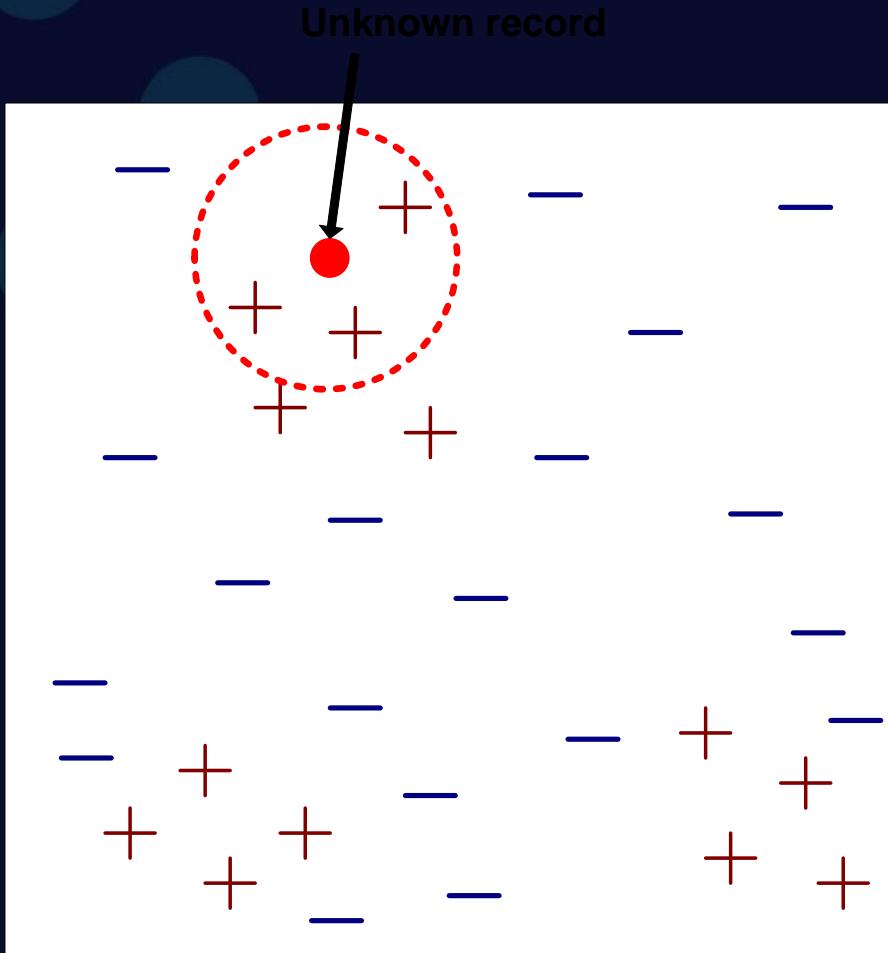
- Examples:
 - Rote-learner
 - Memorizes entire training data and performs classification only if attributes of record match one of the training examples exactly
 - Nearest neighbor
 - Uses k “closest” points (nearest neighbors) for performing classification

Nearest Neighbor Classifiers

- Basic idea:
 - If it walks like a duck, quacks like a duck, then it's probably a duck



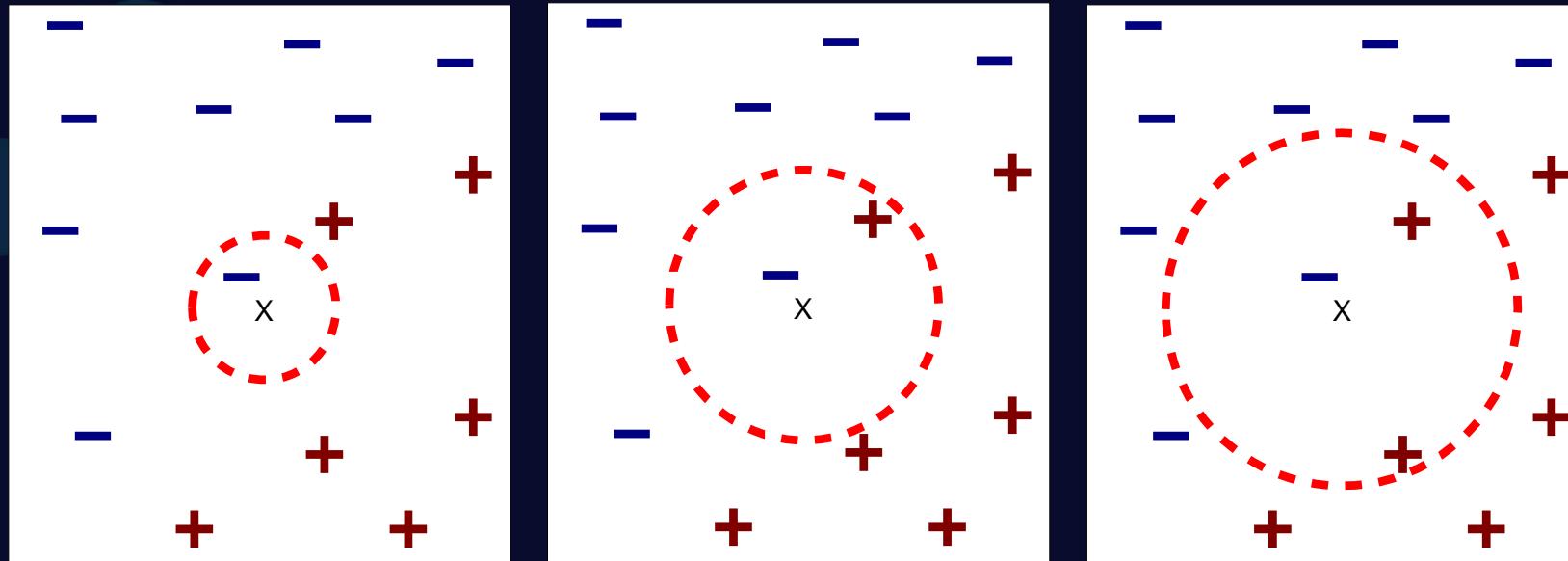
Nearest-Neighbor Classifiers



- Requires three things
 - The set of stored records
 - Distance Metric to compute distance between records
 - The value of k , the number of nearest neighbors to retrieve

- To classify an unknown record:
 - Compute distance to other training records
 - Identify k nearest neighbors
 - Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

Definition of Nearest Neighbor



(a) 1-nearest neighbor

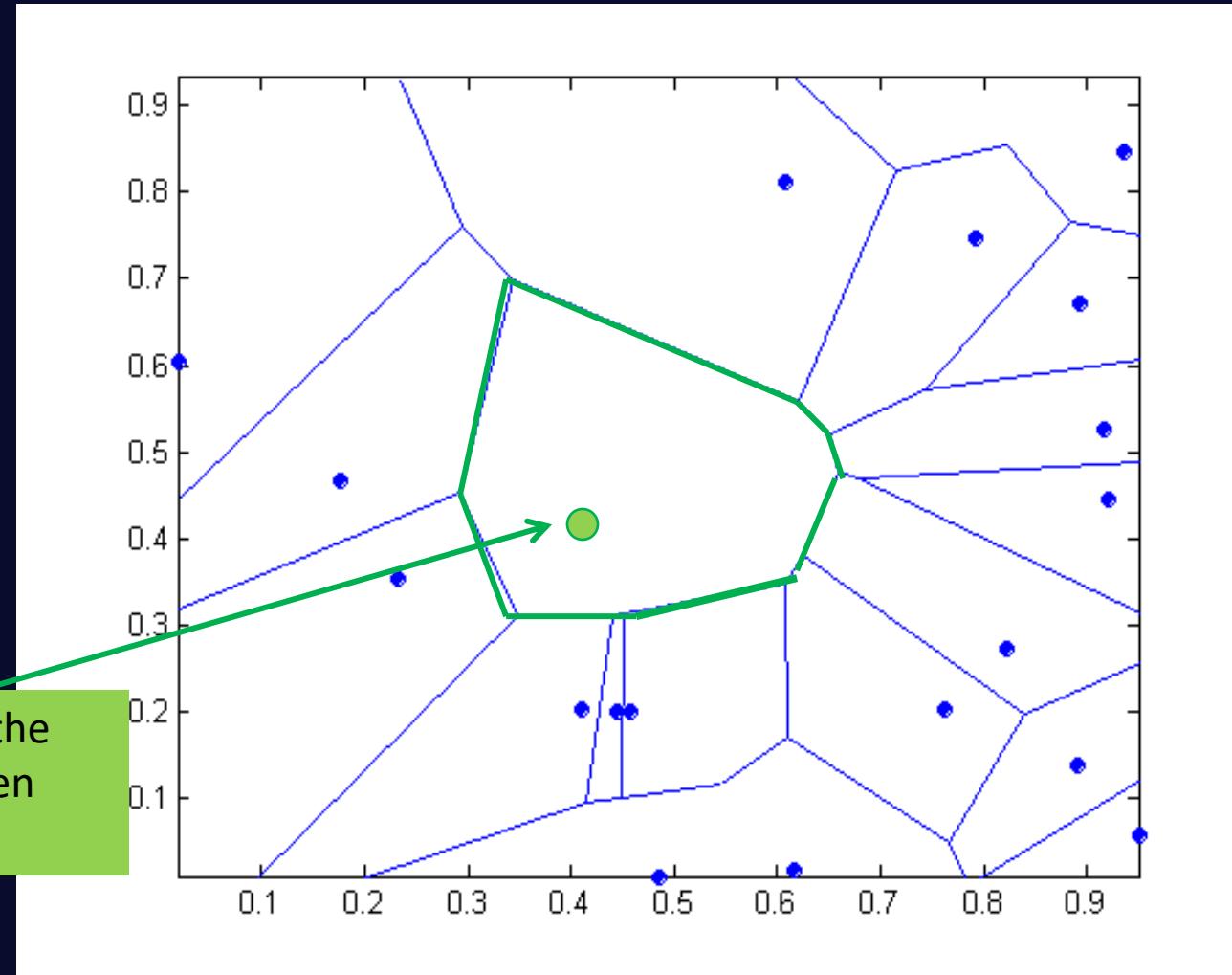
(b) 2-nearest neighbor

(c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

1 nearest-neighbor

Voronoi Diagram defines the classification boundary



Issues

- Distance measure
- Choosing K
- Features Scaling
- For high-dimensional the nearest neighbor may not be very close at all
- Implementation - Memory based technique. For each classification must pass through the entire data

Distance measure

- Compute distance between two points:
 - Common choice - Euclidean distance

$$d(p, q) = \sqrt{\sum_i (p_i - q_i)^2}$$

- Determine the class from nearest neighbor list
 - take the majority vote of class labels among the k-nearest neighbors
 - Weigh the vote according to distance
 - weight factor, $w = 1/d^2$

Distance measure

- Problem with Euclidean measure:
 - High dimensional data
 - curse of dimensionality
 - Can produce counter-intuitive results

1 1 1 1 1 1 1 1 1 1 0

0 1 1 1 1 1 1 1 1 1 1

vs

1 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 1

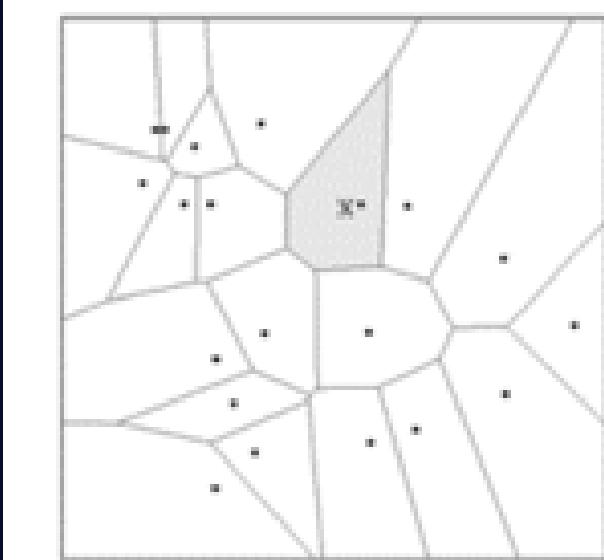
$d = 1.4142$

$d = 1.4142$

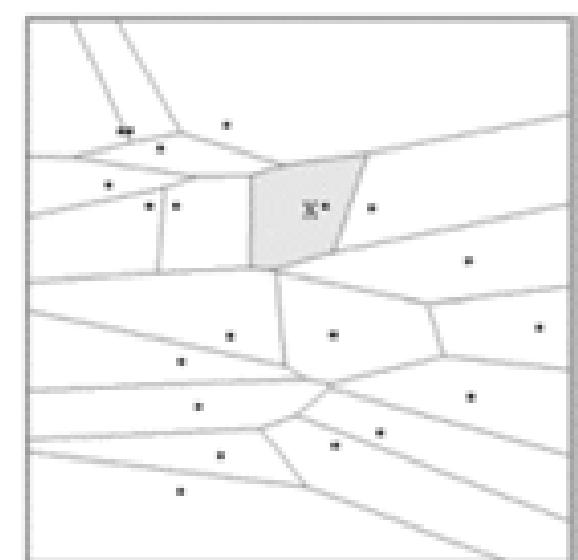
- ◆ Solution: Normalize the vectors to unit length

Distance measure

- The choice of the distance has a significant effect on the decision boundaries



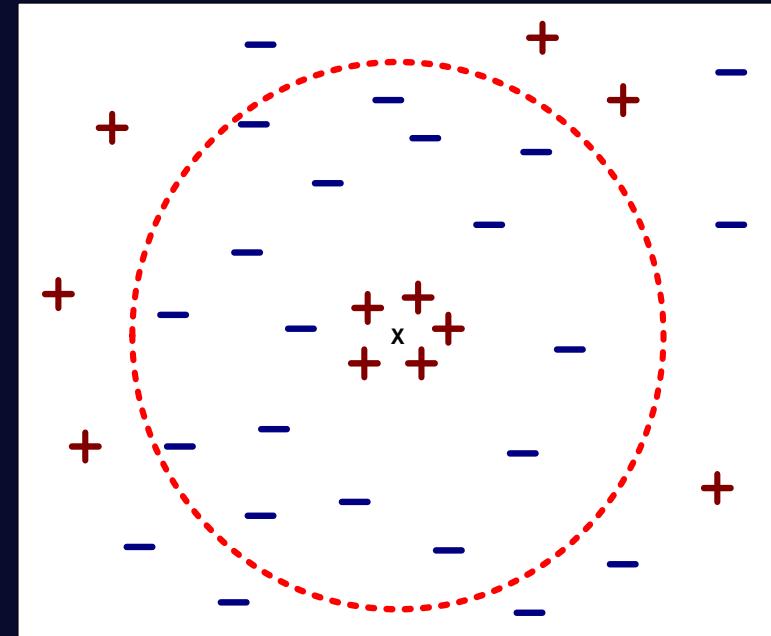
$$\text{Dist}(a,b) = (a_1 - b_1)^2 + (a_2 - b_2)^2$$



$$\text{Dist}(a,b) = (a_1 - b_1)^2 + (3a_2 - 3b_2)^2$$

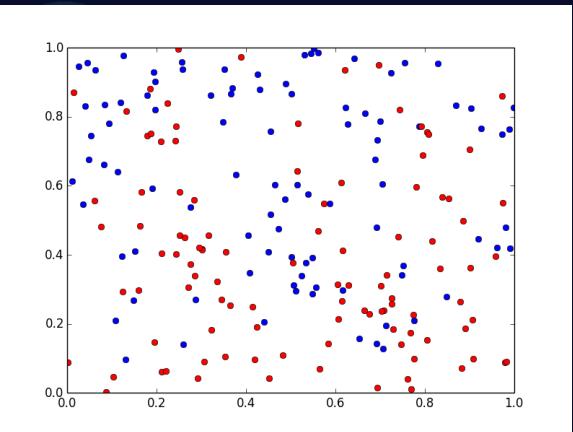
Choosing K

- Choosing the value of k:
 - If k is too small, sensitive to noise points
 - If k is too large, neighborhood may include points from other classes

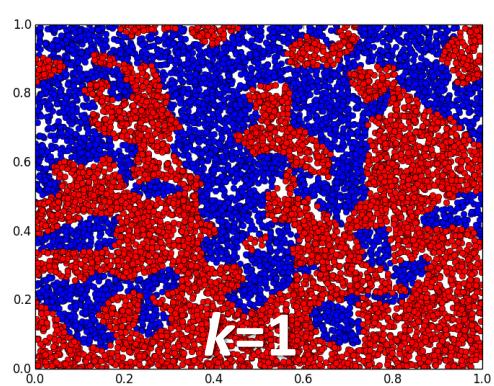
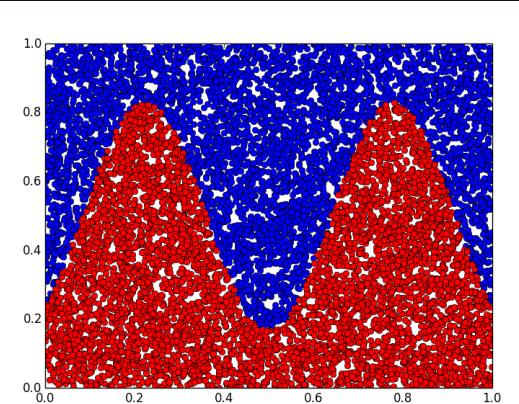


k -Nearest Neighbor: Bias- Variance Tradeoff

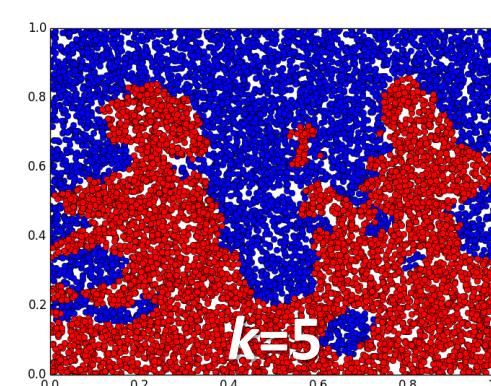
$S =$



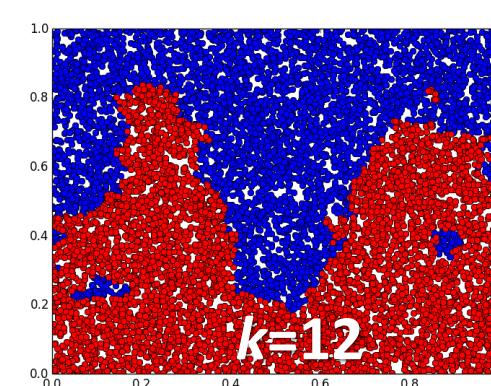
$h^* =$



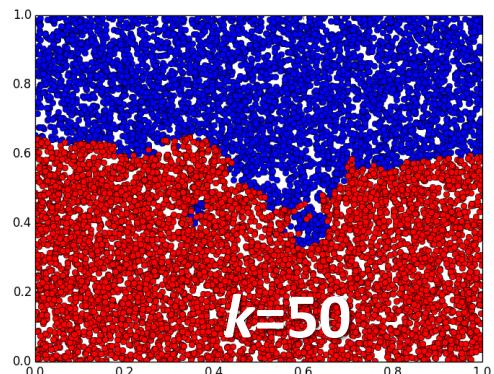
$k=1$



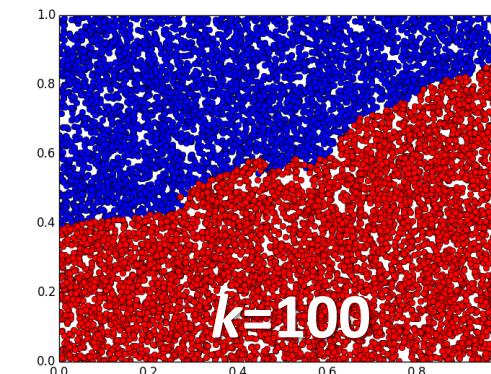
$k=5$



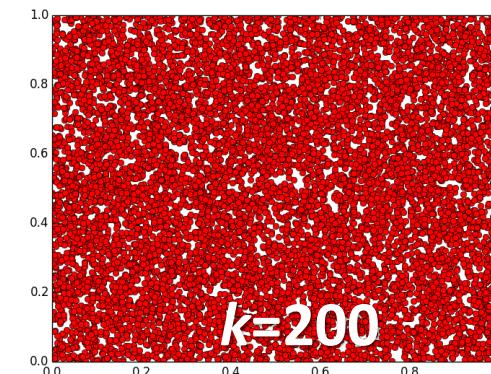
$k=12$



$k=50$



$k=100$



$k=200$

Features Scaling

- Scaling issues
 - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
 - Example:
 - height of a person may vary from 1.5m to 1.8m
 - weight of a person may vary from 90lb to 300lb
 - income of a person may vary from \$10K to \$1M

The Curse of Dimensionality

- K-NN breaks down at high-dimensional spaces because of the neighborhood becomes very large.
- Suppose we have 5000 points uniformly distributed in the unit hypercube and we want to apply The 5--nearest neighbor algorithm:
 - 1D - On one dimensional line we must go a distance of $5/5000=0.001$ on average to capture 5 nearest neighbors
 - 2D – We must go $\sqrt{0.001}$ to capture 0.001 of the volume
 - D – We must go $(0.001)^{(1/d)}$

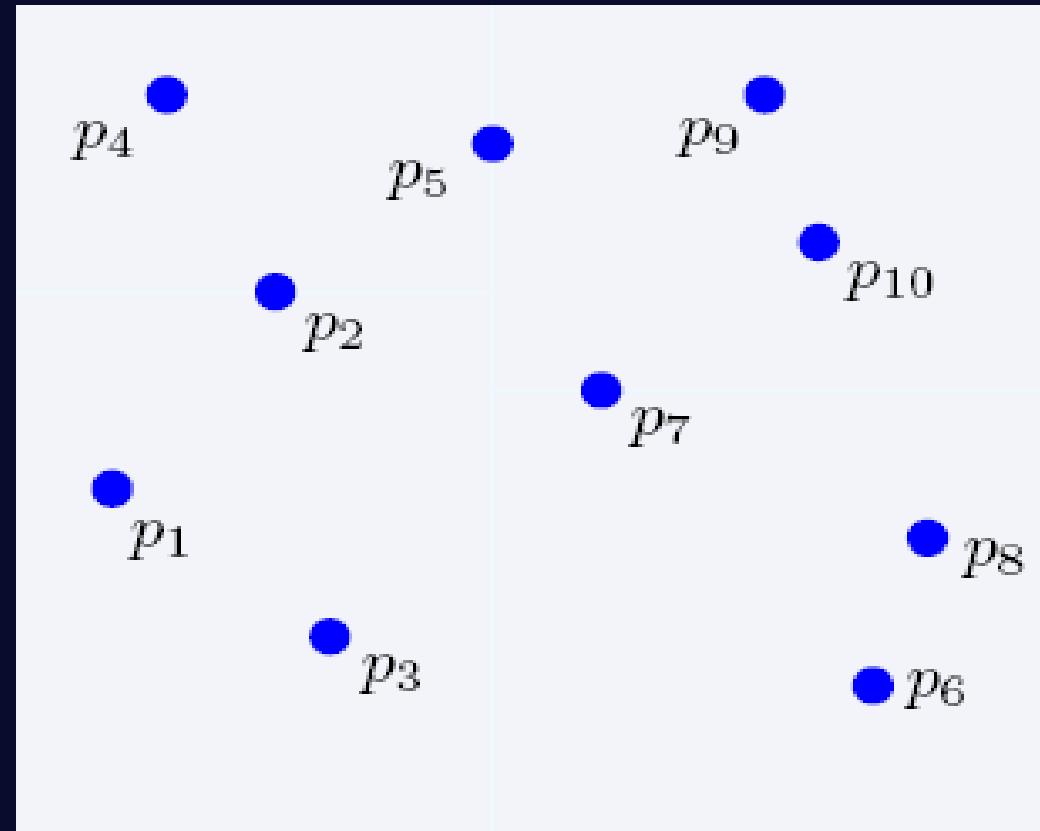
Nearest neighbor Classification

- k-NN classifiers are **lazy learners**
 - It does not build models explicitly
 - Unlike **eager learners** such as decision tree induction and rule-based systems
- Classifying unknown records are relatively expensive
 - Naïve algorithm: $O(n)$
 - Need for structures to retrieve nearest neighbors fast.
 - The **Nearest Neighbor Search** problem.

Nearest Neighbor Search

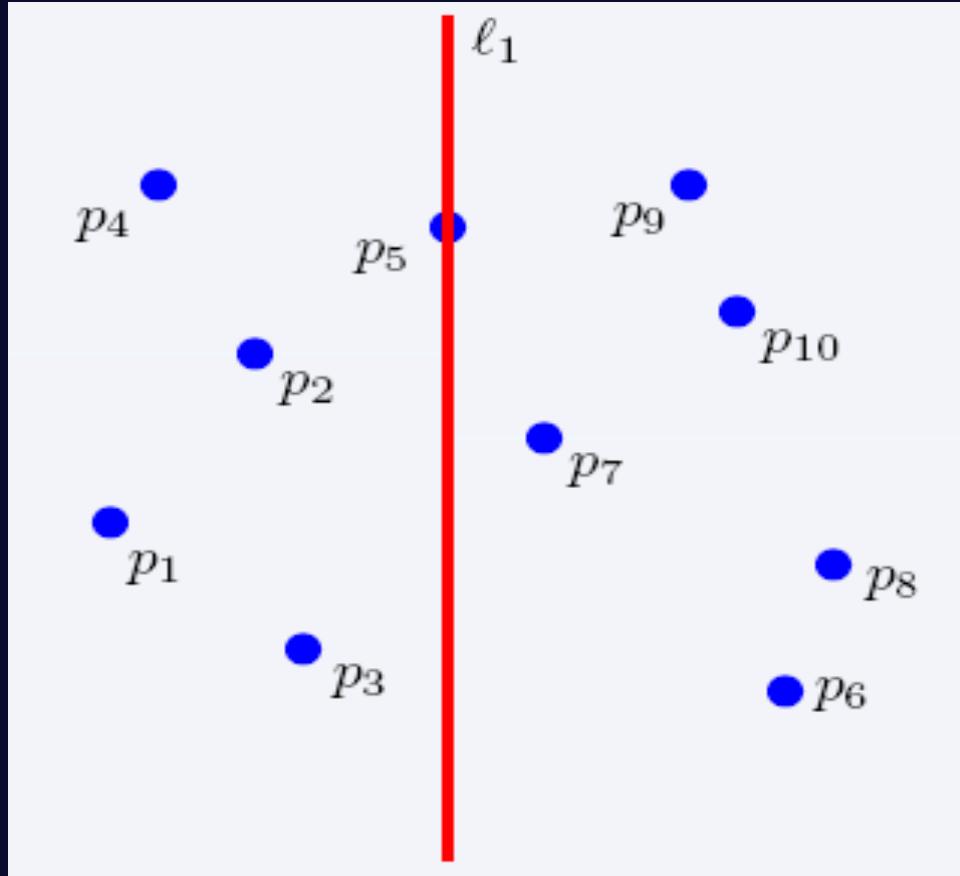
- Two-dimensional **kd-trees**
 - A data structure for answering nearest neighbor queries in \mathbb{R}^2
- kd-tree construction algorithm
 - Select the **x** or **y** dimension (alternating between the two)
 - Partition the space into two with a line passing from the median point
 - Repeat recursively in the two partitions as long as there are enough points

Nearest Neighbor Search



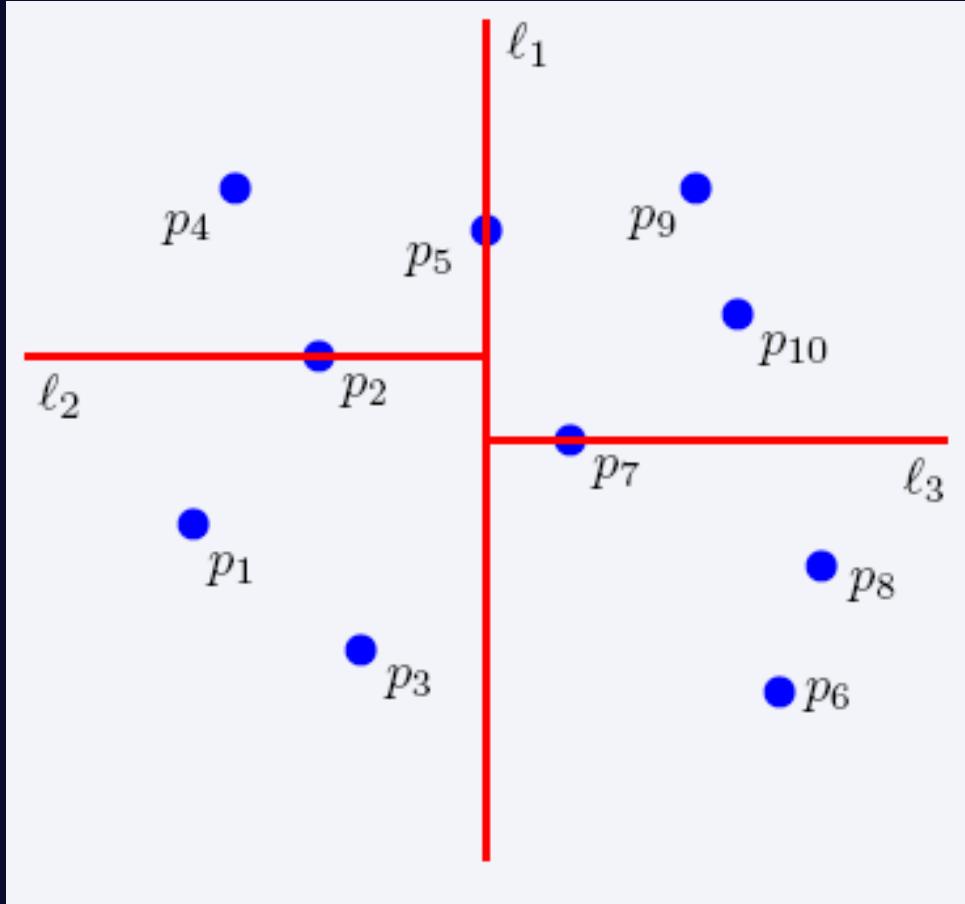
2-dimensional kd-trees

Nearest Neighbor Search



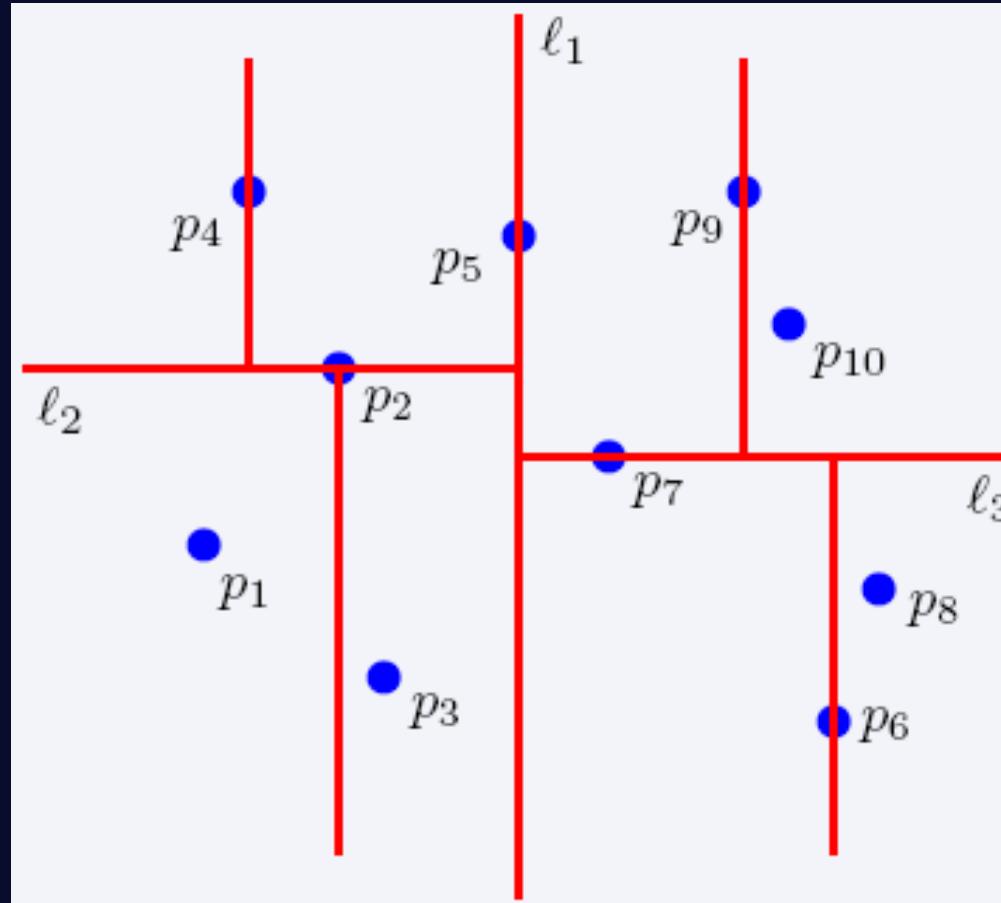
2-dimensional kd-trees

Nearest Neighbor Search



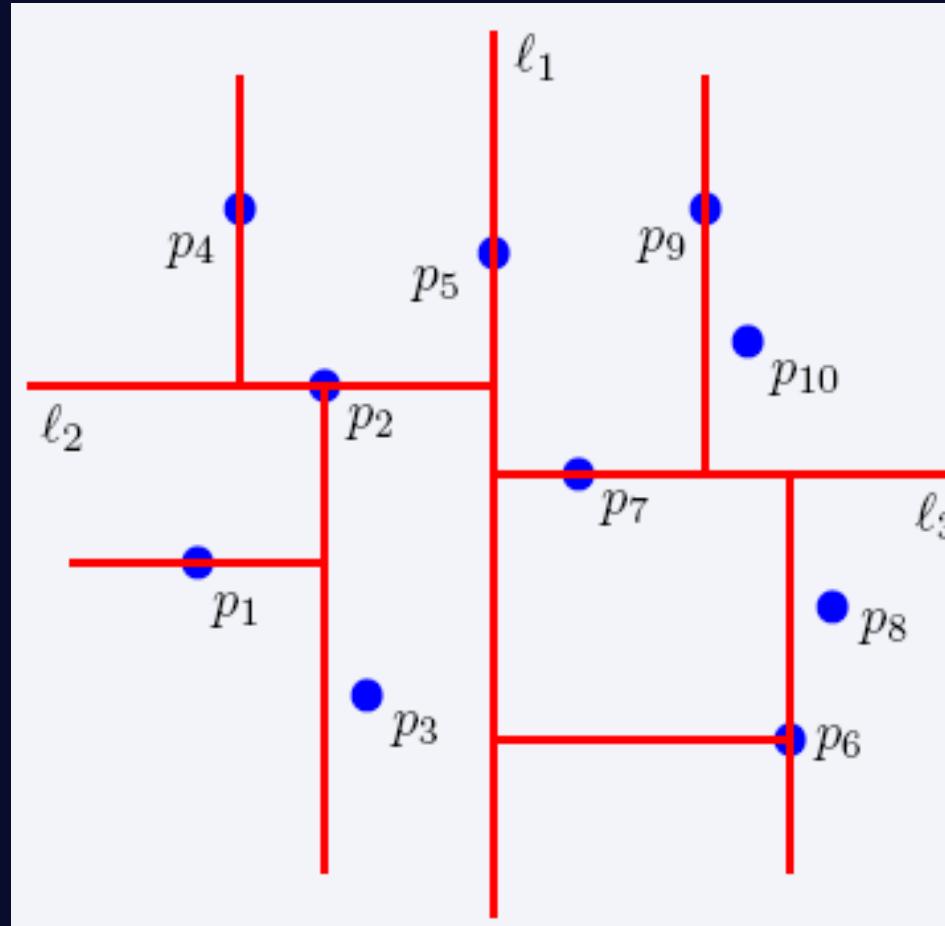
2-dimensional kd-trees

Nearest Neighbor Search



2-dimensional kd-trees

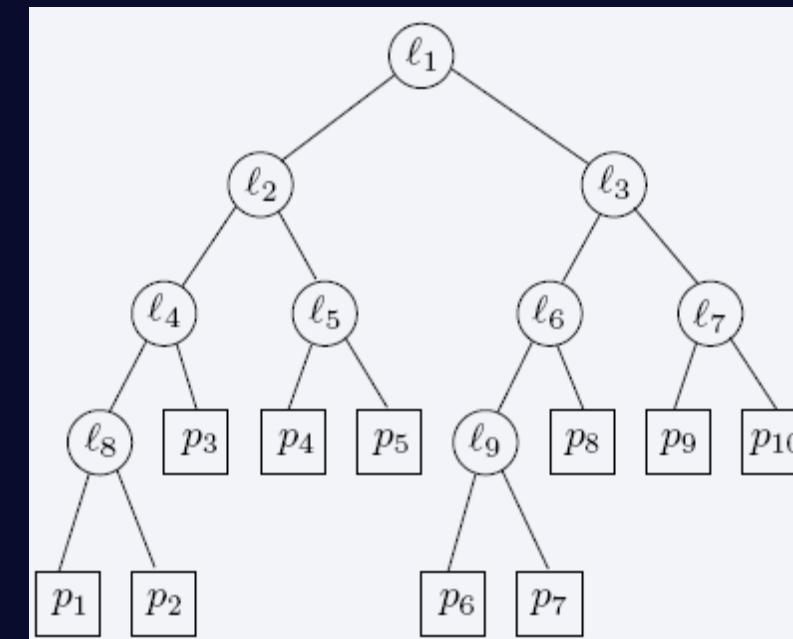
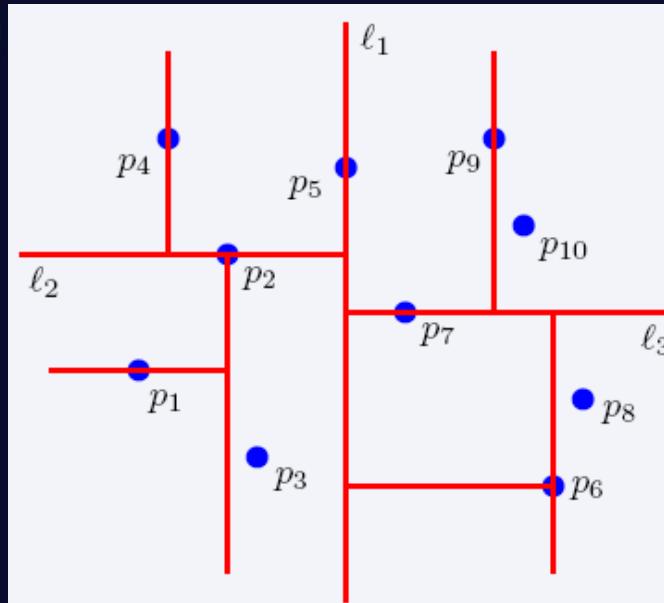
Nearest Neighbor Search



2-dimensional kd-trees

Nearest Neighbor Search

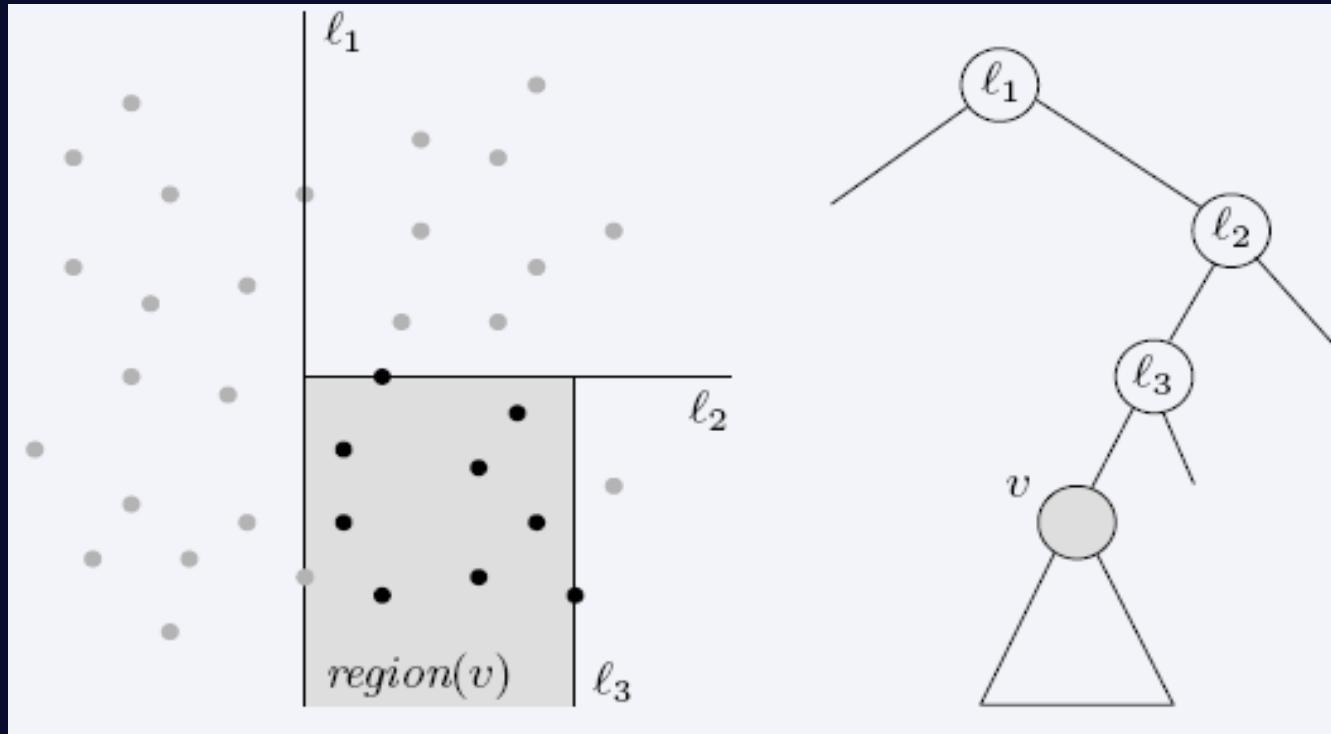
2-dimensional kd-trees



Nearest Neighbor Search

2-dimensional kd-trees

$\text{region}(u)$ – all the black points in the subtree of u



Nearest Neighbor Search

2-dimensional kd-trees

- A binary tree:
 - Size $O(n)$
 - Depth $O(\log n)$
 - Construction time $O(n \log n)$
 - Query time: worst case $O(n)$, but for many cases $O(\log n)$

Generalizes to d dimensions

Summary

- When to Consider:
 - Instance map to points in \mathbb{R}^n
 - Less than 20 attributes per instance
 - Lots of training data
- Advantages
 - Training is very fast
 - Learn complex target functions
 - Do not lose information
- Disadvantages
 - Slow at query time
 - Easily fooled by irrelevant attributes

NAÏVE BAYES CLASSIFIER

The Bayes Classifier

- A probabilistic framework for solving classification problems
- **A, C** random variables
- Joint probability: $\Pr(A=a, C=c)$
- Conditional probability: $\Pr(C=c | A=a)$
- Relationship between joint and conditional probability distributions

$$\Pr(C, A) = \Pr(C|A) \times \Pr(A) = \Pr(A|C) \times \Pr(C)$$

The Bayes Classifier

- Given:
 - A doctor knows that meningitis causes stiff neck 50% of the time
 - Prior probability of any patient having meningitis is 1/50,000
 - Prior probability of any patient having stiff neck is 1/20
- If a patient has stiff neck, what's the probability he/she has meningitis?

$$P(M|S) = \frac{P(S|M)P(M)}{P(S)} = \frac{0.5 \times 1/50000}{1/20} = 0.0002$$

The Bayes Classifier

- d – a given example
- c - a class

$$c_{MAP} = \operatorname{argmax}_{c \in C} P(c | d)$$

MAP is “maximum a posteriori”
= most likely class

$$= \operatorname{argmax}_{c \in C} \frac{P(d | c)P(c)}{P(d)}$$

Bayes Rule

$$= \operatorname{argmax}_{c \in C} P(d | c)P(c)$$

Dropping the denominator

The Bayes Classifier

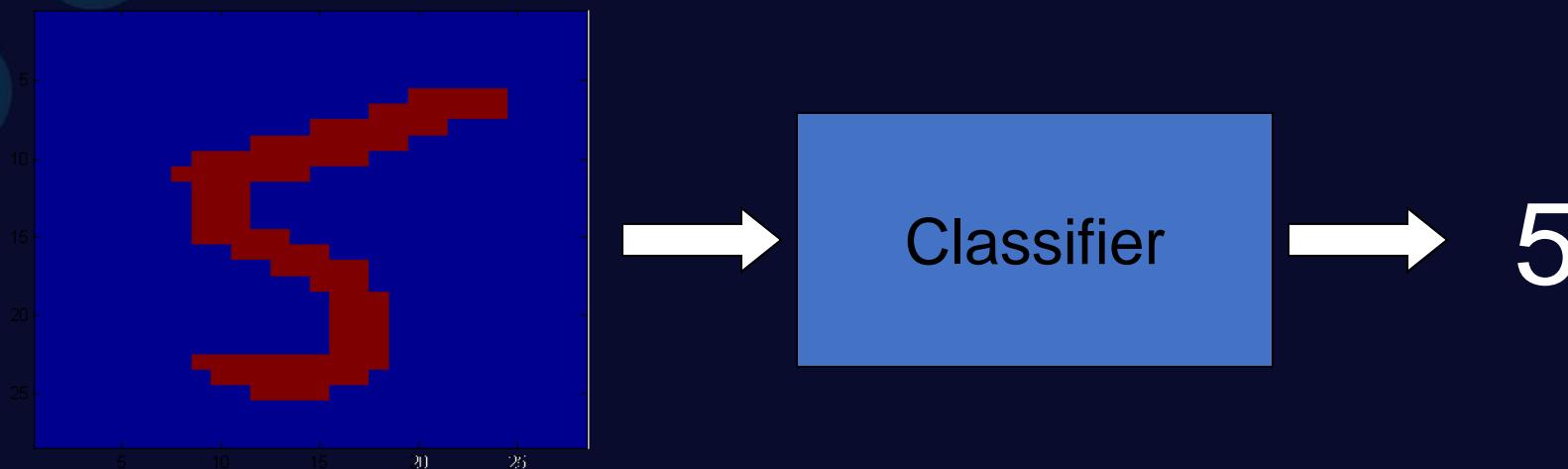
$$c_{MAP} = \operatorname{argmax}_{c \in C} P(d | c)P(c)$$

$$= \operatorname{argmax}_{c \in C} P(x_1, x_2, \square, x_n | c)P(c)$$

Example d
represented as
features x1..xn

Example

- Digit Recognition



- $X_1, \dots, X_n \in \{0,1\}$ (Black vs. White pixels)
- $Y \in \{5,6\}$ (predict whether a digit is a 5 or a 6)

The Bayes Classifier

- As we saw before we will use the Bayes rule

$$P(Y|X_1, \dots, X_n) = \frac{P(X_1, \dots, X_n|Y)P(Y)}{P(X_1, \dots, X_n)}$$

Likelihood Prior
 ↓
 ↓
 Normalization Constant

The Bayes Classifier

- By expanding the formula using the law of total probability we get that.

$$P(Y = 5|X_1, \dots, X_n) = \frac{P(X_1, \dots, X_n|Y = 5)P(Y = 5)}{P(X_1, \dots, X_n|Y = 5)P(Y = 5) + P(X_1, \dots, X_n|Y = 6)P(Y = 6)}$$
$$P(Y = 6|X_1, \dots, X_n) = \frac{P(X_1, \dots, X_n|Y = 6)P(Y = 6)}{P(X_1, \dots, X_n|Y = 5)P(Y = 5) + P(X_1, \dots, X_n|Y = 6)P(Y = 6)}$$

- To classify, we'll simply compute these two probabilities and predict based on which one is greater

Model Parameters

- For the Bayes classifier, we need to “learn” two functions, the likelihood and the prior
- How many parameters are required to specify the likelihood?
 - (Supposing that each image is 30x30 pixels)

Model Parameters

- The problem with explicitly modeling $P(X_1, \dots, X_n | Y)$ is that there are usually way too many parameters:
 - We'll run out of space
 - We'll run out of time
 - And we'll need tons of training data (which is usually not available)

The Naïve Bayes Model

- The *Naïve Bayes Assumption*: Assume that all features are independent **given the class label Y**
- Equationally speaking:

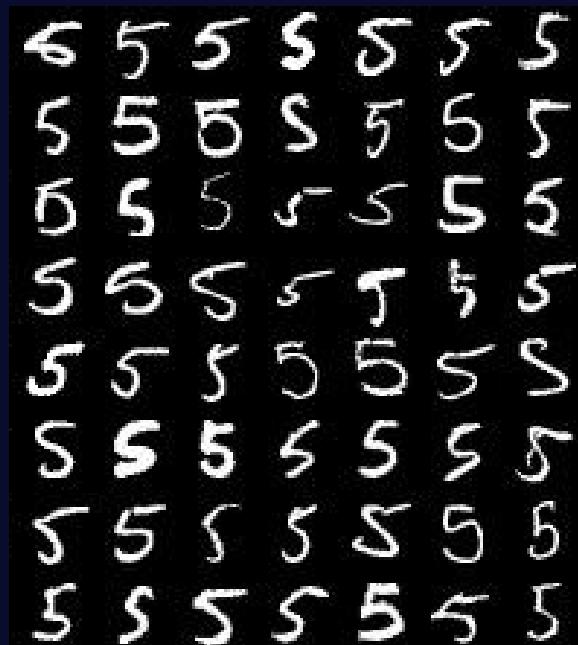
$$P(X_1, \dots, X_n | Y) = \prod_{i=1}^n P(X_i | Y)$$

Why is this useful?

- # of parameters for modeling $P(X_1, \dots, X_n | Y)$:
 - $2(2^n - 1)$
- # of parameters for modeling $P(X_1 | Y), \dots, P(X_n | Y)$
 - $2n$

Naïve Bayes Training

- Now that we've decided to use a Naïve Bayes classifier, we need to train it with some data:



MNIST Training Data

Naïve Bayes Training

- Training in Naïve Bayes is **easy**:
 - Estimate $P(Y=v)$ as the fraction of records with $Y=v$

$$P(Y = v) = \frac{Count(Y = v)}{\# \text{ records}}$$

- Estimate $P(X_i=u|Y=v)$ as the fraction of records with $Y=v$ for which $X_i=u$

$$P(X_i = u|Y = v) = \frac{Count(X_i = u \wedge Y = v)}{Count(Y = v)}$$

- (This corresponds to Maximum Likelihood estimation of model parameters)

Naïve Bayes Training

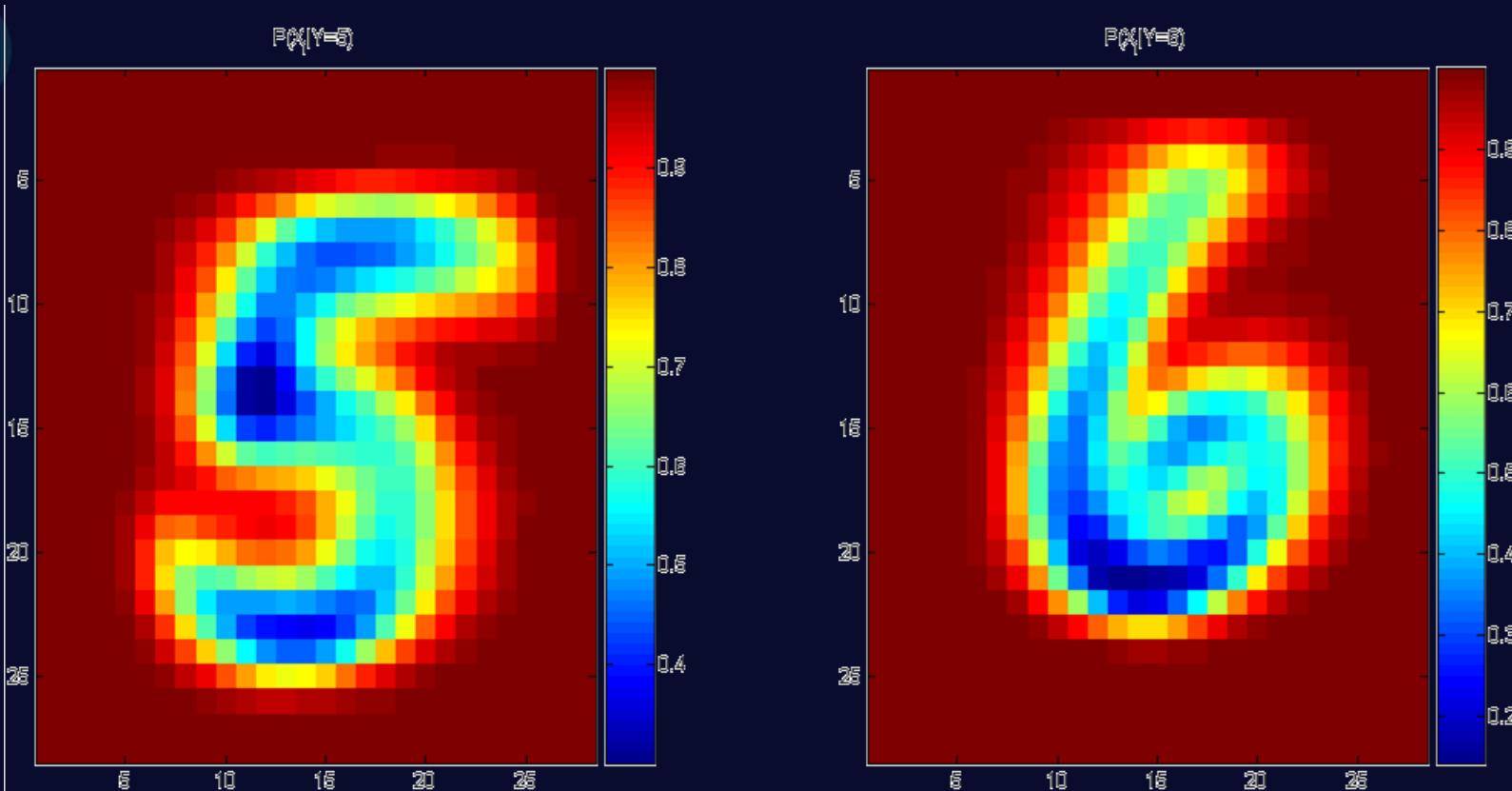
- In practice, some of these counts can be zero
- Fix this by adding “virtual” counts:

$$P(X_i = u|Y = v) = \frac{Count(X_i = u \wedge Y = v) + 1}{Count(Y = v) + 2}$$

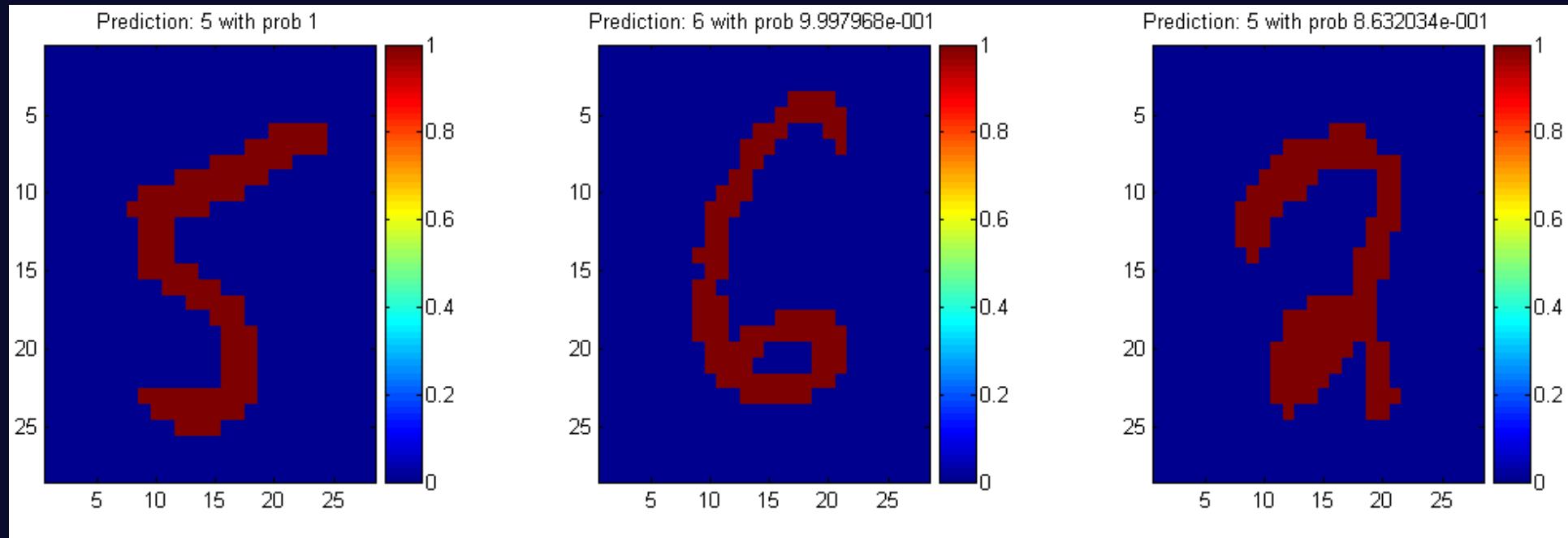
- This is called *Smoothing*

Naïve Bayes Training

- For binary digits, training amounts to averaging all of the training fives together and all of the training sixes together.



Naïve Bayes Classification



Another Example of the Naïve Bayes Classifier

The weather data, with counts and probabilities													
outlook	temperature				humidity				windy		play		
	yes	no	yes	no	yes	no	yes	no	yes	no	yes	no	
sunny	2	3	hot	2	2	high	3	4	false	6	2	9	5
overcast	4	0	mild	4	2	normal	6	1	true	3	3		
rainy	3	2	cool	3	1								
sunny	2/9	3/5	hot	2/9	2/5	high	3/9	4/5	false	6/9	2/5	9/14	5/14
overcast	4/9	0/5	mild	4/9	2/5	normal	6/9	1/5	true	3/9	3/5		
rainy	3/9	2/5	cool	3/9	1/5								

A new day				
outlook	temperature	humidity	windy	play
sunny	cool	high	true	?

- Likelihood of yes

$$= \frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{9}{14} = 0.0053$$

- Likelihood of no

$$= \frac{3}{5} \times \frac{1}{5} \times \frac{4}{5} \times \frac{3}{5} \times \frac{5}{14} = 0.0206$$

- Therefore, the prediction is No

The Naive Bayes Classifier for Data Sets with Numerical Attribute Values

- One common practice to handle numerical attribute values is to assume normal distributions for numerical attributes.

The numeric weather data with summary statistics

outlook		temperature		humidity		windy		play					
	yes	no	yes	no	yes	no	yes	no	yes	no			
sunny	2	3	83	85	86	85	false	6	2	9	5		
overcast	4	0	70	80	96	90	true	3	3				
rainy	3	2	68	65	80	70							
			64	72	65	95							
			69	71	70	91							
			75		80								
			75		70								
			72		90								
			81		75								
sunny	2/9	3/5	mean	73	74.6	mean	79.1	86.2	false	6/9	2/5	9/14	5/14
overcast	4/9	0/5	std dev	6.2	7.9	std dev	10.2	9.7	true	3/9	3/5		
rainy	3/9	2/5											

The Naive Bayes Classifier for Data Sets with Numerical Attribute Values

- Let x_1, x_2, \dots, x_n be the values of a numerical attribute in the training data set.

$$\begin{aligned}\mu &= \frac{1}{n} \sum_{i=1}^n x_i \\ \sigma &= \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \mu)^2} \\ f(w) &= \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(w-\mu)^2}{\sigma^2}}\end{aligned}$$

- For examples,

$$f(\text{temperature} = 66 | \text{Yes}) = \frac{1}{\sqrt{2\pi}(6.2)} e^{-\frac{(66-73)^2}{2(6.2)^2}} = 0.0340$$

- Likelihood of Yes =

$$\frac{2}{9} \times 0.0340 \times 0.0221 \times \frac{3}{9} \times \frac{9}{14} = 0.000036$$

- Likelihood of No =

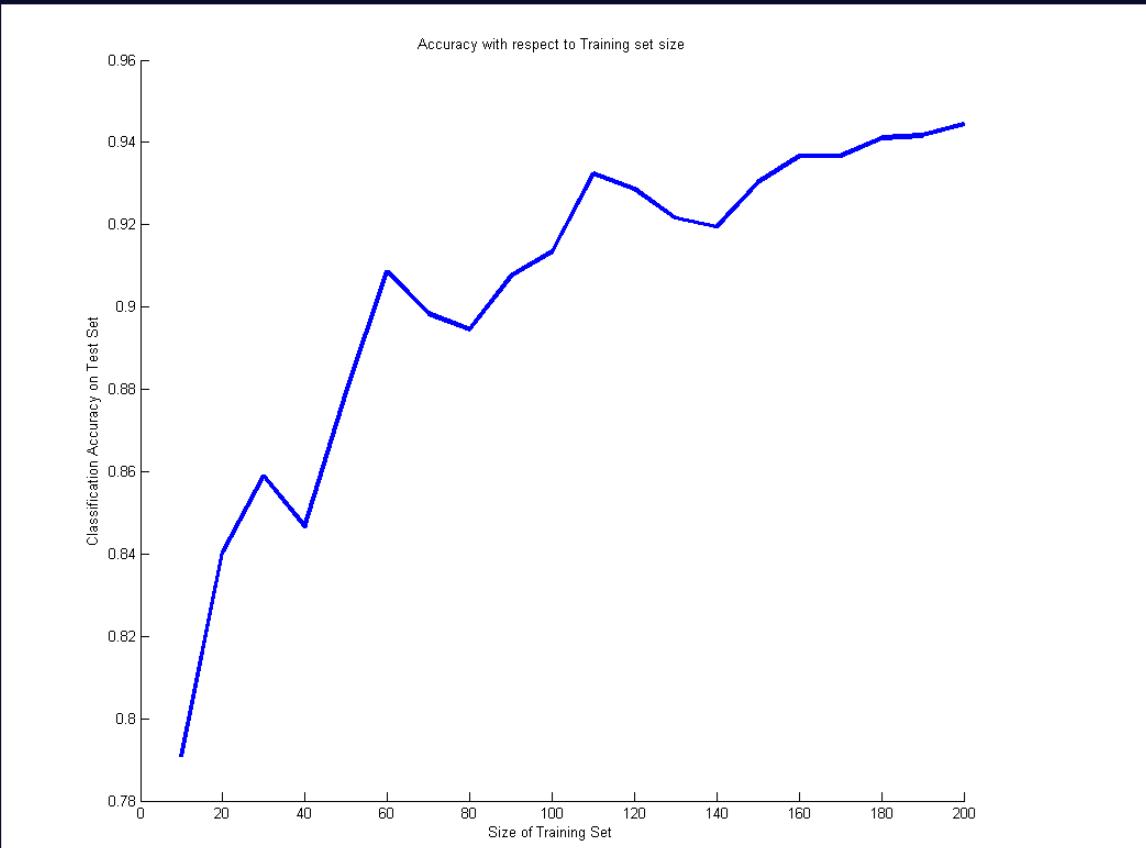
$$\frac{3}{5} \times 0.0291 \times 0.038 \times \frac{3}{5} \times \frac{5}{14} = 0.000136$$

Outputting Probabilities

- What's nice about Naïve Bayes (and generative models in general) is that it returns probabilities
 - These probabilities can tell us how confident the algorithm is
 - So... don't throw away those probabilities!

Performance on a Test Set

- Naïve Bayes is often a good choice if you don't have much training data!



Naïve Bayes Assumption

- Recall the Naïve Bayes assumption:
 - that all features are independent **given the class label Y**
- Does this hold for the digit recognition problem?

Exclusive-OR Example

- For an example where conditional independence fails:
 - $Y = \text{XOR}(X_1, X_2)$

X_1	X_2	$P(Y=0 X_1, X_2)$	$P(Y=1 X_1, X_2)$
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	0

Naïve Bayes Assumption

- Actually, the Naïve Bayes assumption is almost never true
- Still... Naïve Bayes often performs surprisingly well even when its assumptions do not hold

Numerical Stability

- It is often the case that machine learning algorithms need to work with very small numbers
 - Imagine computing the probability of 2000 independent coin flips
 - MATLAB thinks that $(.5)^{2000}=0$

Numerical Stability

- Multiplying lots of probabilities
floating-point underflow.
- Recall: $\log(xy) = \log(x) + \log(y)$,

better to sum logs of probabilities rather than multiplying probabilities.

Numerical Stability

- Class with highest final un-normalized log probability score is still the most probable.

$$c_{NB} = \operatorname{argmax}_{c_j \in C} \log P(c_j) + \sum_{i \in positions} \log P(x_i | c_j)$$

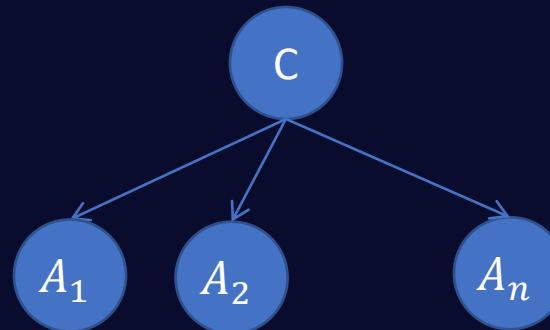
Numerical Stability

- Instead of comparing $P(Y=5|X_1, \dots, X_n)$ with $P(Y=6|X_1, \dots, X_n)$,
 - Compare their logarithms

$$\begin{aligned}\log(P(Y|X_1, \dots, X_n)) &= \log\left(\frac{P(X_1, \dots, X_n|Y) \cdot P(Y)}{P(X_1, \dots, X_n)}\right) \\ &= \text{constant} + \log\left(\prod_{i=1}^n P(X_i|Y)\right) + \log P(Y) \\ &= \text{constant} + \sum_{i=1}^n \log P(X_i|Y) + \log P(Y)\end{aligned}$$

Generative models

- Naïve Bayes is a type of a generative model
 - Generative process:
 - First pick the category of the record
 - Then given the category, generate the attribute values from the distribution of the category



- Conditional independence given C
- We use the training data to learn the distribution of the values in a class

Recap

- We defined a *Bayes classifier* but saw that it's intractable to compute $P(X_1, \dots, X_n | Y)$
- We then used the *Naïve Bayes assumption* – that everything is independent given the class label Y
- Naïve Bayes is:
 - Really easy to implement and often works well
 - Often a good first thing to try
 - Commonly used as a “punching bag” for smarter algorithms

Evaluation of Learning Models

Motivation

- It is important to evaluate classifier's generalization performance in order to:
 - Determine whether to employ the classifier
 - Compare classifiers
 - Optimize the classifier

Metrics for Classifier's Evaluation

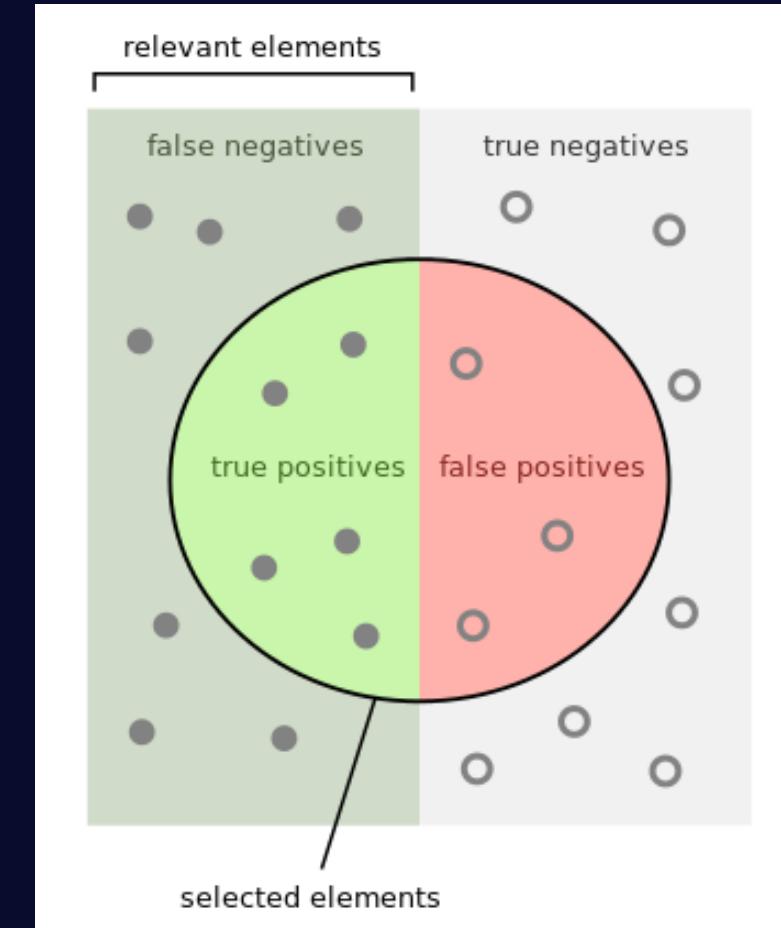
- Accuracy = $\frac{TP+TN}{P+N} = \frac{\text{green}}{\text{green} + \text{red}}$

- Error = $\frac{FP+FN}{P+N} = \frac{\text{green} + \text{red}}{\text{green} + \text{red} + \text{grey}}$

- Precision = $\frac{TP}{TP+FP} = \frac{\text{green}}{\text{green} + \text{red}}$

- Recall/TP rate = $\frac{TP}{P} = \frac{\text{green}}{\text{green} + \text{grey}}$

- FP rate = $\frac{FP}{N} = \frac{\text{red}}{\text{green} + \text{grey}}$



Metrics for Classifier's Evaluation

		<i>Predicted class</i>	
		Pos	Neg
<i>Actual class</i>	Pos	TP	FN
	Neg	FP	TN

F_1 score

- The F_1 score is the harmonic average of the precision and recall

$$F_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

How to Estimate the Metrics?

- We can use:
 - Training data;
 - Independent test data;
 - Hold-out method;
 - k -fold cross-validation method;
 - Leave-one-out method;
 - Bootstrap method;
 - And many more...

Estimation with Training Data

- The accuracy/error estimates on the training data are *not* good indicators of performance on future data.



- **Q: Why?**
- **A:** Because new data will probably not be **exactly** the same as the training data!

Estimation with Independent test data

- Estimation with independent test data is used.

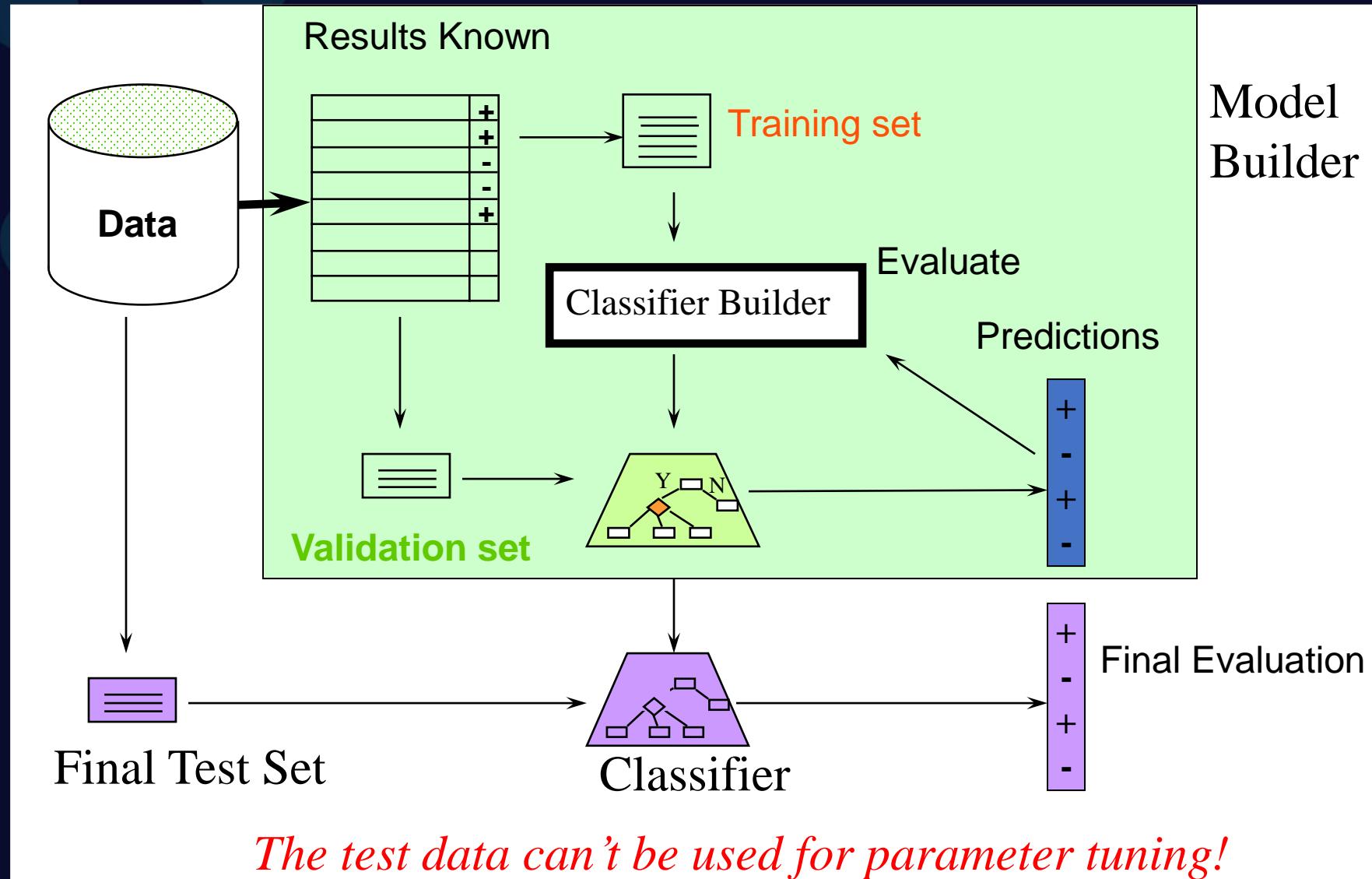


Hold-out Method

- The hold-out method splits the data into training data and test data. Then we build a classifier using the train data and test it using the test data.



Classification: Train, Validation, Test Split



Making the Most of the Data

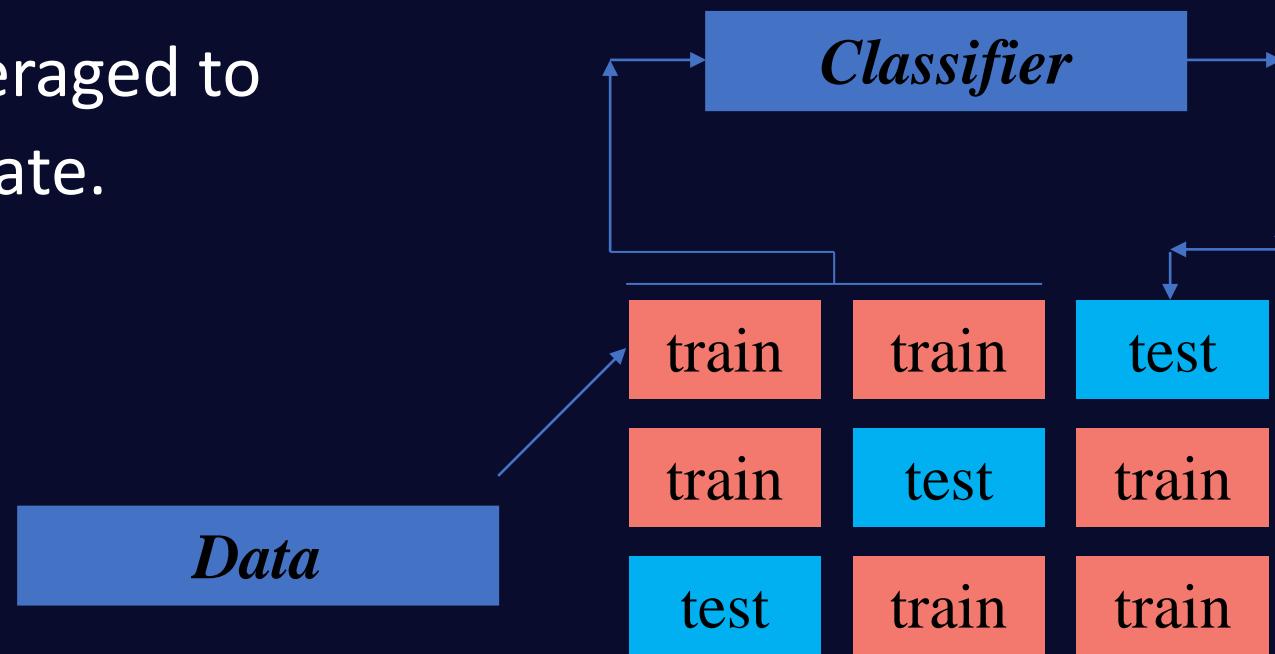
- Once evaluation is complete, *all the data* can be used to build the final classifier.
- Generally, the **larger** the training data the **better** the classifier (but returns diminish).
- The **larger** the test data the more accurate the error estimate.

Stratification

- The *holdout* method reserves a certain amount for testing and uses the remainder for training.
- For “unbalanced” datasets, samples might not be representative.
 - *Few or none instances of some classes.*
- **Stratified sample:** balancing the data.
 - *Make sure that each class is represented with approximately equal proportions in both subsets.*

k-Fold Cross-Validation

- *k-fold cross-validation*:
 - *First step*: data is split into k subsets of equal size;
 - *Second step*: each subset in turn is used for testing and the remainder for training.
- The estimates are averaged to yield an overall estimate.



More on Cross-Validation

- Standard method for evaluation: stratified 10-fold cross-validation.
- Stratification reduces the estimate's variance.
- Even better: repeated stratified cross-validation:
 - E.g. ten-fold cross-validation is repeated ten times and results are averaged (reduces the variance).

K-folds from what?

- Which dataset should we do k-fold on? Training? Training+validation? Everything?
- The *test set* should be put aside and it cannot be used for ANY purpose (including data prep).
- The *validation set* is used to test performances of the models at the various stages of the model development.
 - Different models, various versions of the same model, etc.
 - A good practice is to use the validation set as little as possible (to avoid "mental" overfit)
- *Train set* is used for training. This includes cross validation.

Leave-One-Out Cross-Validation

- Leave-One-Out is a particular form of cross-validation:
 - Set number of folds to number of training instances;
 - I.e., for n training instances, build classifier n times.
- Makes best use of the data.
- Involves no random sub-sampling.

Very computationally expensive.

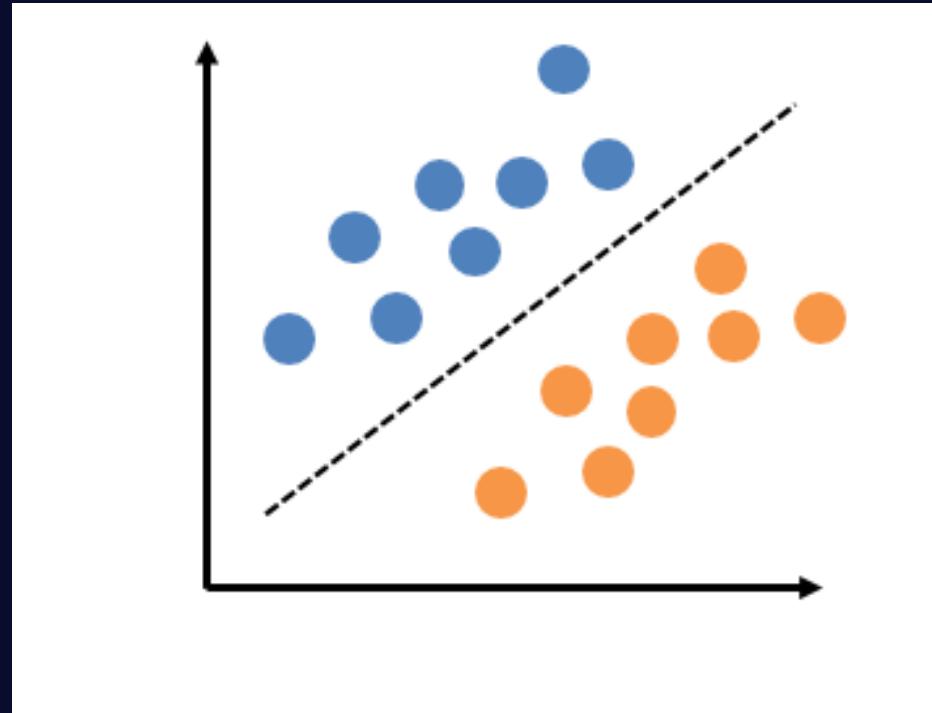
Leave-One-Out Cross-Validation and Stratification

- A disadvantage of Leave-One-Out-CV is that stratification is not possible:
 - It *guarantees* a non-stratified sample because there is only one instance in the test set!
- Extreme example - random dataset split equally into two classes:
 - Best inducer predicts majority class;
 - 50% accuracy on fresh data;
 - Leave-One-Out-CV estimate is 100% error!

Pre-processing steps on validation set?

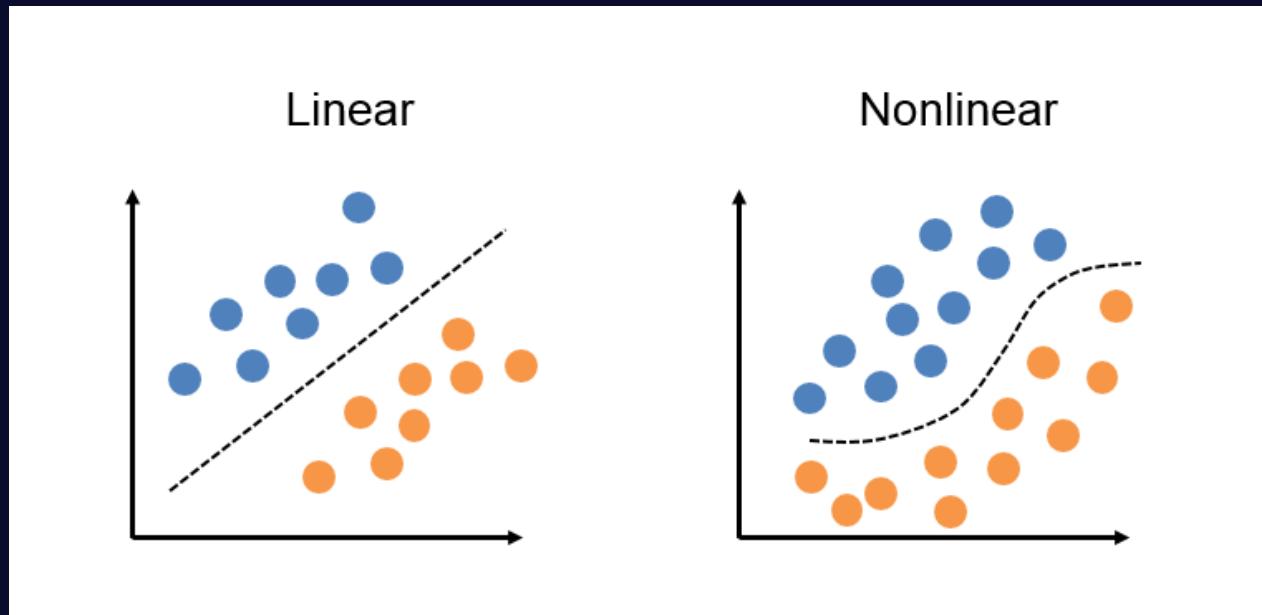
- Whatever data prep steps done on training data, should be able to be applied (not redesign) at the final test on unseen data.
 - For example, if normalizing a feature with a mean, the mean is fixed at the data prep and not recalculated. Otherwise, a model that was trained and tested on data that used the former mean may not be adequate
- **Q:** So should we use the validation set for pre-processing?
- **A:** There are quite a few opinions and no consensus.
Both options are valid and the decision is mainly based on the stability of the statistics (measures) that can be extracted from Train vs. Train+Validation.
- Data preparation steps are mostly based on descriptive statistics. So a **larger sample** that provides a **stable statistics** is an advantage.
- By not using the validation set, we have a **better estimate of the “true error”**, which we measure by applying pre-processing steps to test data.
- By using the validation set, we **separate the effect of the pre-processing from that of the classifier**

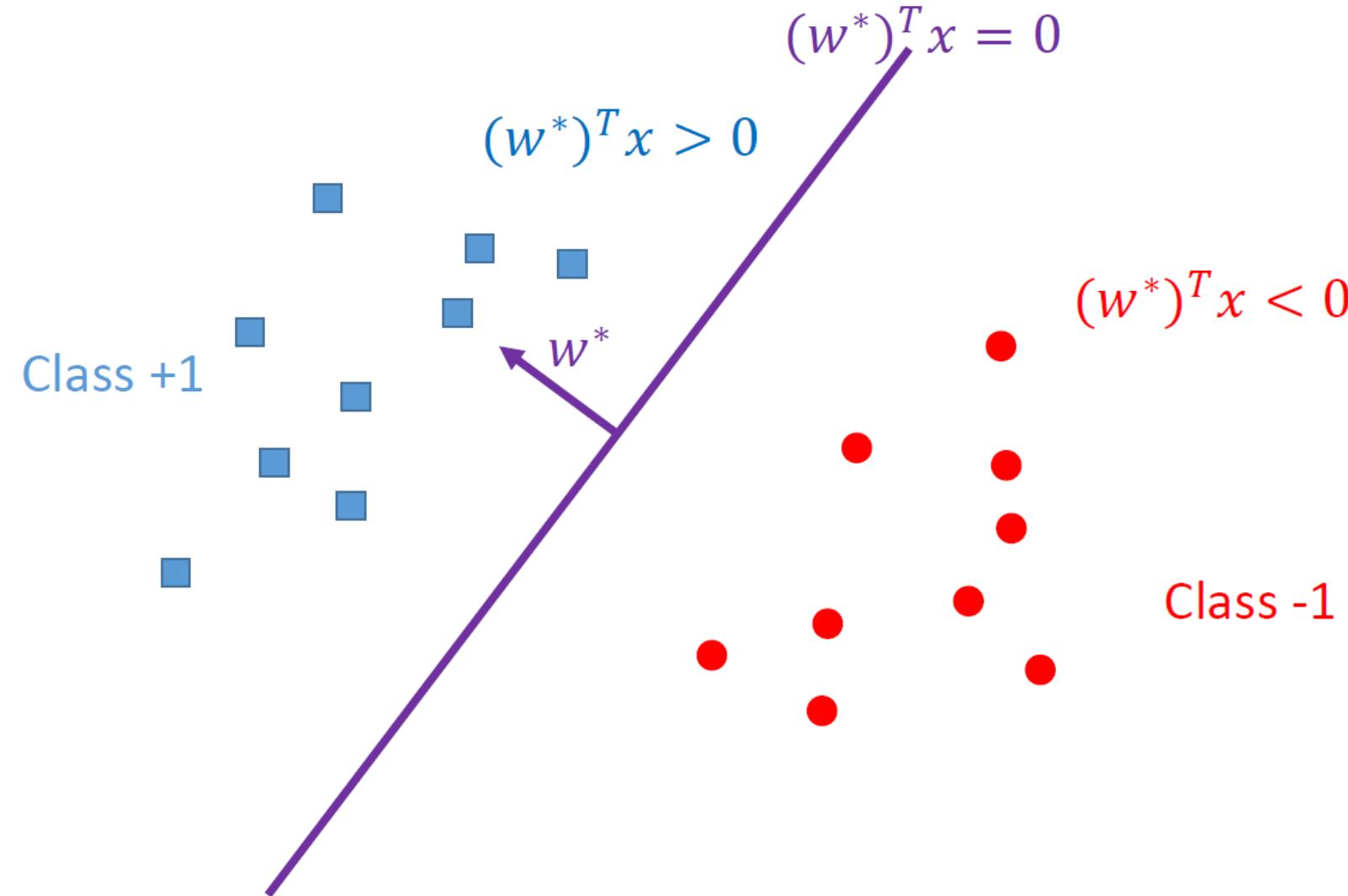
Linear models



Linear models

- Definition:
- Methods that give linear decision boundaries between classes $\{x | w^T x + w_0 = 0\}$



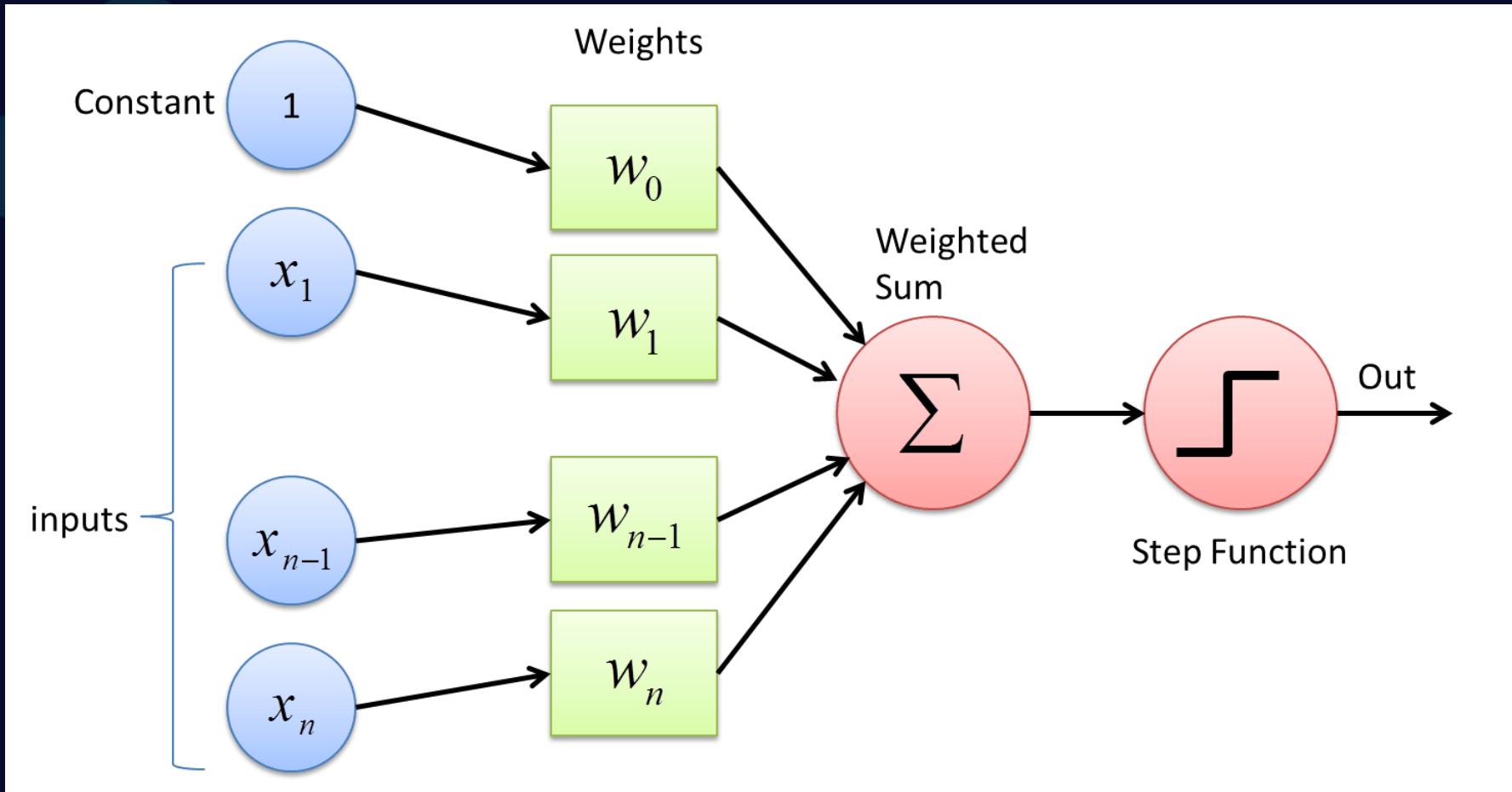


Linear models

- Given a training data $\{(x_i, y_i): 1 \leq i \leq n\}$ sampled i.i.d from source D.
- Our hypothesis will be:
 - $y = +1$ if $w^T x > 0$
 - $y = -1$ if $w^T x < 0$

$$y = \text{sign}(f_w(x)) = \text{sign}(w^T x)$$

Perceptron Algorithm



Perceptron Algorithm

1. Start with the all-zeroes weight vector $\mathbf{w}_1 = \mathbf{0}$, and initialize t to 1.
 2. Given example \mathbf{x} , predict positive iff $\mathbf{w}_t \cdot \mathbf{x} > 0$.
 3. On a mistake, update as follows:
 - Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}$.
 - Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}$.
- $t \leftarrow t + 1$.

Perceptron Algorithm

- In case of mistake on positive example

$$w_{t+1}^T x = (w_t + x)^T x = w_t^T x + x^T x = w_t^T x + 1$$

- In case of mistake on negative example

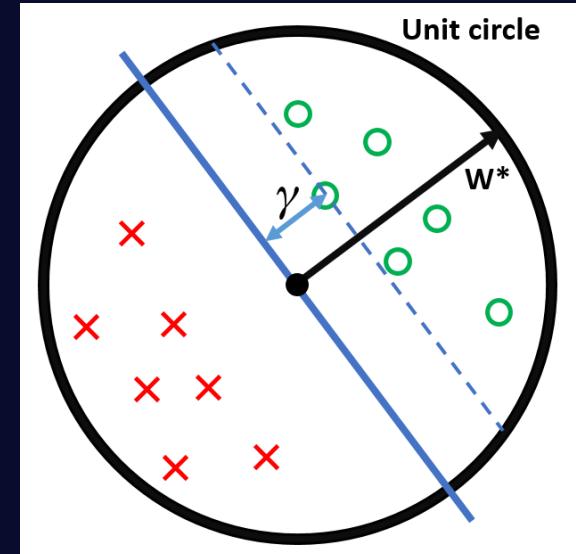
$$w_{t+1}^T x = (w_t - x)^T x = w_t^T x - x^T x = w_t^T x - 1$$

Perceptron Algorithm

- The Perceptron was arguably the first algorithm with a strong formal guarantee. If a data set is linearly separable, the Perceptron will find a separating hyperplane in a finite number of updates. (If the data is not linearly separable, it will loop forever.)

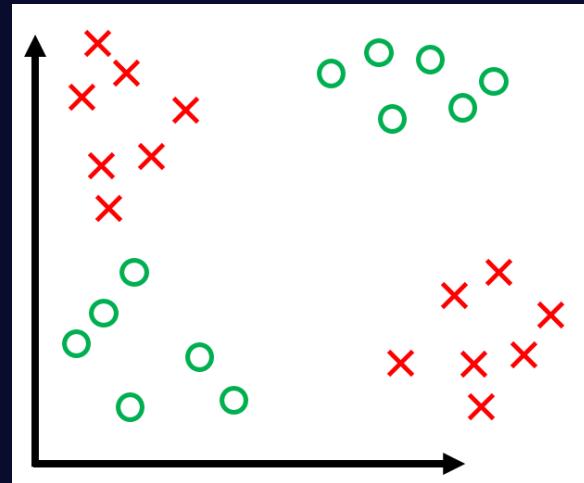
Perceptron Algorithm – Guarantee

- All inputs live within the unit sphere
 - There exists a separating hyperplane defined by w^* , with $\|w^*\|=1$ (i.e. w^* lies exactly on the unit sphere).
 - γ is the distance from this hyperplane (blue) to the closest data point.
- If all the above holds, then the Perceptron algorithm makes at most $\frac{1}{\gamma^2}$ updates



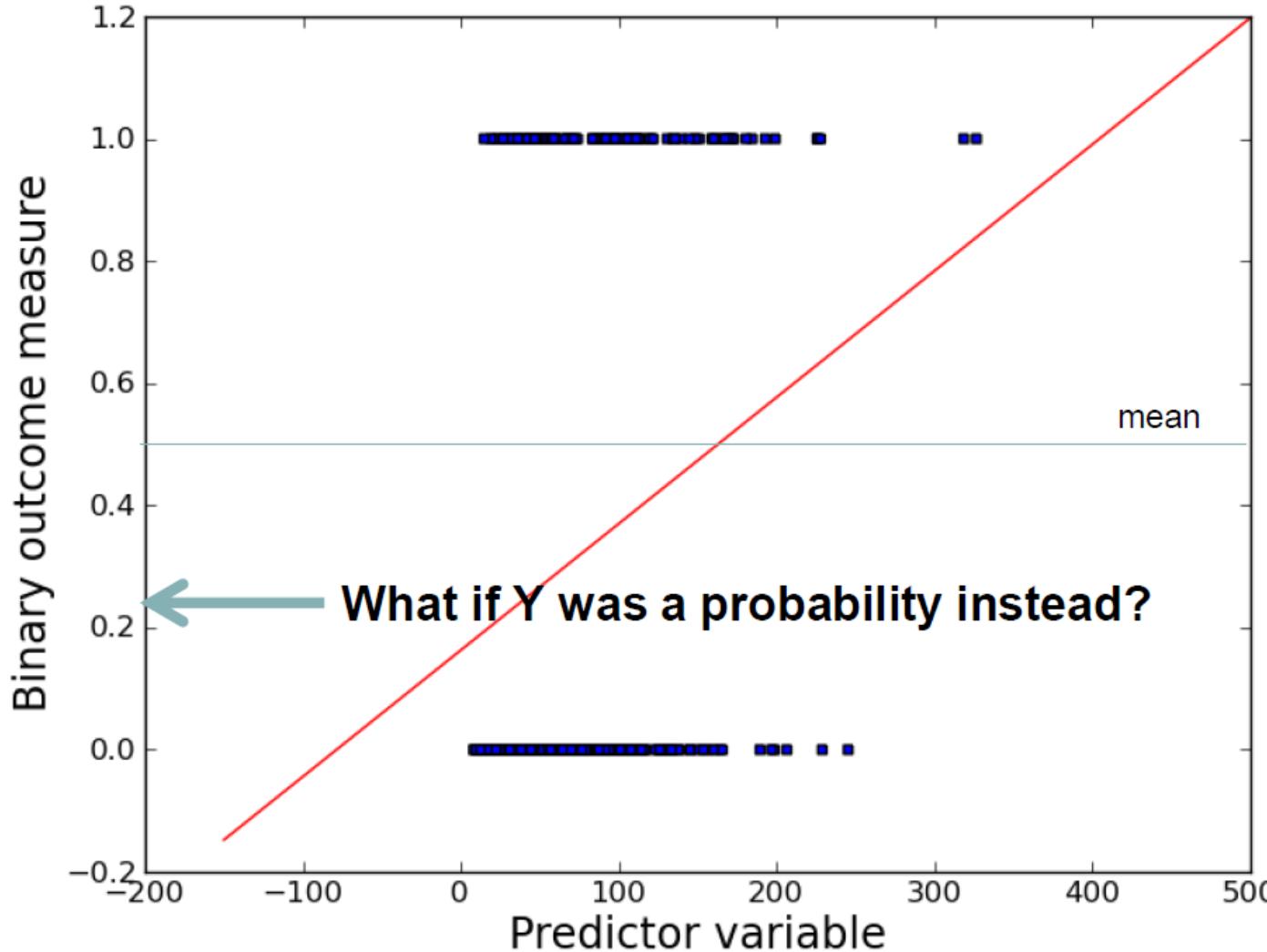
Perceptron Algorithm

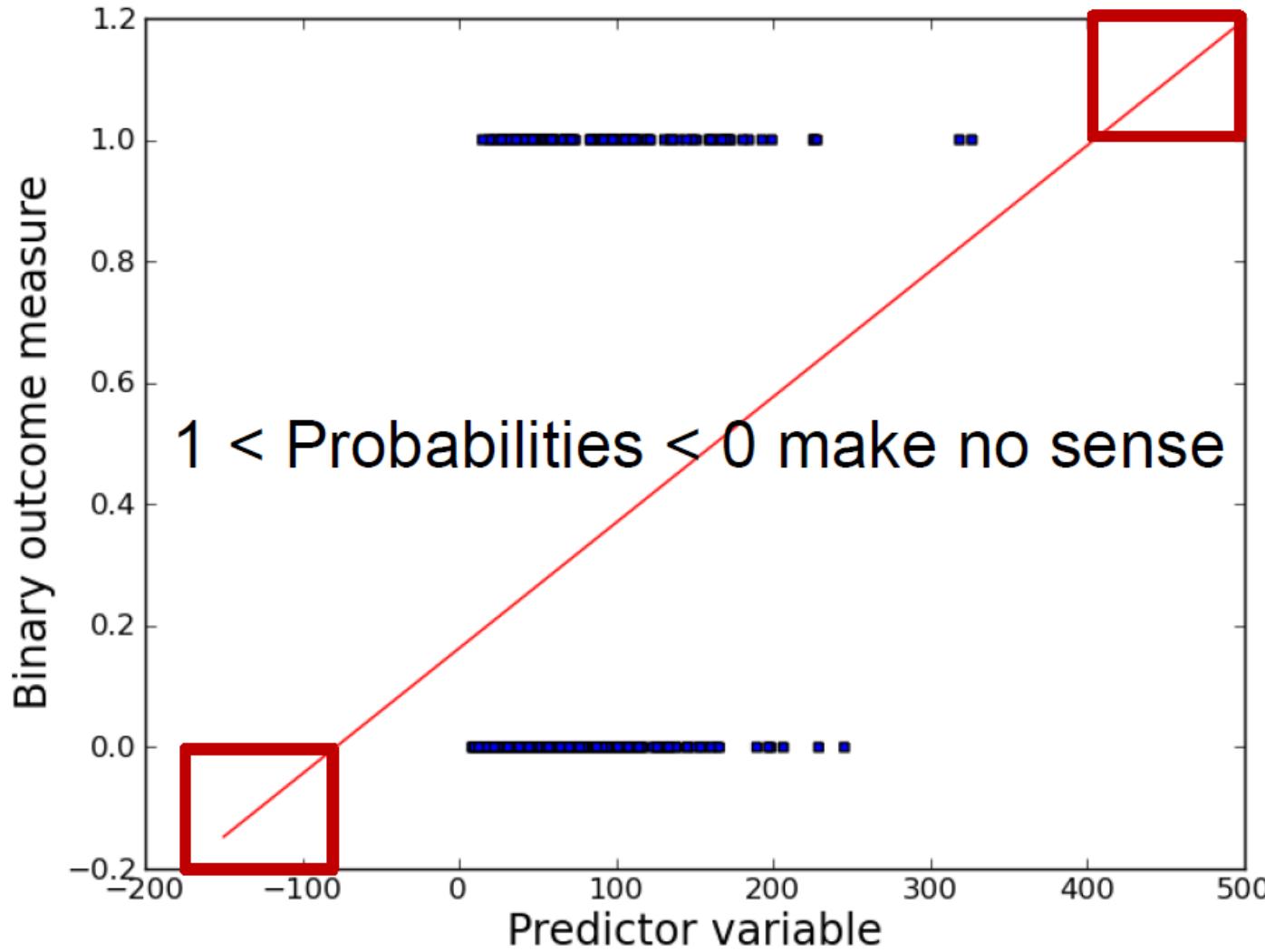
- Initially, huge wave of excitement ("Digital brains") (See The New Yorker December 1958)
- Then, contributed to the A.I. Winter. Famous example of a simple non-linearly separable data set, the XOR problem (Minsky 1969):





Logistic Regression





Logistic Regression

- A danger of "saturation": once we predict a patient has a 100% chance of survival, we're maxed out and no new fact about the patient could improve their odds.

Logistic Regression

- The trick is that we can transform the outcome variable to something which we can use linear regression on.
 - This would have a valid range of values in the range –infinity to + infinity.
 - But still be related to the “probability”.

Odds

- Say I have a bag of 100 marbles
20 are blue, 10 are red 10 are green, 60 yellow.
20% probability of picking a blue one
20 chances of picking a blue one
80 chances of pick another color
- So the odds of picking a blue marble are $20/80 = 2/8 = 0.25$

Odds

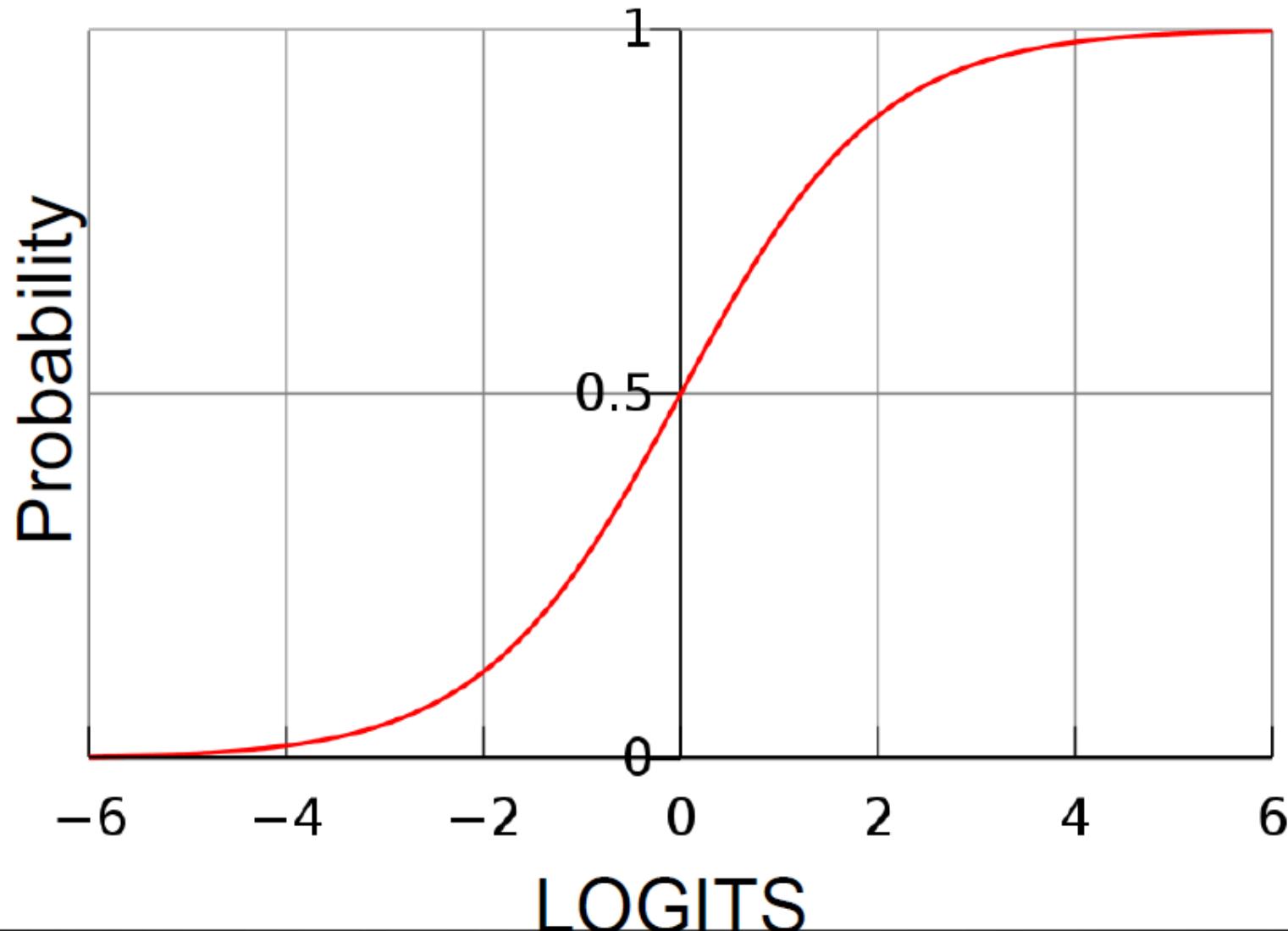
- Another way of thinking about probabilities is to transform them using the function:
 - Odds = $p / (1 - p) = 0.2/0.8 = 0.25$
- This is the probability of something happening divided by the probability of it not happening

Odds

- Odds are commonly used in gambling, especially horse-racing.
 - "9 to 1 against", meaning a probability of 0.1.
 - "Even odds", meaning $p = 0.5$.
 - "3 to 1 on" meaning $p = 0.75$.
 - "8 to 2 against" meaning odds = 0.25
 - "4 to 1 against" meaning odds = 0.25

Odds

- Odds give us a value that ranges from 0 (for very small probabilities) to positive infinity (for probabilities close to 1.0).
- With one more transformation we can get a value that is unbounded over the real numbers.
- This is the logistic function: $y = \log(p / (1-p))$.
- y is measured in logits.



Logistic Regression

- Probabilities transformed through the logistic function are known as "logit" values.
- We now have a value that ranges from negative infinity to positive infinity: ideal for linear regression.

Logistic Regression

$$\log\left(\frac{P(y_1|x)}{P(y_0|x)}\right) = w_0 + x^T w$$

- Criterion:

$$E(w) = - \sum_{i=1}^n y_i \log \sigma(w^T x) + (1 - y_i) \log(1 - \sigma(w^T x))$$

- Training: GD
- Gradient: $w_{t+1} = x_i^T (\pi_i - y_i)$

Multi-class logistic regression

- Following a similar derivation for a **categorical distribution** we get:

$$p(y = c|x, \theta) = \frac{e^{\beta_c^T x + \gamma_c}}{\sum_{c'} e^{\beta_{c'}^T x + \gamma_{c'}}}$$

- Where the multi-categorical loss function is given by

$$E(w) = - \sum_{i=1}^n \sum_{c=1}^C y_i \log p(y = c|x_i, \theta)$$

- This distribution is known as the softmax function or the Boltzmann distribution



Linear Discriminant Analysis

Linear Discriminant Analysis (LDA)

- Linear discriminant analysis (LDA) takes a different approach to classification than logistic regression. Rather than attempting to model the conditional distribution of Y given X , $P(Y = k|X = x)$, LDA models the distribution of the predictors X given the different categories that Y takes on, $P(X = x|Y = k)$.
- In order to flip these distributions around to model $P(X = x|Y = k)$ an analyst uses Bayes' theorem.
- In this setting with one feature (one X), Bayes' theorem can then be written as:

$$P(Y = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{j=1}^K f_j(x)\pi_j}$$

LDA

$$P(Y = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{j=1}^K f_j(x)\pi_j}$$

- The left hand side, $P(Y = k|X = x)$, is called the *posterior* probability and gives the probability that the observation is in the k^{th} category given the feature, X , takes on a specific value, x . The numerator on the right is conditional distribution of the feature within category k , $f_k(x)$, times the *prior* probability that observation is in the k^{th} category.
- The *Bayes' classifier* is then selected. That is the observation assigned to the group for which the posterior probability is the largest.

LDA for one predictor

- LDA has the simplest form when there is just one predictor/feature ($p = 1$). In order to estimate $f_k(x)$, we have to assume it comes from a specific distribution.
- One common assumption is that $f_k(x)$ comes from a Normal distribution:

$$f_k(x) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(x - \mu_k)^2}{2\sigma_k^2}\right)$$

- The distribution of the feature X within category k is Normally distributed with mean μ_k and variance σ_k^2 .

LDA for one predictor

- An extra assumption that the variances are equal,

$\sigma_1^2 = \sigma_2^2 = \dots = \sigma_K^2$ will simplify our lives.

- Plugging this assumed likelihood into the Bayes' formula (to get the posterior) results in:

$$P(Y = k|X = x) = \frac{\pi_k \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma^2}\right)}{\sum_{j=1}^K \pi_j \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_j)^2}{2\sigma^2}\right)}$$

- The Bayes classifier will be the one that maximizes this over all values chosen for x . How should we maximize?
- So we take the log of this expression and rearrange to simplify our maximization...

LDA for one predictor

- So we maximize the following simplified expression:

$$\delta_k(x) = x \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log \pi_k$$

- How does this simplify if we have just two classes ($K = 2$) and if we set our prior probabilities to be equal?
- This is equivalent to choosing a decision boundary for x for which

$$x = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)} = \frac{\mu_1 + \mu_2}{2}$$

LDA for one predictor

- In practice we don't know the true mean, variance, and prior. So we estimate them with the classical estimates, and plug-them into the expression:

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i$$

- and

$$\hat{\sigma}^2 = \frac{1}{n - K} \sum_{k=1}^K \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2$$

- where n is the total sample size and n_k is the sample size within class k (thus, $n = \sum n_k$).

LDA for one predictor

- This classifier works great if the classes are about equal in proportion, but can easily be extended to unequal class sizes.
- Instead of assuming all priors are equal, we instead set the priors to match the 'prevalence' in the data set:

$$\hat{\pi}_k = \hat{n}_k/n$$

- Note: we can use a prior probability from knowledge of the subject as well; for example, if we expect the test set to have a different prevalence than the training set.

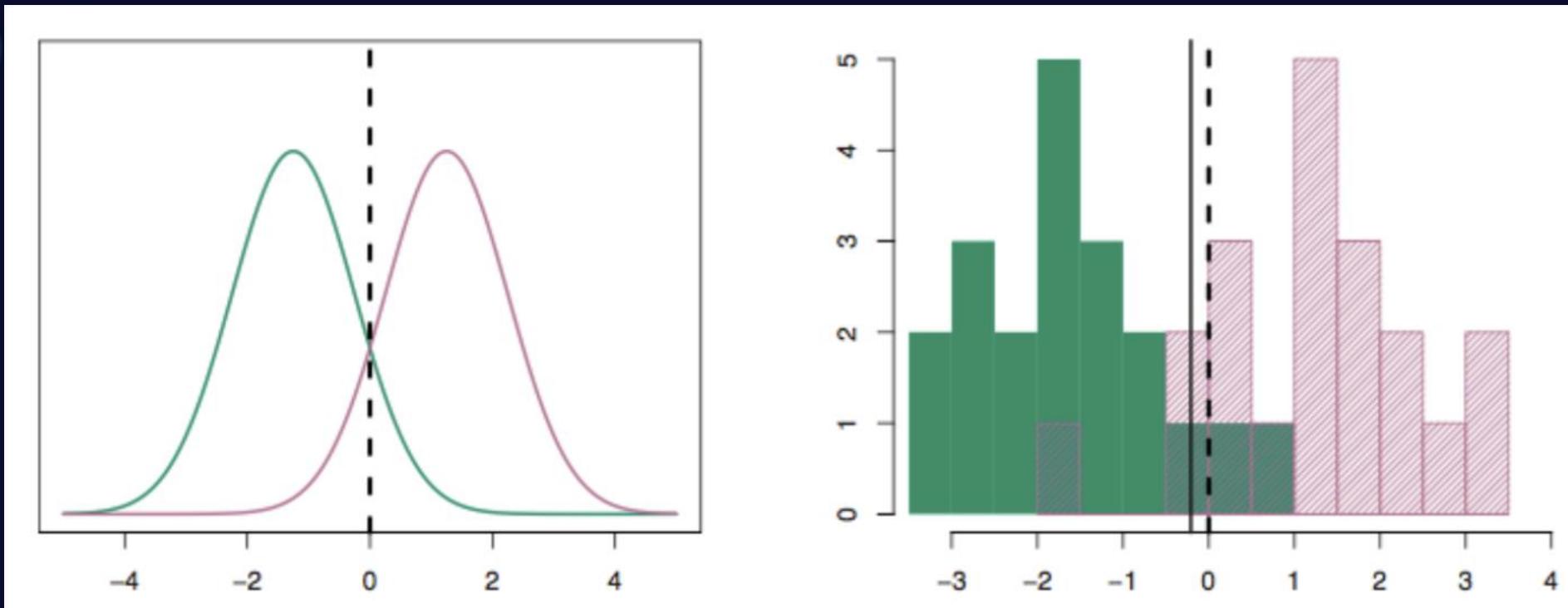
LDA for one predictor

- Plugging all of these estimates back into the original logged maximization formula we get:

$$\hat{\delta}_k(x) = x \frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log \hat{\pi}_k$$

- Thus this classifier is called the linear discriminant classifier: this discriminant function is a linear function of x .

Illustration of LDA when $p = 1$

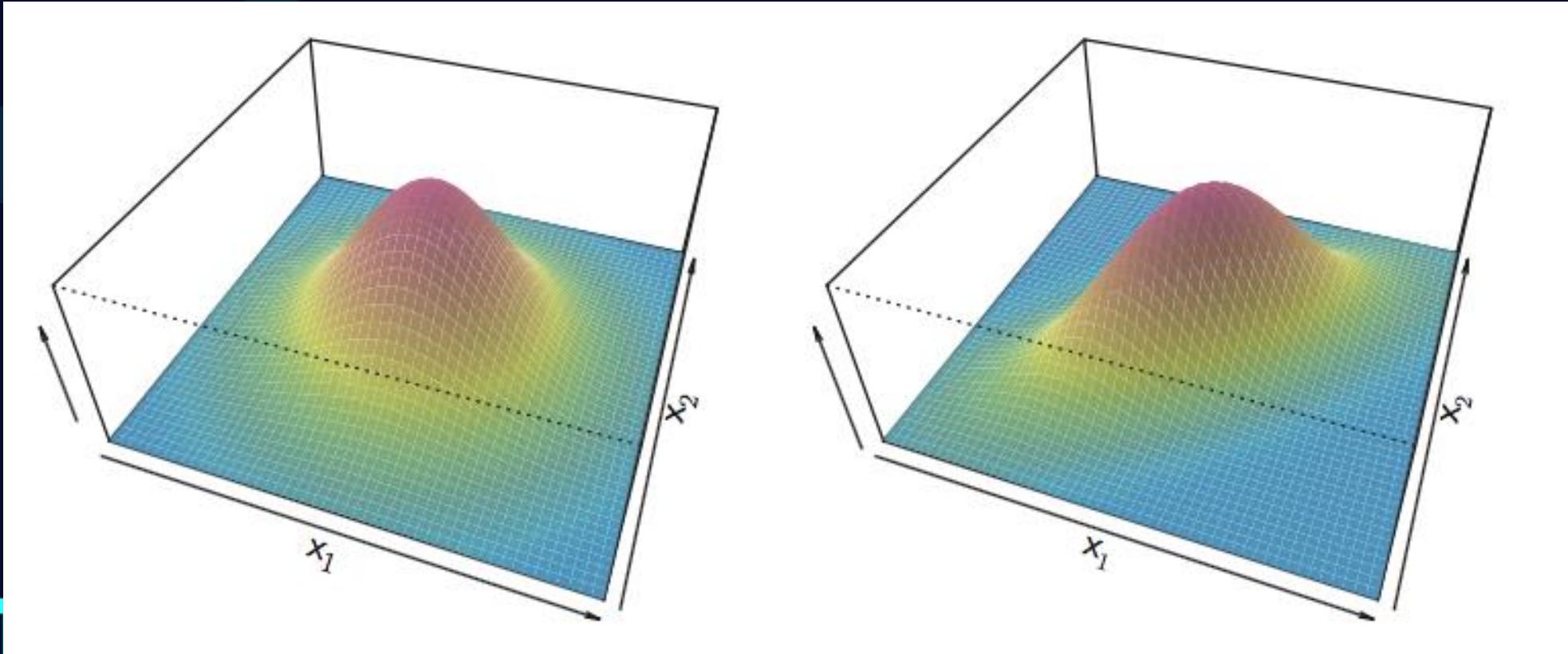


LDA when $p > 1$

- Instead of assuming the one predictor is Normally distributed, it assumes that the set of predictors for each class is 'multivariate normal distributed' (shorthand: MVN). What does that mean?
- This means that the vector of X for an observation has a multidimensional normal distribution with a mean vector, μ , and a covariance matrix, Σ .

Multivariate Normal Distribution

- Here is a visualization of the Multivariate Normal distribution with 2 variables:



MVN Distribution

- The joint PDF of the Multivariate Normal distribution, $\vec{X} \sim MVN(\vec{\mu}, \Sigma)$ is:

$$f(\vec{x}) = \frac{1}{2\pi^{p/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu})\right)$$

- where \vec{X} is a p dimensional vector and $|\Sigma|$ is the determinant of the $p \times p$ covariance matrix.
- Let's do a quick dimension analysis sanity check...
 - What do \vec{X} and Σ look like?

LDA when $p > 1$

- Discriminant analysis in the multiple predictor case assumes the set of predictors for each class is then multivariate Normal:

$$\vec{X} \sim MVN(\vec{\mu}, \Sigma)$$

- Just like with LDA for one predictor, we make an extra assumption that the covariances are equal in each group, $\Sigma_1 = \Sigma_2 = \dots = \Sigma_K$. in order to simplify our lives.
- Now plugging this assumed likelihood into the Bayes' formula (to get the posterior) results in:

$$P(Y = k | \vec{X} = \vec{x}) = \frac{\pi_k \frac{1}{2\pi^{p/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu}_k)^T \Sigma^{-1} (\vec{x} - \vec{\mu}_k)\right)}{\sum_{j=1}^K \frac{1}{2\pi^{p/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu}_j)^T \Sigma^{-1} (\vec{x} - \vec{\mu}_j)\right)}$$

LDA when $p > 1$

- Then doing the same steps as before (taking log and maximizing), we see that the classification will for an observation based on its predictors, \vec{x} , will be the one that maximizes (maximum of K of these $\delta_k(\vec{x})$):

$$\delta_k(\vec{x}) = \vec{x}^T \Sigma^{-1} \vec{\mu}_k - \frac{1}{2} \vec{\mu}_k^T \Sigma^{-1} \vec{\mu}_k + \log \pi_k$$

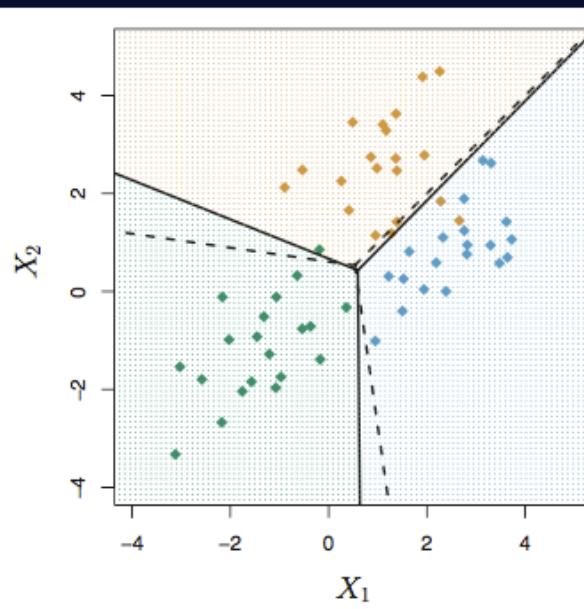
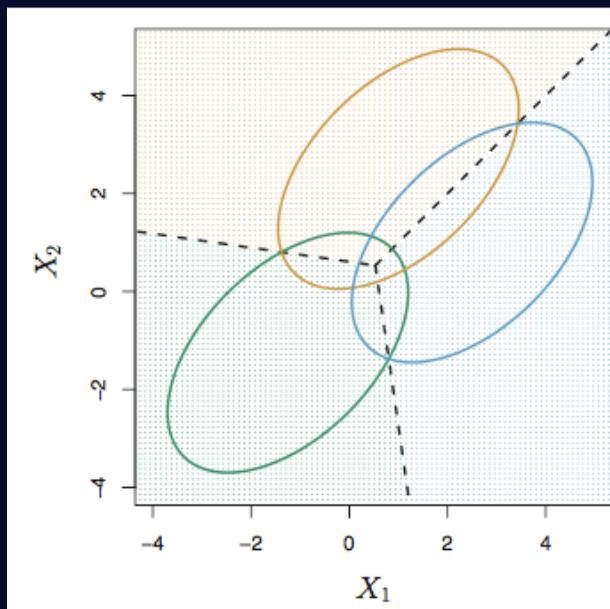
- Note: this is just the vector-matrix version of the formula we saw earlier in lecture:

$$\delta_k(x) = x \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log \pi_k$$

- What do we have to estimate now with the vector-matrix version? How many parameters are there?
- There are pK means, pK variances, K prior proportions, and $\binom{p}{2} = \frac{p(p-1)}{2}$ covariances to estimate.

LDA when $K > 2$

- The linear discriminant nature of LDA still holds not only when $p > 1$, but also when $K > 2$ for that matter as well. A picture can be very illustrative:



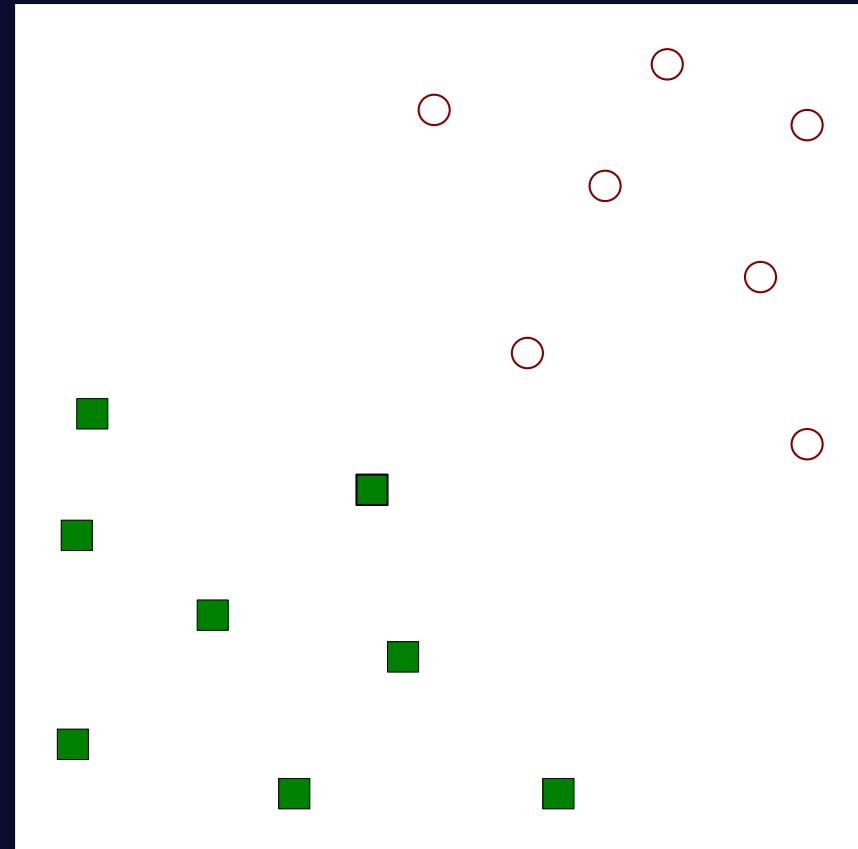
Diagonal LDA

- Diagonal LDA adds another assumption that the covariance matrix is diagonal
- \Rightarrow features are independent
- This model is also a private case of *Gaussian Naïve Bayes*
- *Estimation of parameters is identical to LDA though with diagonal assumption – i.e. solve d one dimensional MLE problems for each dimension independently*



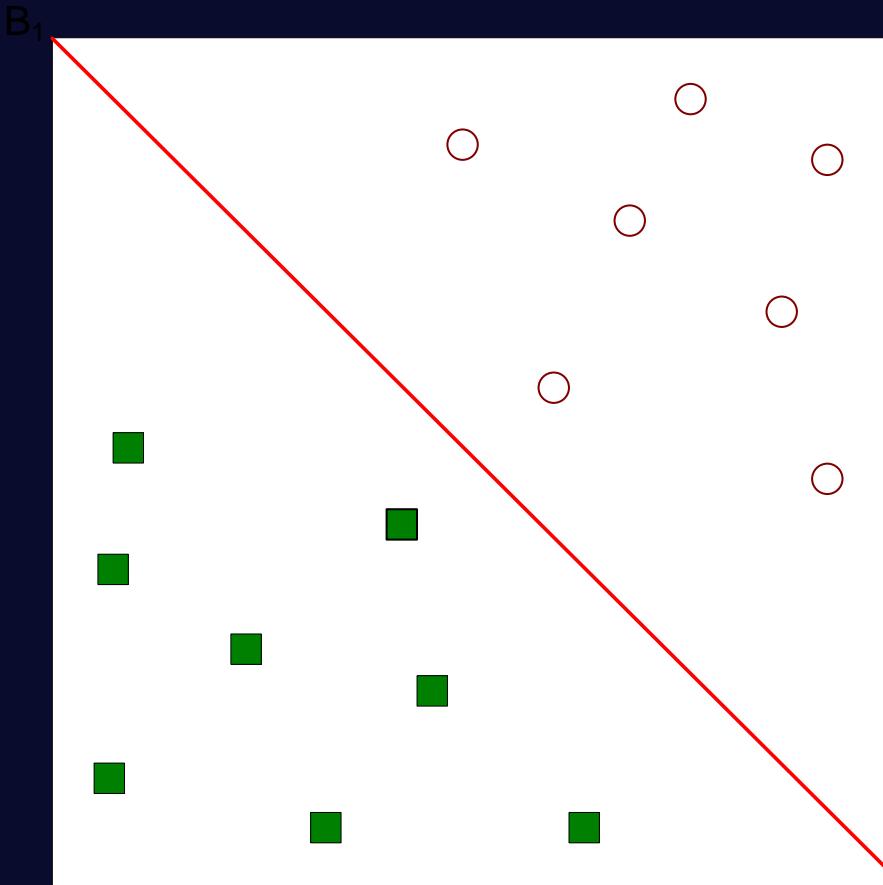
Support Vector Machines

Support Vector Machines



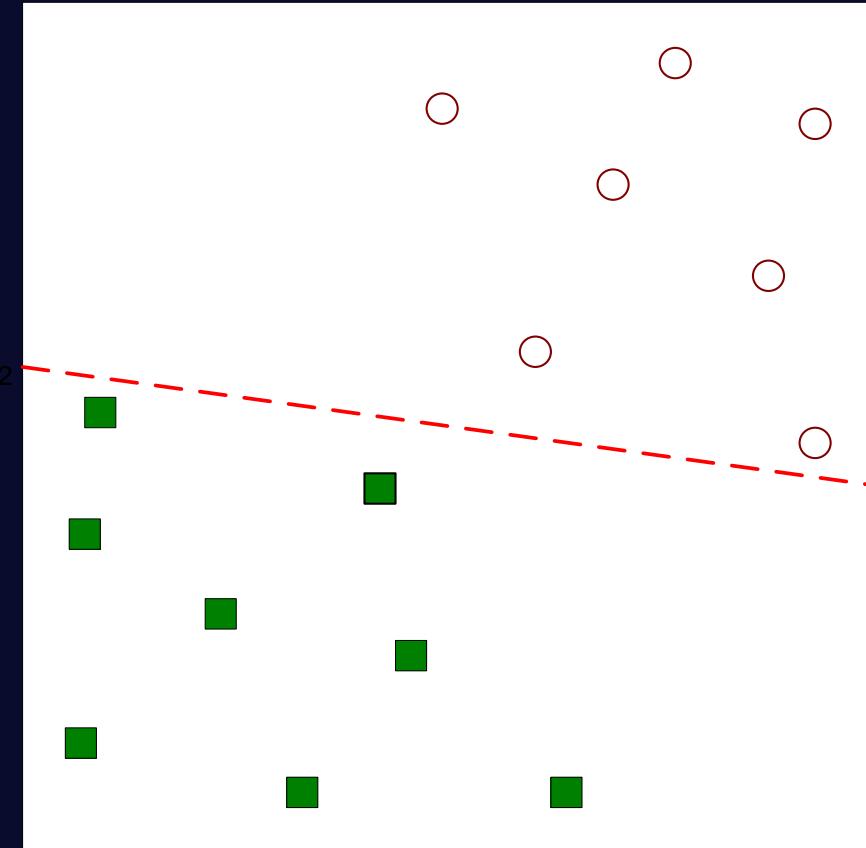
- Find a linear hyperplane (decision boundary) that will separate the data

Support Vector Machines



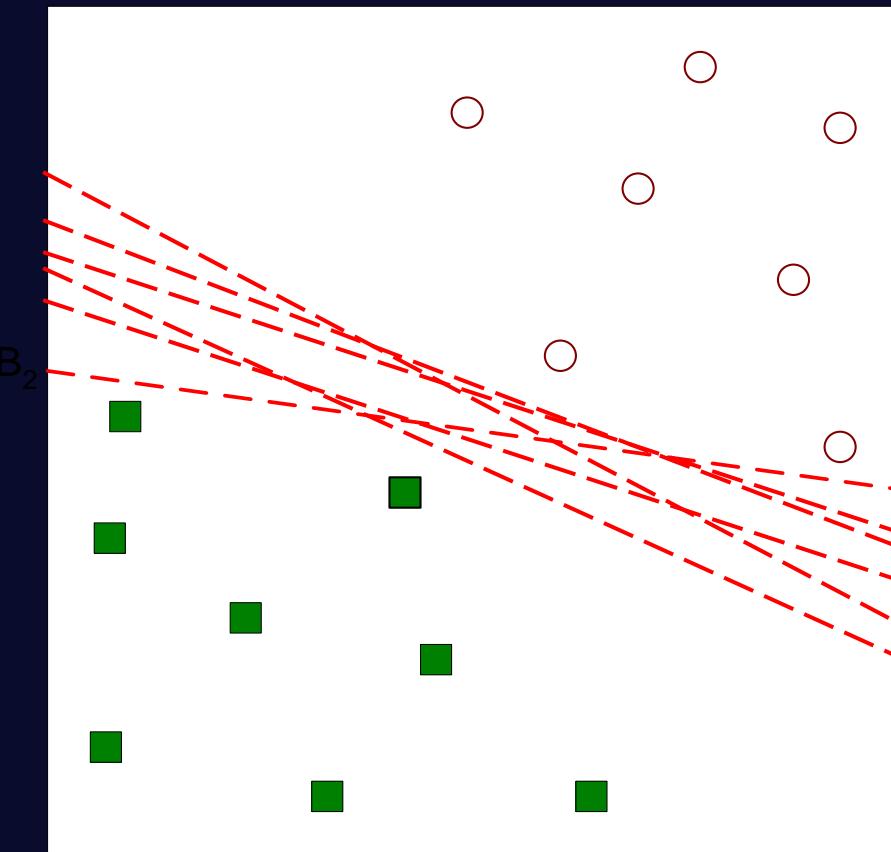
- One Possible Solution

Support Vector Machines



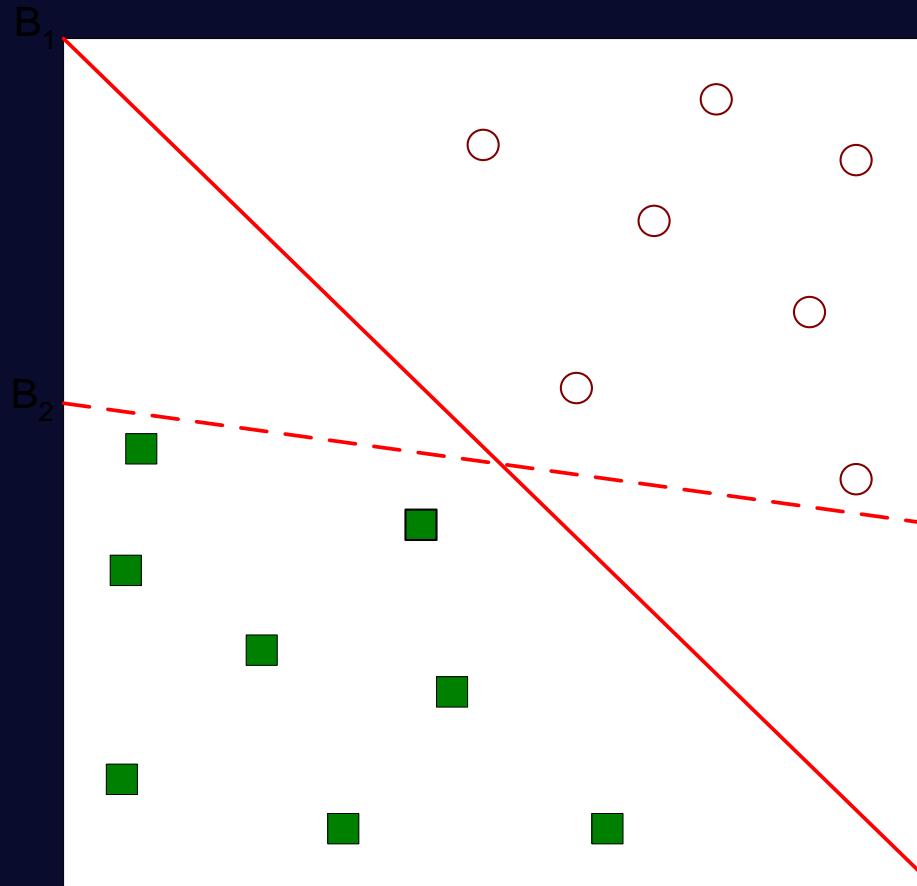
- Another possible solution

Support Vector Machines



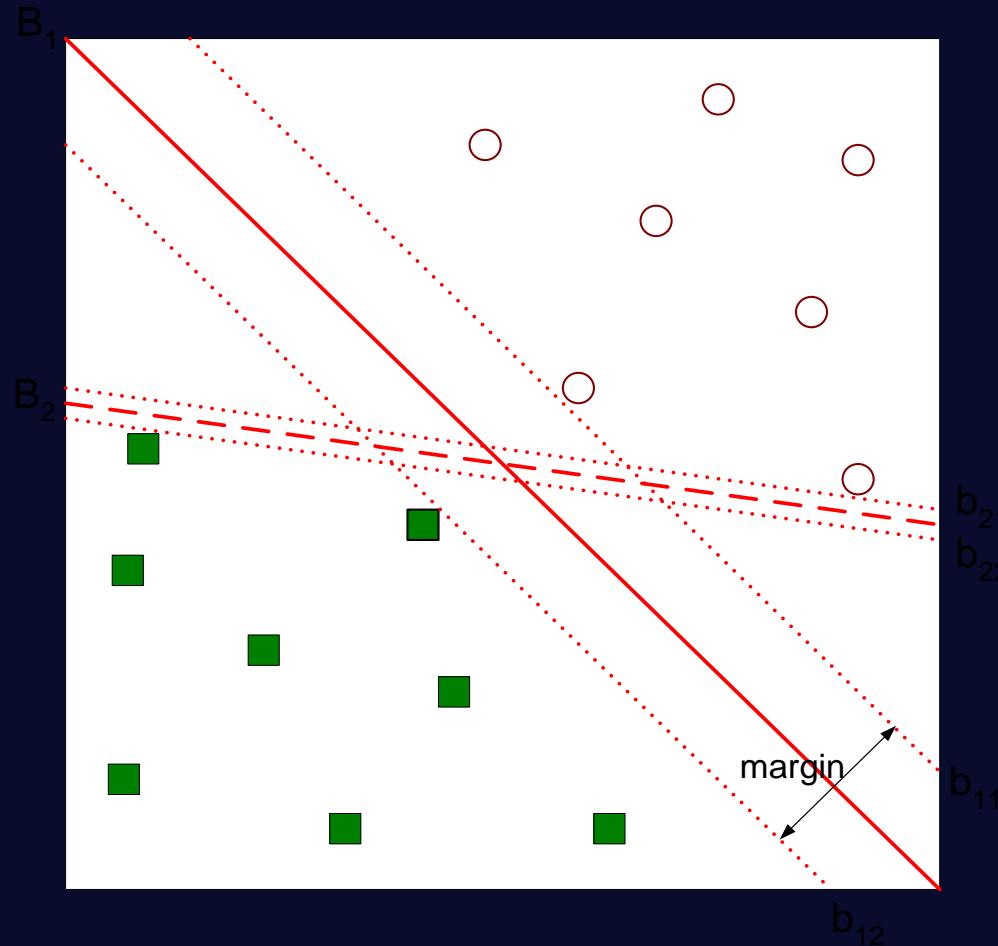
- Other possible solutions

Support Vector Machines



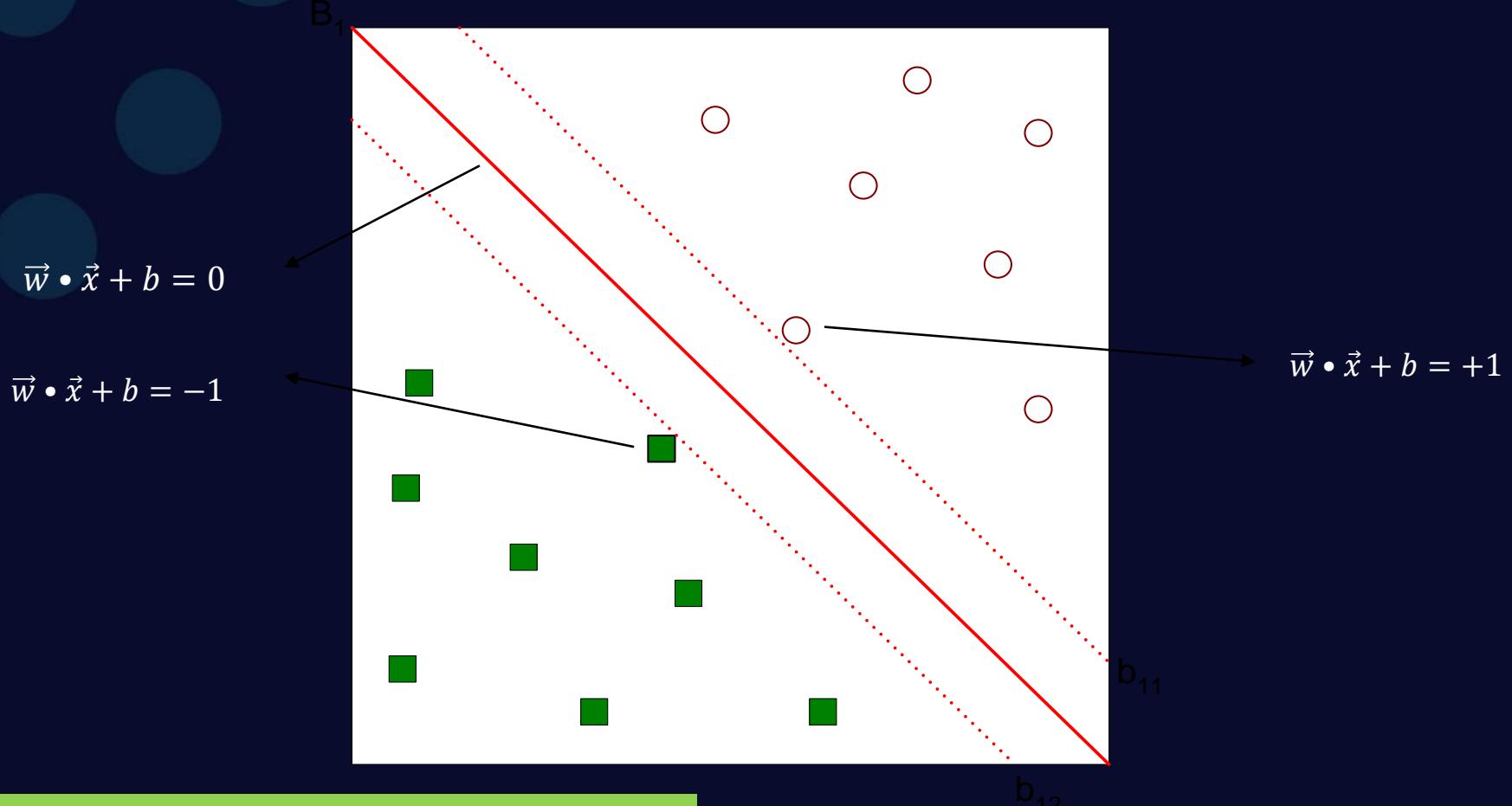
- Which one is better? B_1 or B_2 ?
- How do you define better?

Support Vector Machines



- Find hyperplane **maximizes** the margin => B1 is better than B2

Support Vector Machines

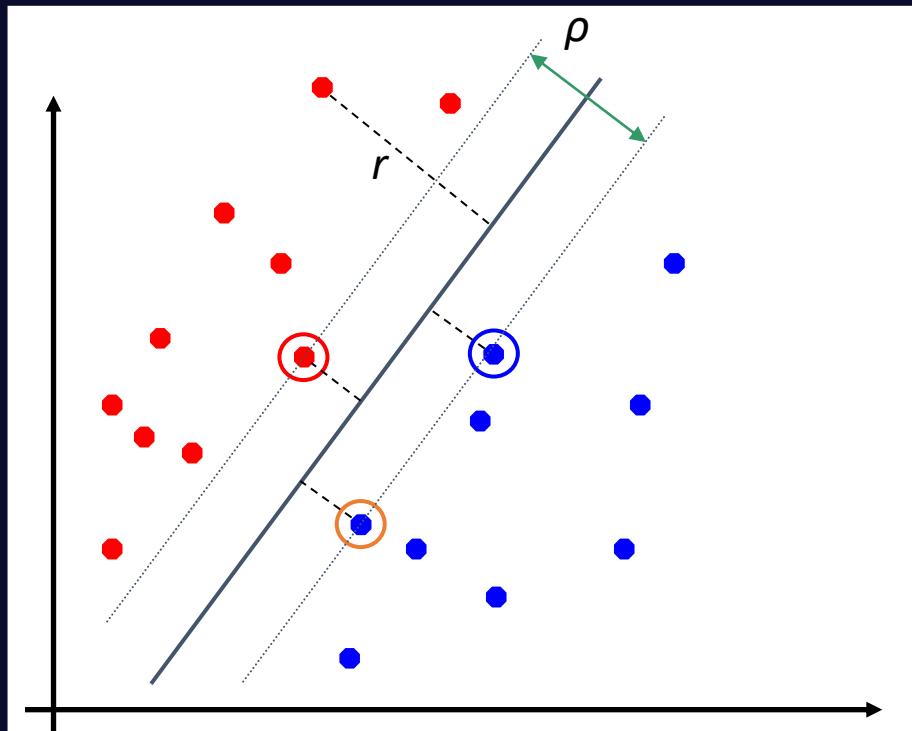


$$f(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} + b \geq 1 \\ -1 & \text{if } \vec{w} \cdot \vec{x} + b \leq -1 \end{cases}$$

$$\text{Margin} = \frac{2}{\|\vec{w}\|}$$

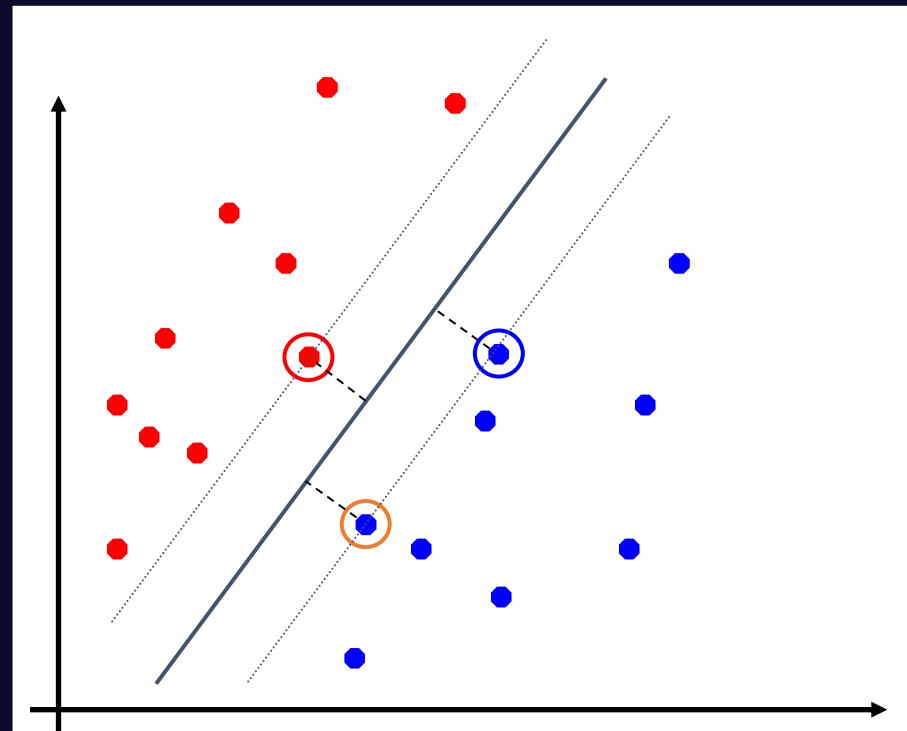
Classification Margin

- Distance from example \mathbf{x}_i to the separator is $r = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are called ***support vectors***.
- ***Margin* ρ** of the separator is the distance between support vectors.



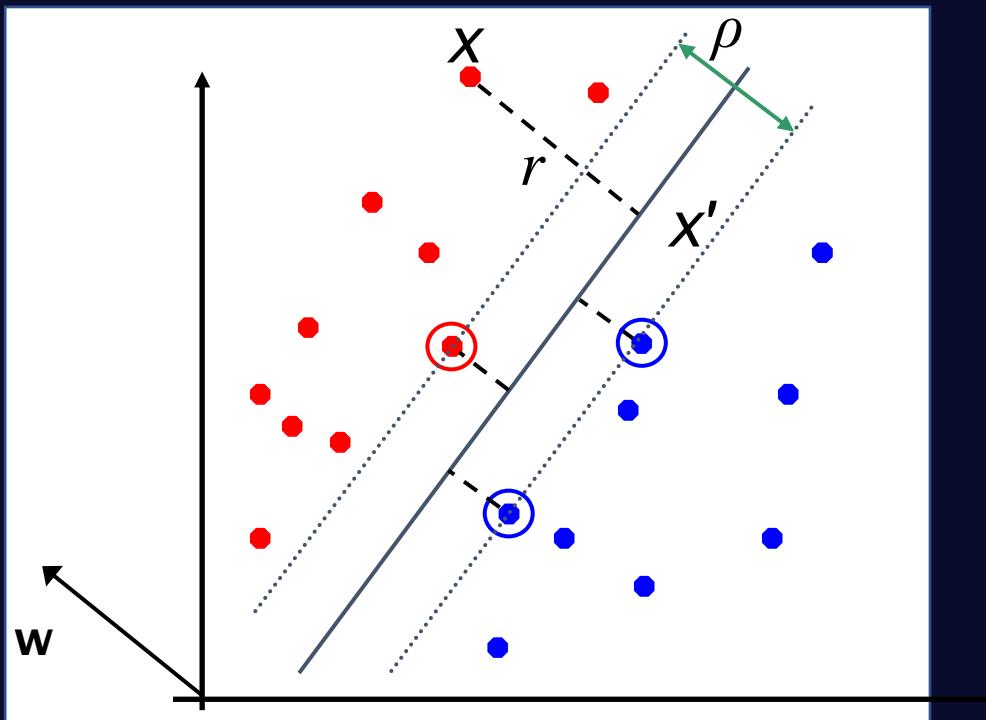
Maximum Margin Classification

- Maximizing the margin is good according to intuition.
- Implies that only support vectors matter; other training examples are ignorable.



Geometric Margin

- Distance from example to the separator is $r = y \frac{\mathbf{w}^T \mathbf{x} + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are **support vectors**.
- **Margin ρ** of the separator is the width of separation between support vectors of classes.



Derivation of finding r :

Dotted line $\mathbf{x}' - \mathbf{x}$ is perpendicular to decision boundary so parallel to \mathbf{w} . Unit vector is $\mathbf{w}/|\mathbf{w}|$, so line is $r\mathbf{w}/|\mathbf{w}|$.

$$\mathbf{x}' = \mathbf{x} - yr\mathbf{w}/|\mathbf{w}|.$$

$$\mathbf{x}' \text{ satisfies } \mathbf{w}^T \mathbf{x}' + b = 0.$$

$$\text{So } \mathbf{w}^T(\mathbf{x} - yr\mathbf{w}/|\mathbf{w}|) + b = 0$$

$$\text{Recall that } |\mathbf{w}| = \sqrt{\mathbf{w}^T \mathbf{w}}.$$

$$\text{So } \mathbf{w}^T \mathbf{x} - yr|\mathbf{w}| + b = 0$$

So, solving for r gives:

$$r = y(\mathbf{w}^T \mathbf{x} + b)/|\mathbf{w}|$$

Linear SVM Mathematically

- Let training set $\{(\mathbf{x}_i, y_i)\}_{i=1..n}$, $\mathbf{x}_i \in \mathbb{R}^d$, $y_i \in \{-1, 1\}$ be separated by a hyperplane with margin ρ . Then for each training example (\mathbf{x}_i, y_i) :

$$\begin{array}{ll} \mathbf{w}^T \mathbf{x}_i + b \leq -\rho/2 & \text{if } y_i = -1 \\ \mathbf{w}^T \mathbf{x}_i + b \geq \rho/2 & \text{if } y_i = 1 \end{array} \Leftrightarrow y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq \rho/2$$

- For every support vector \mathbf{x}_s the above inequality is an equality. After rescaling \mathbf{w} and b by $\rho/2$ in the equality, we obtain that distance between each \mathbf{x}_s and the hyperplane is

$$r = \frac{y_s(\mathbf{w}^T \mathbf{x}_s + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

- Then the margin can be expressed through (rescaled) \mathbf{w} and b as:

$$\rho = 2r = \frac{2}{\|\mathbf{w}\|}$$

Linear SVMs Mathematically (cont.)

- Then we can formulate the *quadratic optimization problem*:

Find \mathbf{w} and b such that

$$\rho = \frac{2}{\|\mathbf{w}\|} \text{ is maximized}$$

and for all $(\mathbf{x}_i, y_i), i=1..n : y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

Which can be reformulated as:

Find \mathbf{w} and b such that

$$\Phi(\mathbf{w}) = \|\mathbf{w}\|^2 = \mathbf{w}^T \mathbf{w} \text{ is minimized}$$

and for all $(\mathbf{x}_i, y_i), i=1..n : y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

Solving the Optimization Problem

Find \mathbf{w} and b such that
 $\Phi(\mathbf{w}) = \mathbf{w}^T \mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i), i=1..n : y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

- Need to optimize a *quadratic* function subject to *linear* constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist.

Solving the Optimization Problem

Find \mathbf{w} and b such that
 $\Phi(\mathbf{w}) = \mathbf{w}^T \mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i), i=1..n :$ $y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every inequality constraint in the primal (original) problem:

Find $\alpha_1 \dots \alpha_n$ such that
 $Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and
(1) $\sum \alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

The Optimization Problem Solution

- Given a solution $\alpha_1 \dots \alpha_n$ to the dual problem, solution to the primal is:

$$\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i \quad b = y_k - \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_k \quad \text{for any } \alpha_k > 0$$

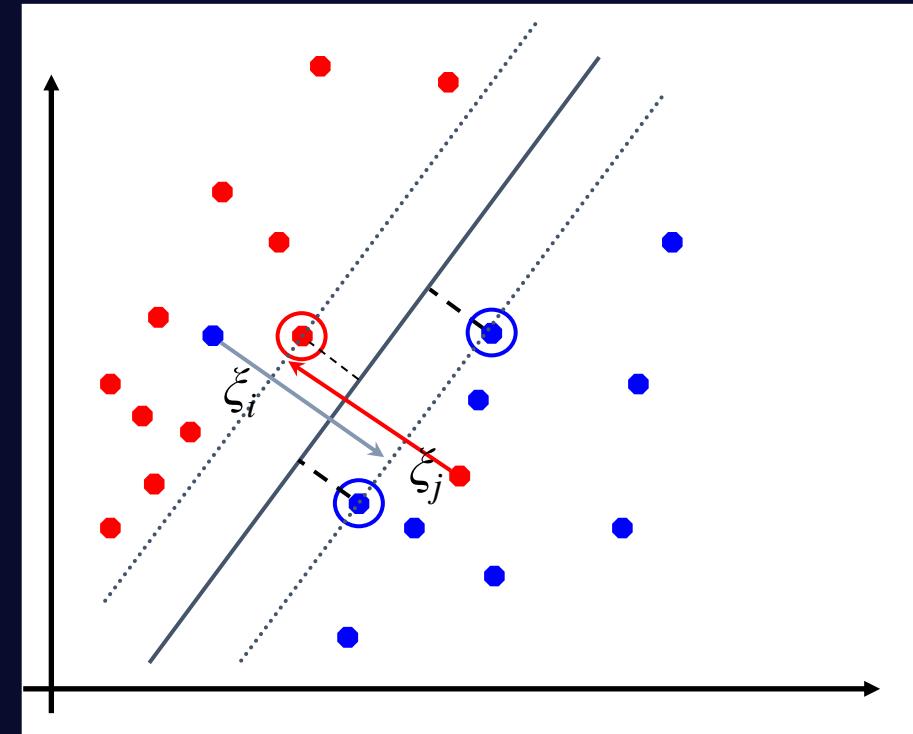
- Each non-zero α_i indicates that corresponding \mathbf{x}_i is a support vector.
- Then the classifying function is (note that we don't need \mathbf{w} explicitly):

$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

- Notice that it relies on an *inner product* between the test point \mathbf{x} and the support vectors \mathbf{x}_i – we will return to this later.
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x}_i^T \mathbf{x}_j$ between all training points.

Soft Margin Classification

- If the training data is not linearly separable, *slack variables* ξ_i can be added to allow misclassification of difficult or noisy examples.
- Allow some errors
 - Let some points be moved to where they belong, at a cost
 - Still, try to minimize training set errors, and to place hyperplane “far” from each class (large margin)



Soft Margin Classification Mathematically

- The old formulation:

Find \mathbf{w} and b such that

$$\begin{aligned}\Phi(\mathbf{w}) = & \frac{1}{2} \mathbf{w}^T \mathbf{w} \text{ is minimized and for all } \{(\mathbf{x}_i, y_i)\} \\ y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq & 1\end{aligned}$$

- The new formulation incorporating slack variables:

Find \mathbf{w} and b such that

$$\begin{aligned}\Phi(\mathbf{w}) = & \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum \xi_i \text{ is minimized and for all } \{(\mathbf{x}_i, y_i)\} \\ y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq & 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0 \text{ for all } i\end{aligned}$$

- Parameter C can be viewed as a way to control overfitting
- A regularization term

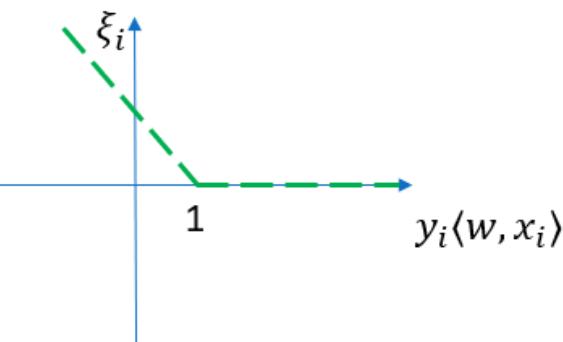
Soft Margin Classification Mathematically

- The new formulation incorporating slack variables:

Find \mathbf{w} and b such that

$$\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum \xi_i \quad \text{is minimized and for all } \{(\mathbf{x}_i, y_i)\}$$
$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0 \text{ for all } i$$

At optimum, must have $\forall i :$



Soft Margin Classification – Solution

- The dual problem for soft margin classification:

Find $\alpha_1 \dots \alpha_N$ such that

$\mathbf{Q}(\mathbf{\alpha}) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and

$$(1) \quad \sum \alpha_i y_i = 0$$

$$(2) \quad 0 \leq \alpha_i \leq C \text{ for all } \alpha_i$$

- Neither slack variables ξ_i nor their Lagrange multipliers appear in the dual problem!
- Again, \mathbf{x}_i with non-zero α_i will be support vectors.
- Solution to the dual problem is:

$$\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$$
$$b = y_k(1 - \xi_k) - \mathbf{w}^T \mathbf{x}_k \text{ where } k = \operatorname{argmax}_{k'} \alpha_{k'}$$

\mathbf{w} is not needed explicitly for classification!

$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

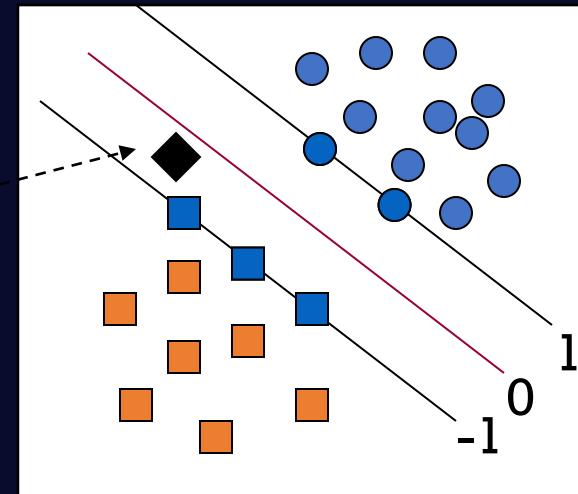
Classification with SVMs

- Given a new point \mathbf{x} , we can score its projection onto the hyperplane normal:
 - i.e., compute score: $\mathbf{w}^T \mathbf{x} + b = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$
 - Decide class based on whether $<$ or > 0
- Can set confidence threshold t .

Score $> t$: yes

Score $< -t$: no

Else: don't know



Linear SVMs: Summary

- The classifier is a *separating hyperplane*.
- The most “important” training points are the support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points \mathbf{x}_i are support vectors with non-zero Lagrangian multipliers α_i ,
- Both in the dual formulation of the problem and in the solution, training points appear only inside inner products:

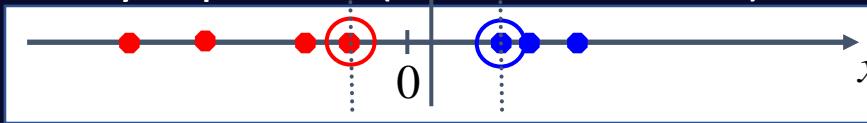
Find $\alpha_1 \dots \alpha_N$ such that
$$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$
 is maximized and

- (1) $\sum \alpha_i y_i = 0$
- (2) $0 \leq \alpha_i \leq C$ for all α_i

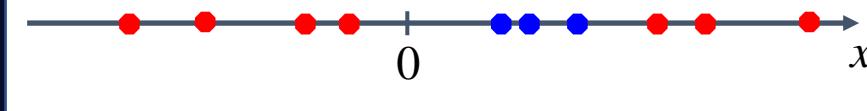
$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

Non-linear SVMs

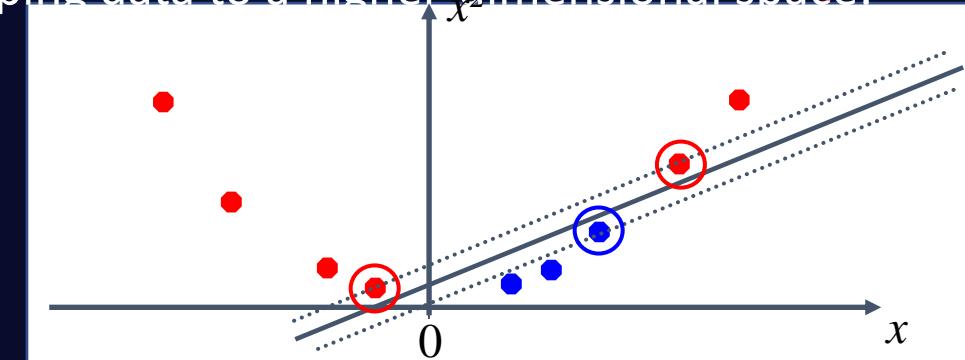
- Datasets that are linearly separable (with some noise) work out great:



- But what are we going to do if the dataset is just too hard?



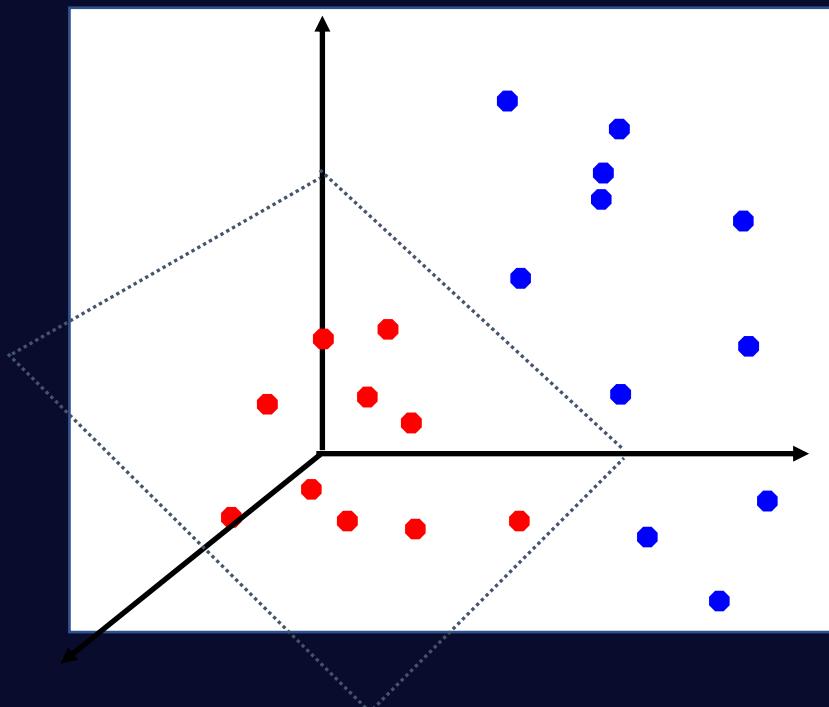
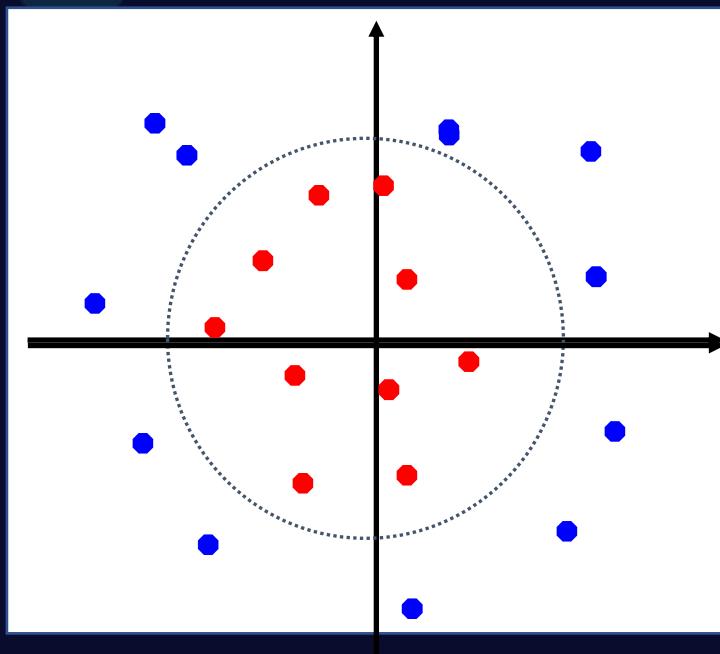
- How about ... mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:

$$\Phi: \mathbf{x} \rightarrow \varphi(\mathbf{x})$$



The “Kernel Trick”

- The linear classifier relies on an inner product between vectors $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- If every datapoint is mapped into high-dimensional space via some transformation $\Phi: \mathbf{x} \rightarrow \phi(\mathbf{x})$, the inner product becomes:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

- A *kernel function* is some function that corresponds to an inner product in some expanded feature space.

The “Kernel Trick”

- Kernel methods replace this dot-product similarity with an arbitrary Kernel function that computes the similarity between x and y

$$K(x, y) : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$$

The “Kernel Trick”

- kernel must be symmetric

$$K(x, y) = K(y, x)$$

- kernel must be positive semi-definite

$$\forall c_i \in \mathbb{R}, x_i \in \mathcal{X}, \sum_{i=1}^n \sum_{j=1}^n c_i c_j K(x_i, x_j) \geq 0$$

The “Kernel Trick”

- kernels are often non-negative

$$K(x, y) \geq 0$$

- kernels are often scaled such that

$$K(x, y) \leq 1, \text{ and } K(x, y) = 1 \Leftrightarrow x = y$$

These properties capture the idea that the kernel is expressing the similarity between x and y

The “Kernel Trick”

- Example:

2-dimensional vectors $\mathbf{x} = [x_1 \ x_2]$; let $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^\top \mathbf{x}_j)^2$,

Need to show that $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^\top \mathbf{x}_j)^2 = 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2} =$$

$$= [1 \ x_{i1}^2 \ \sqrt{2} x_{i1} x_{i2} \ x_{i2}^2 \ \sqrt{2} x_{i1} \ \sqrt{2} x_{i2}]^\top [1 \ x_{j1}^2 \ \sqrt{2} x_{j1} x_{j2} \ x_{j2}^2 \ \sqrt{2} x_{j1} \ \sqrt{2} x_{j2}]$$

$$= \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) \quad \text{where } \phi(\mathbf{x}) = [1, x_1^2, \sqrt{2}, x_1 x_2, x_2^2, \sqrt{2} x_1, \sqrt{2} x_2]$$

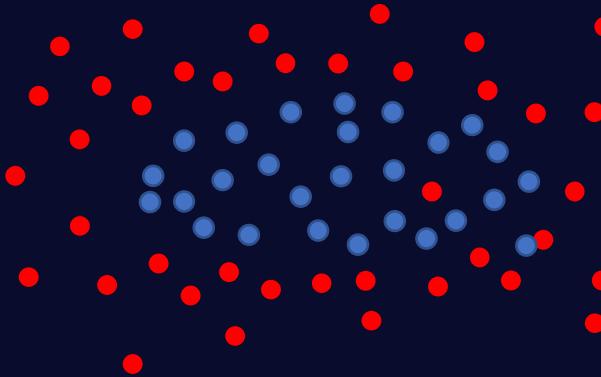
Kernels

- Why use kernels?
 - Make non-separable problem separable.
 - Map data into better representational space
- Common kernels
 - Linear
 - Polynomial $K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^T \mathbf{z})^d$
 - Gives feature conjunctions
 - Radial basis function (infinite dimensional space)

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2}$$

Kernels As Prior Knowledge

- If we think that positive examples can (almost) be separated by some ellipse:



then we should use polynomials of degree 2

- What should we do if we believe that we can classify a text message using words in a dictionary?
- A Kernel encodes a measure of *similarity* between objects.
- Must be a valid inner product function.

Solving SVM's Efficiently

- The objective is convex in....
 - $\alpha_1, \dots, \alpha_m$
- Gradient Descent!
 - ...with a randomization trick
- Instead of going in the direction of the gradient, we go in direction of a random vector
 - $\mathbb{E}[\text{random vector}] = \text{gradient}$

SGD for SVM

Parameter: T (number of steps)

Initialize: $\beta^{(1)} = 0 \in \mathbb{R}^m$

For $t = 1..T$

Let $\alpha^{(t)} = \frac{1}{\lambda t} \beta^{(t)}$

Choose i randomly from $[m]$ // random example

For all $j \neq i$ set $\beta_j^{(t+1)} = \beta_j^{(t)}$

If $y_j \sum_j \alpha_j^{(t)} K(x_j, x_i) < 1$ // x_i 's prediction incorrect/low margin?

Set $\beta_i^{(t+1)} = \beta_i^{(t)} + y_i$

Else

Set $\beta_i^{(t+1)} = \beta_i^{(t)}$

Output $\bar{w} = \sum_j \bar{\alpha}_j \phi(x_j)$ where $\bar{\alpha} = \frac{1}{T} \sum_t \alpha^{(t)}$



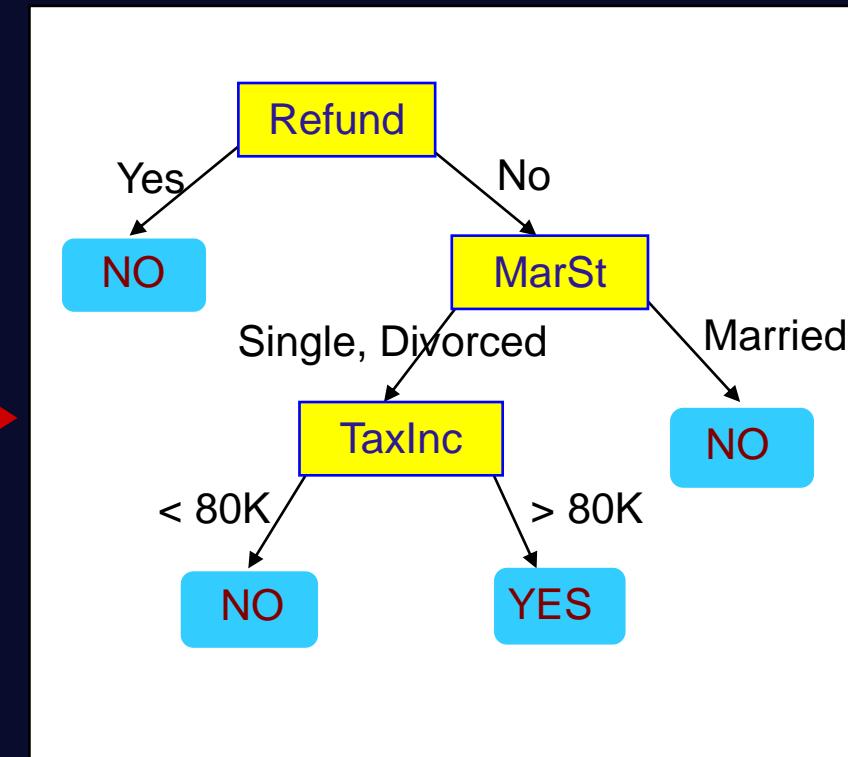
Decision Trees

Example of a Decision Tree

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

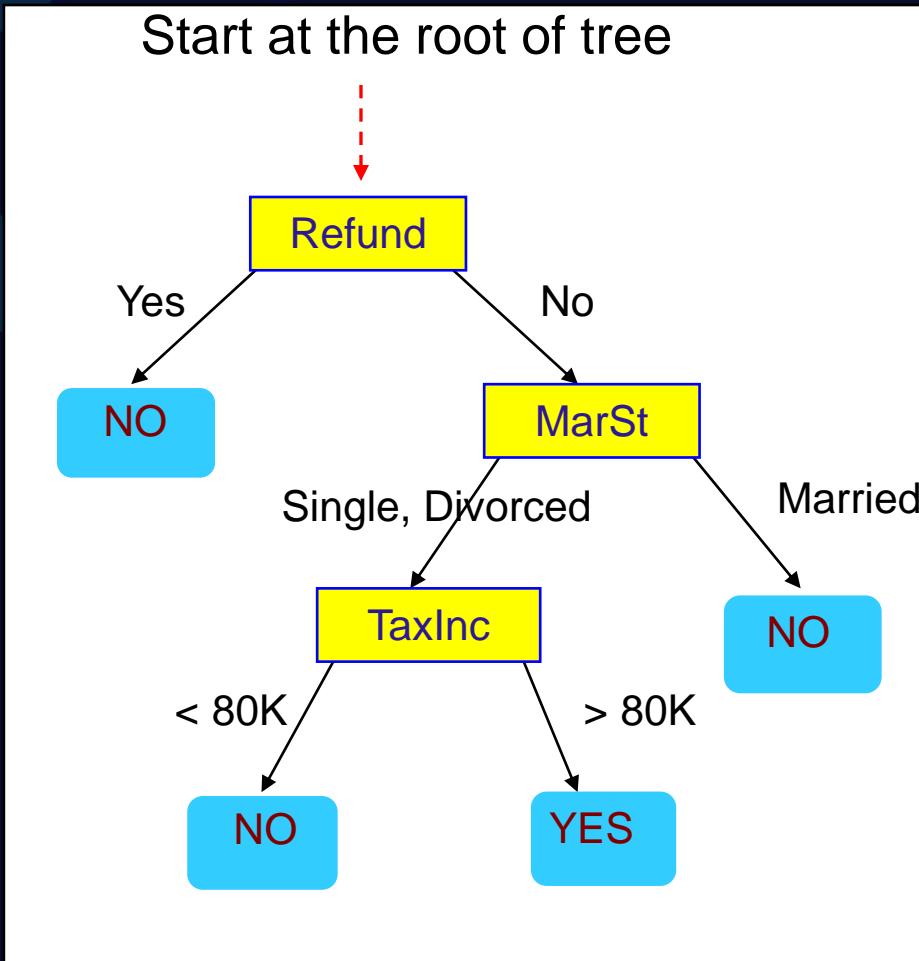


Training Data



Model: Decision Tree

Apply Model to Test Data

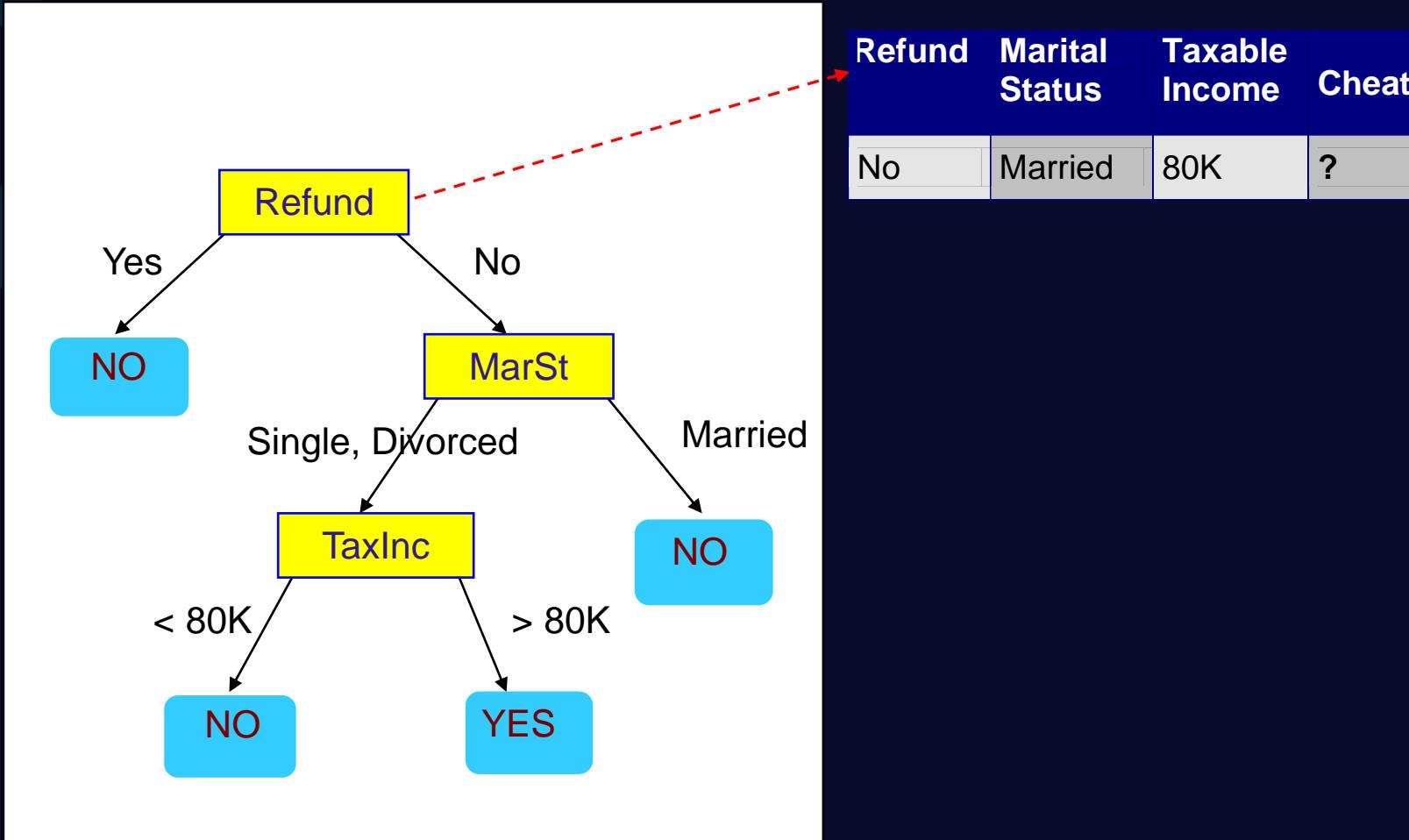


Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply Model to Test Data

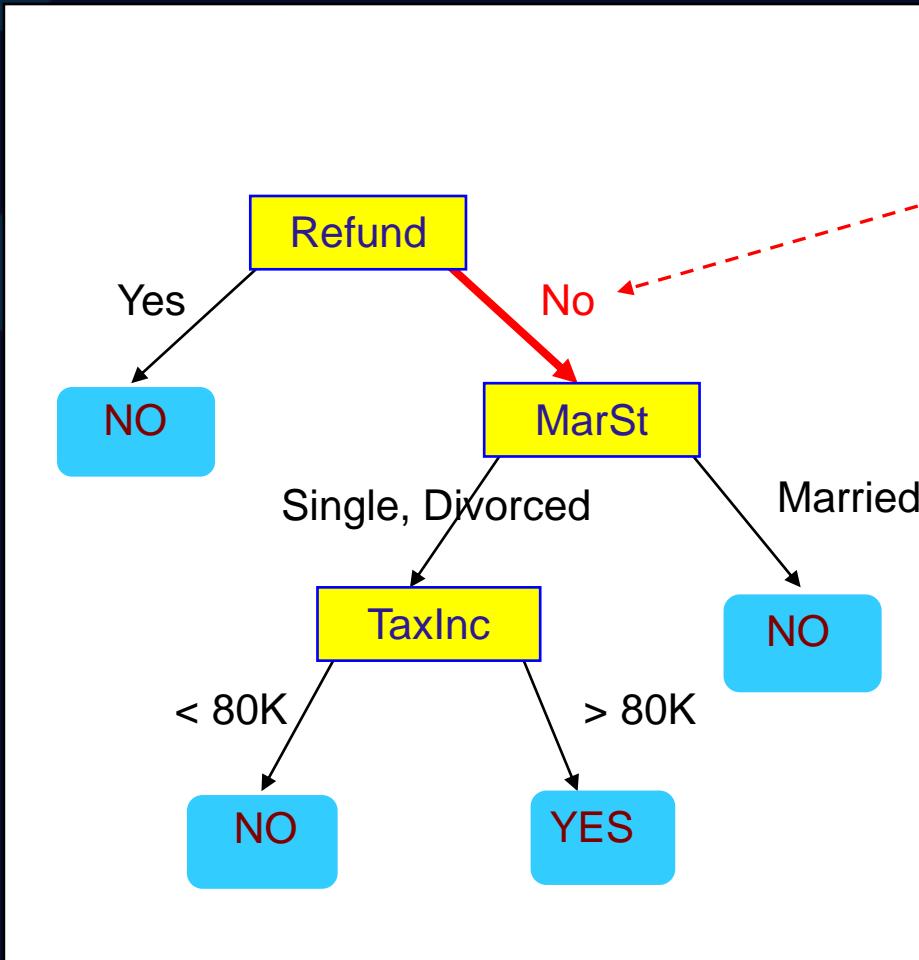
Test Data



Apply Model to Test Data

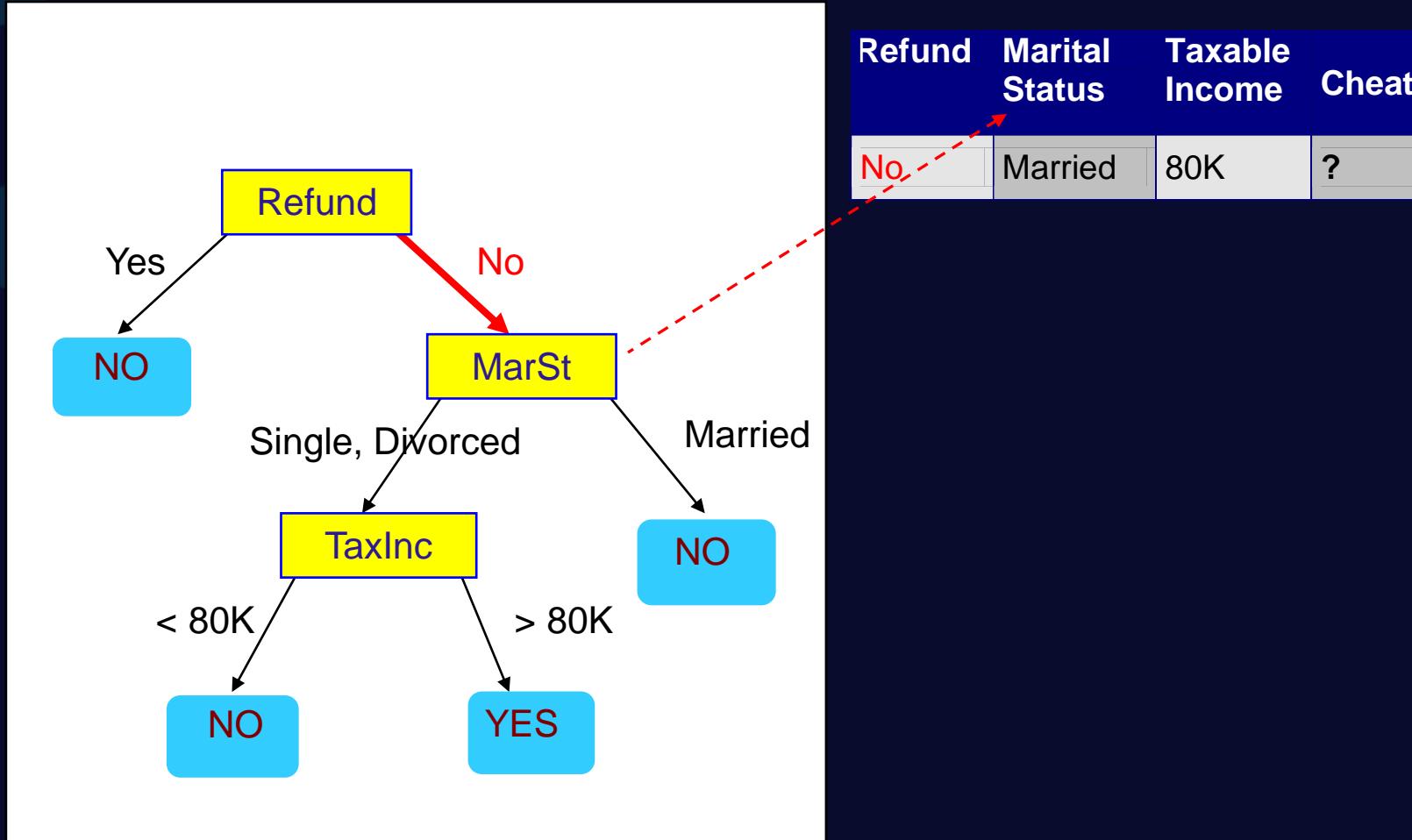
Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



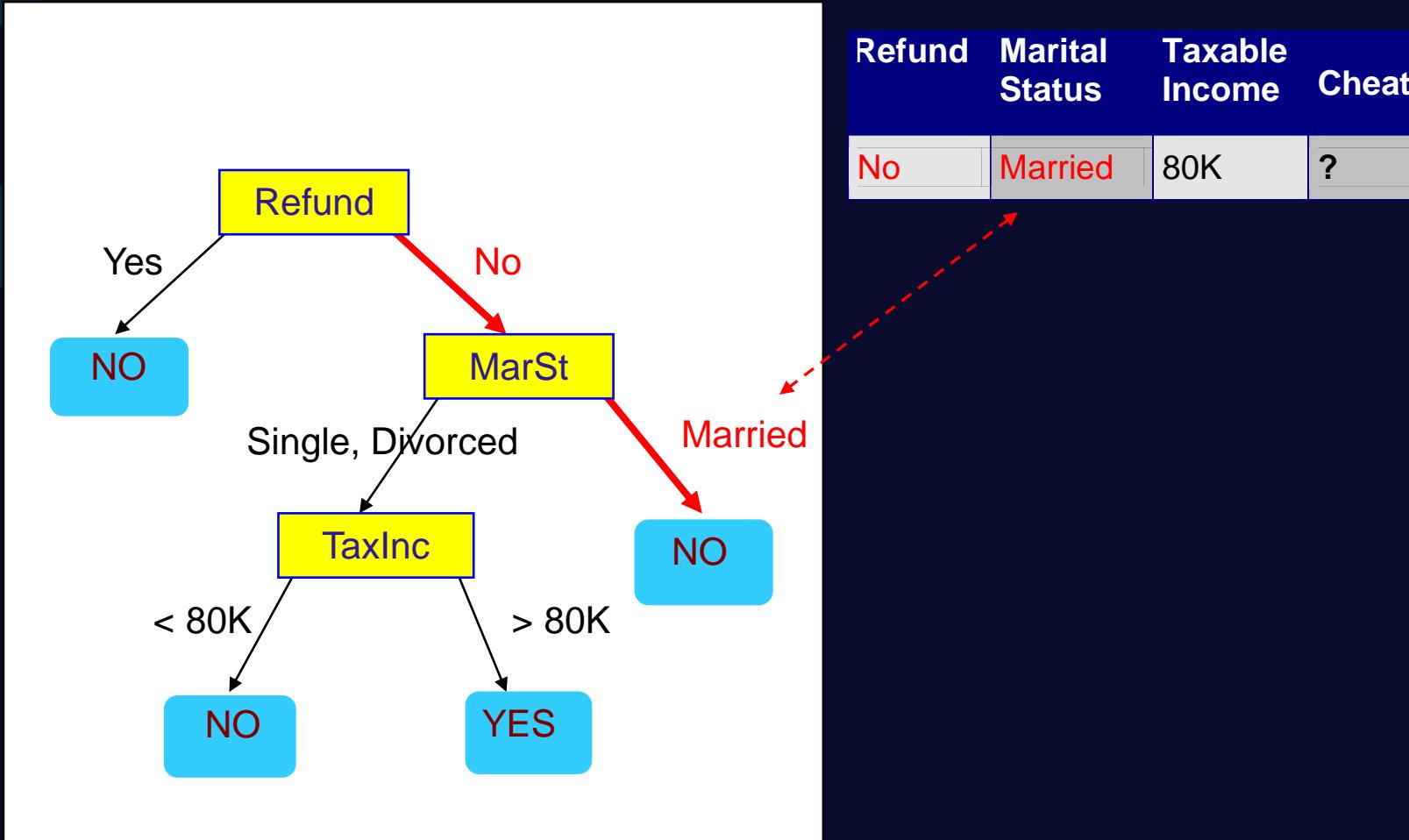
Apply Model to Test Data

Test Data



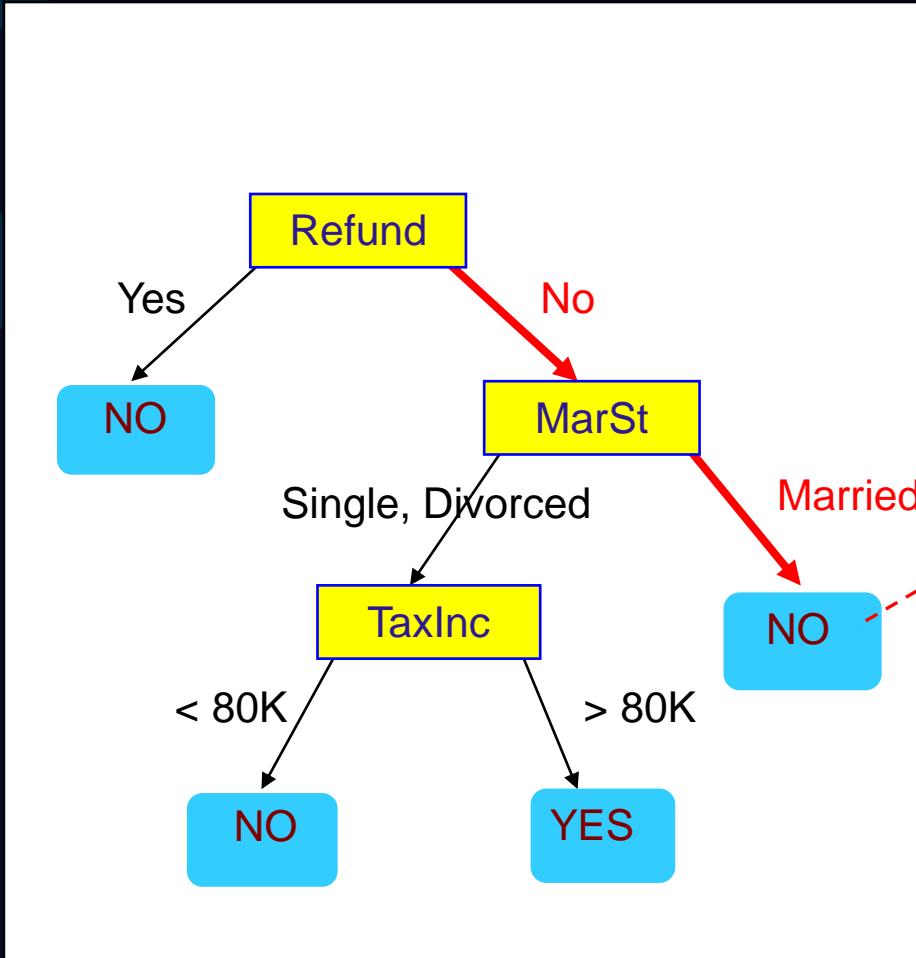
Apply Model to Test Data

Test Data



Apply Model to Test Data

Test Data



Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Assign Cheat to “No”

Decision Tree Learning: ID3

ID3(*Training-set, Attributes*)

- If all elements in *Training-set* are in same class, then return leaf node labeled with that class
- Else if *Attributes* is empty, then return leaf node labeled with majority class in *Training-set*
- Else if *Training-Set* is empty, then return leaf node labeled with default majority class
- Else

Select and remove *A* from *Attributes*

Make *A* the root of the current tree

For each value *V* of *A*

- Create a branch of the current tree labeled by *V*
- $\text{Partition}_V \leftarrow$ Elements of *Training-set* with value *V* for *A*
- Induce-Tree(Partition_V , *Attributes*)
- Attach result to branch *V*

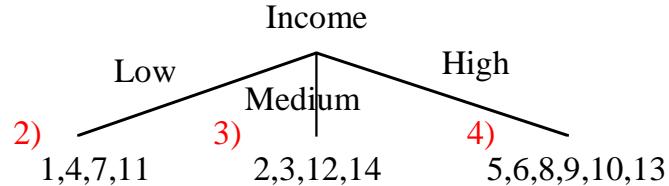
Illustrative Training Set

Risk Assessment for Loan Applications

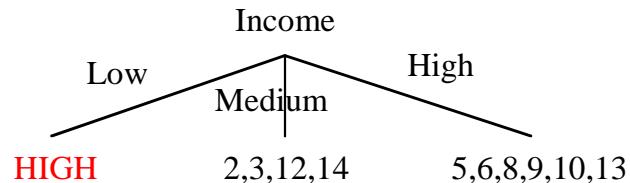
Client #	Credit History	Debt Level	Collateral	Income Level	RISK LEVEL
1	Bad	High	None	Low	HIGH
2	Unknown	High	None	Medium	HIGH
3	Unknown	Low	None	Medium	MODERATE
4	Unknown	Low	None	Low	HIGH
5	Unknown	Low	None	High	LOW
6	Unknown	Low	Adequate	High	LOW
7	Bad	Low	None	Low	HIGH
8	Bad	Low	Adequate	High	MODERATE
9	Good	Low	None	High	LOW
10	Good	High	Adequate	High	LOW
11	Good	High	None	Low	HIGH
12	Good	High	None	Medium	MODERATE
13	Good	High	None	High	LOW
14	Bad	High	None	Medium	HIGH

ID3 Example (I)

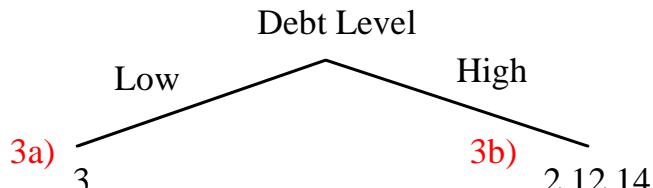
1) Choose Income as root of tree.



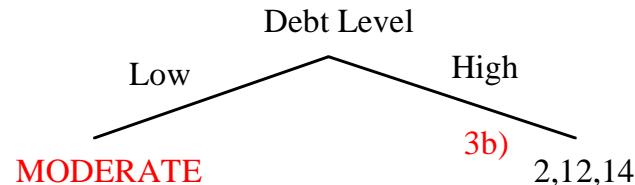
2) All examples are in the same class, HIGH.
Return Leaf Node.



3) Choose Debt Level as root of subtree.

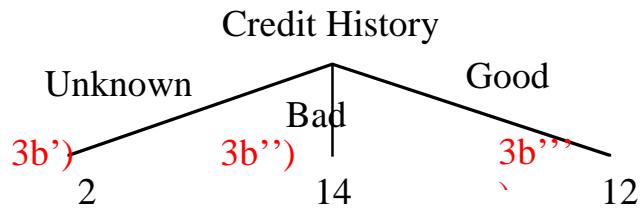


3a) All examples are in the same class, MODERATE.
Return Leaf node.

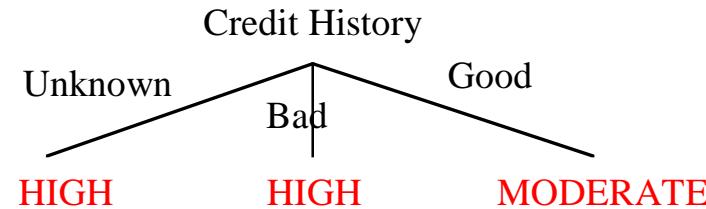


ID3 Example (II)

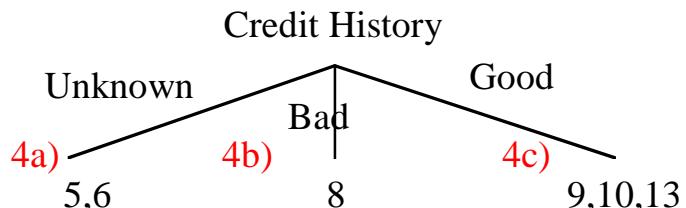
3b) Choose Credit History as root of subtree.



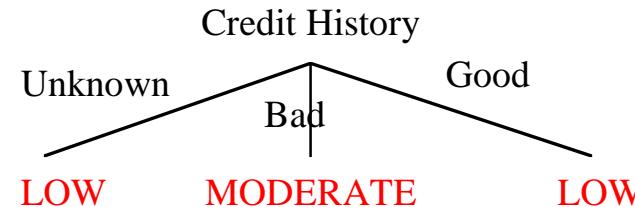
3b'-3b''') All examples are in the same class.
Return Leaf nodes.



4) Choose Credit History as root of subtree.

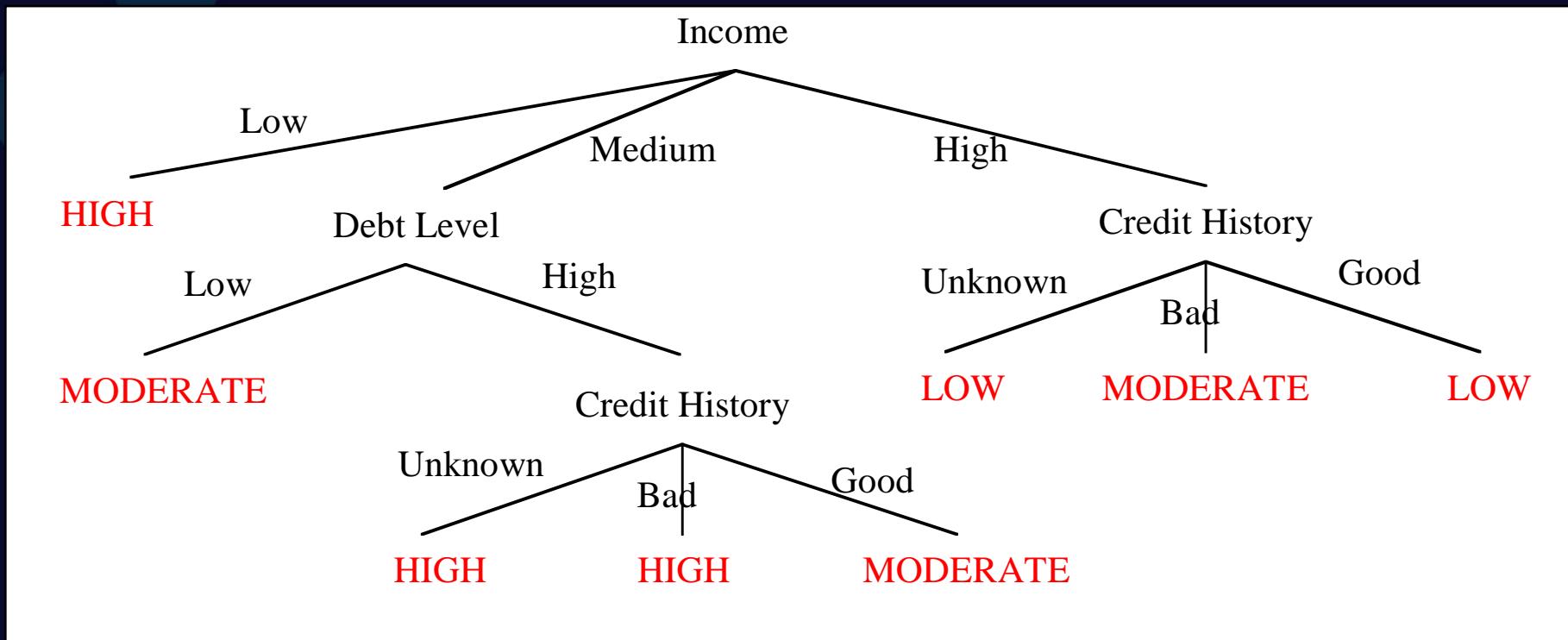


4a-4c) All examples are in the same class.
Return Leaf nodes.



ID3 Example (III)

Attach subtrees at appropriate places.

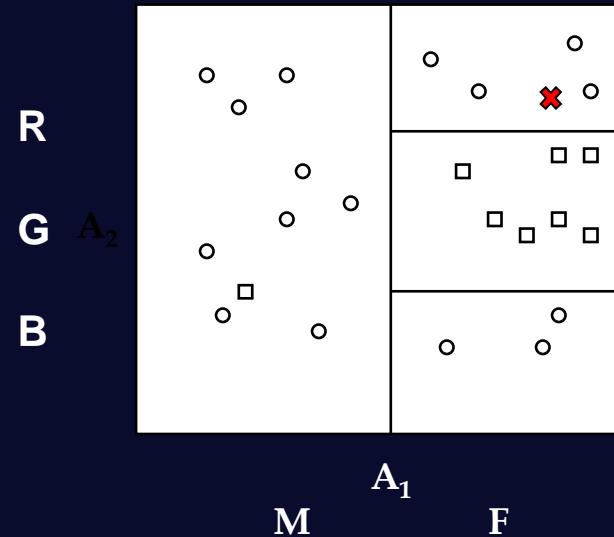


Another Example

Assume A_1 is binary feature (Gender: M/F)

Assume A_2 is nominal feature (Color: R/G/B)

Decision surfaces are axis-aligned Hyper-rectangles

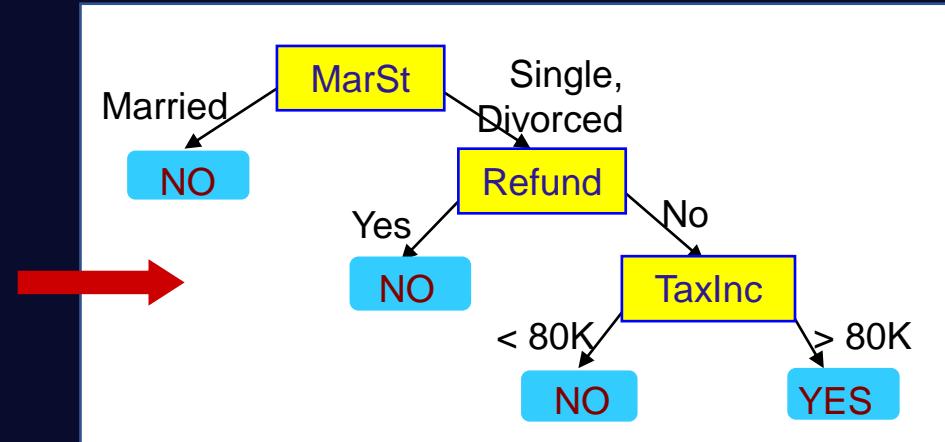


Non-Uniqueness

Decision trees are not unique:

- Given a set of training instances T , there generally exists a number of decision trees that are consistent with (or fit) T

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



ID3's Question

Given a training set, which of all of the decision trees consistent with that training set should we pick?

More precisely:

Given a training set, which of all of the decision trees consistent with that training set has the **greatest likelihood of correctly classifying unseen instances of the population?**

ID3's (Approximate) Bias

- ID3 (and family) prefers simpler decision trees
- Occam's Razor Principle:
 - “It is vain to do with more what can be done with less...Entities should not be multiplied beyond necessity.”
- Intuitively:
 - Always accept the simplest answer that fits the data, avoid unnecessary constraints
 - Simpler trees are more general

ID3's Question Revisited

Since ID3 builds a decision tree by recursively selecting attributes and splitting the training data based on the values of these attributes

Practically:

Given a training set, how do we select attributes so that the resulting tree is as small as possible, i.e. contains as few attributes as possible?

Not All Attributes Are Created Equal

Each attribute of an instance may be thought of as contributing a certain amount of information to its classification

- Think 20-Questions:

- What are good questions?

- Ones whose answers maximize information gained

- For example, determine shape of an object:

- Num. sides contributes a certain amount of information

- Color contributes a different amount of information

ID3 measures information gained by making each attribute the root of the current subtree, and subsequently chooses the attribute that produces the greatest information gain

Entropy (as information)

Entropy at a given node t:

$$Entropy(t) = -\sum_j p(j | t) \log p(j | t)$$

(where $p(j | t)$ is the relative frequency of class j at node t)

Entropy (as homogeneity)

Think chemistry/physics

- Entropy is measure of disorder or homogeneity
- Minimum (0.0) when homogeneous / perfect order
- Maximum (1.0, in general $\log C$) when most heterogeneous / complete chaos

In ID3

- Minimum (0.0) when all records belong to one class, implying most information
- Maximum ($\log C$) when records are equally distributed among all classes, implying least information
- Intuitively, the smaller the entropy the purer the partition

Examples of Computing Entropy

$$Entropy(t) = -\sum_j p(j | t) \log_2 p(j | t)$$

C1	0
C2	6

$$P(C1) = 0/6 = 0 \quad P(C2) = 6/6 = 1$$

$$\text{Entropy} = -0 \log 0 - 1 \log 1 = -0 - 0 = 0$$

C1	1
C2	5

$$P(C1) = 1/6 \quad P(C2) = 5/6$$

$$\text{Entropy} = -(1/6) \log_2 (1/6) - (5/6) \log_2 (1/6) = 0.65$$

C1	2
C2	4

$$P(C1) = 2/6 \quad P(C2) = 4/6$$

$$\text{Entropy} = -(2/6) \log_2 (2/6) - (4/6) \log_2 (4/6) = 0.92$$

Information Gain

Information Gain:

$$GAIN_{split} = Entropy(p) - \left(\sum_{i=1}^k \frac{n_i}{n} Entropy(i) \right)$$

(where parent Node, p is split into k partitions, and
n_i is number of records in partition i)

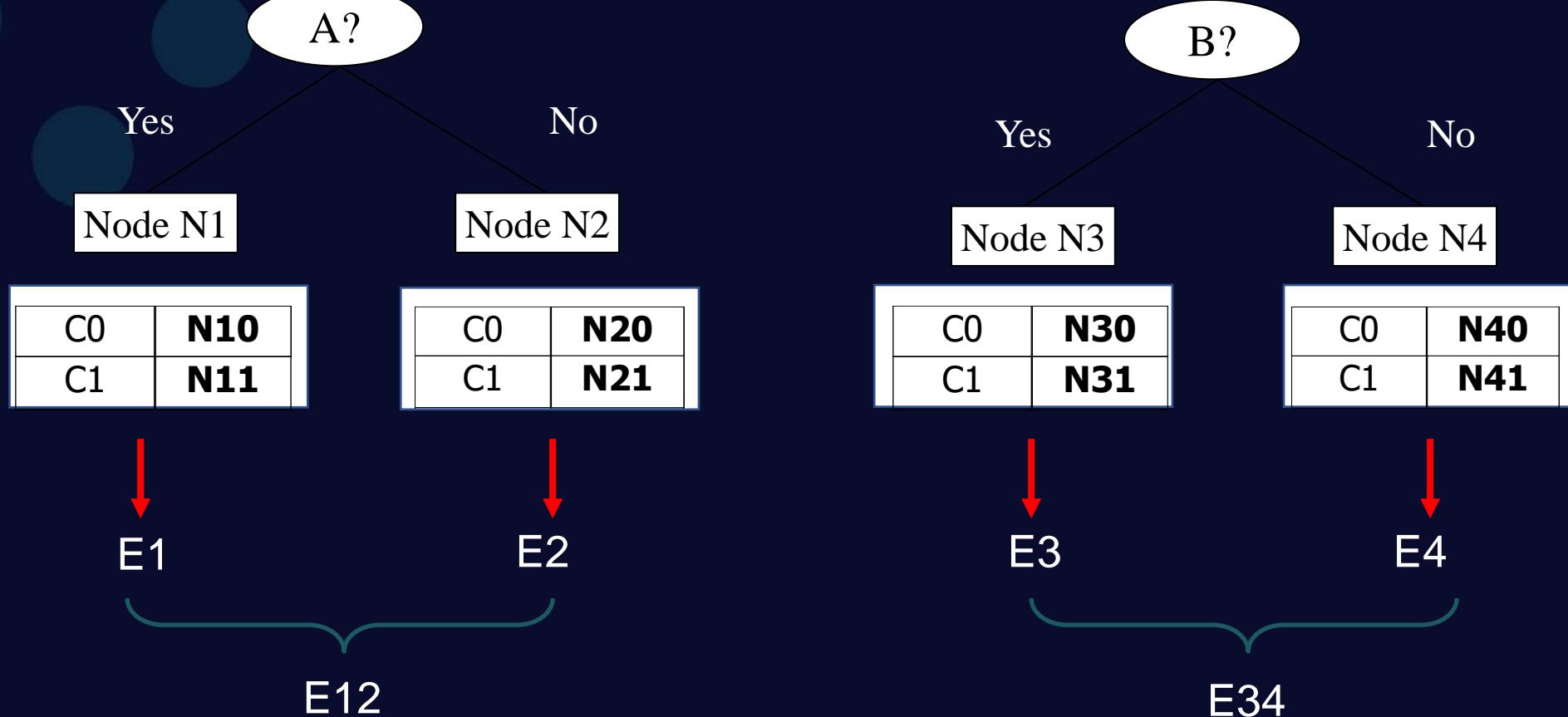
- Measures reduction in entropy achieved because of the split → maximize
- ID3 chooses to split on the attribute that results in the largest reduction, i.e, (maximizes GAIN)
- Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure.

Computing Gain

Before Splitting:

C0	N00
C1	N01

E0



$$\text{Gain} = E_0 - E_{12} \text{ vs. } E_0 - E_{34}$$

Gain Ratio

Gain Ratio:

$$GainRATIO_{split} = \frac{GAIN_{Split}}{SplitINFO}$$

$$SplitINFO = -\sum_{i=1}^k \frac{n_i}{n} \log \frac{n_i}{n}$$

(where parent Node, p is split into k partitions, and
n_i is number of records in partition i)

- Designed to overcome the disadvantage of GAIN
- Adjusts GAIN by the entropy of the partitioning (SplitINFO)
- Higher entropy partitioning (large number of small partitions) is penalized
- Used by C4.5 (an extension of ID3)

Other Splitting Criterion: GINI Index

GINI Index for a given node t :

$$GINI(t) = 1 - \sum_j [p(j|t)]^2$$

$$GINI_{split} = \sum_{i=1}^k \frac{n_i}{n} GINI(i)$$

- Maximum ($1 - 1/n_c$) when records are equally distributed among all classes, implying least interesting information
- Minimum (0.0) when all records belong to one class, implying most interesting information
- Minimize GINI split value
- Used by CART, SLIQ, SPRINT

How to Specify Test Condition?

Depends on attribute types

- Nominal
- Ordinal
- Continuous

Depends on number of ways to split

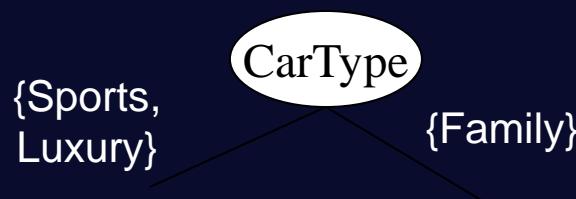
- Binary split
- Multi-way split

Splitting Based on Nominal Attributes

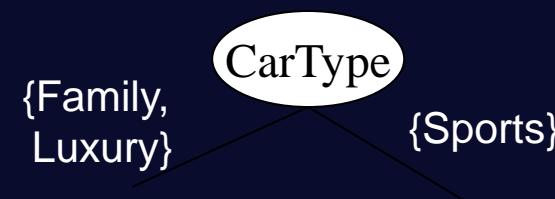
Multi-way split: Use as many partitions as values



Binary split: Divide values into two subsets



OR



Need to find optimal partitioning!

Splitting Based on Continuous Attributes

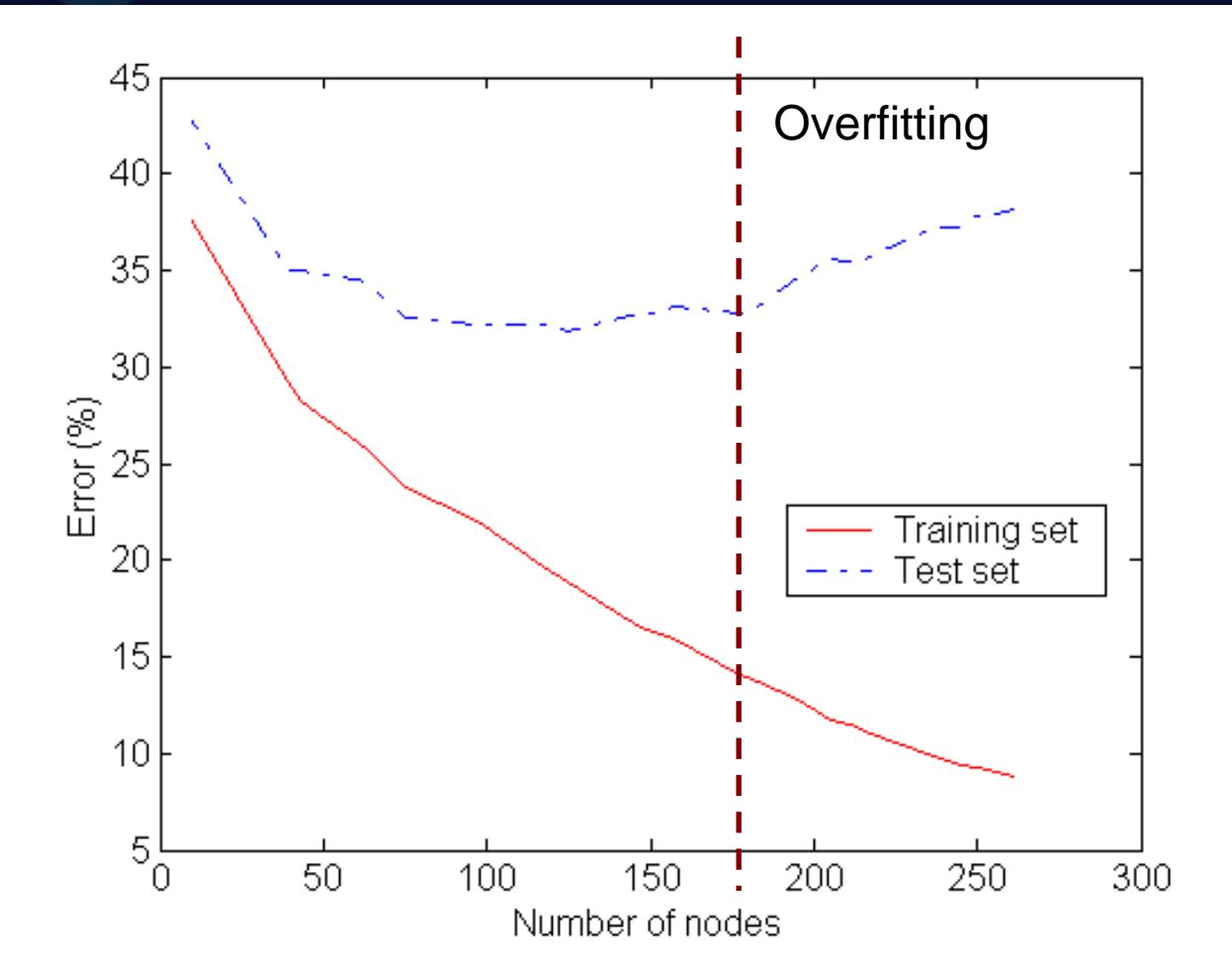
- Different ways of handling
 - **Multi-way split:** form ordinal categorical attribute
 - Static – discretize once at the beginning
 - Dynamic – repeat on each new partition
 - **Binary split:** $(A < v)$ or $(A \geq v)$
 - How to choose v ?

Need to find optimal partitioning!



Can use GAIN or GINI !

Detecting Overfitting



Overfitting in Decision Tree Learning

Overfitting results in decision trees that are more complex than necessary

- Tree growth went too far
- Number of instances gets smaller as we build the tree (e.g., several leaves match a single example)

Training error no longer provides a good estimate of how well the tree will perform on previously unseen records

Avoiding Tree Overfitting – Solution 1

Pre-Pruning (Early Stopping Rule)

- Stop the algorithm before it becomes a fully-grown tree
- Typical stopping conditions for a node:
 - Stop if all instances belong to the same class
 - Stop if all the attribute values are the same
- More restrictive conditions:
 - Stop if number of instances is less than some user-specified threshold
 - Stop if class distribution of instances are independent of the available features (e.g., using χ^2 test)
 - Stop if expanding the current node does not improve impurity measures (e.g., GINI or GAIN)

Avoiding Tree Overfitting – Solution 2

Post-pruning

Split dataset into training and validation sets

Grow full decision tree on training set

While the accuracy on the validation set increases:

- Evaluate the impact of pruning each subtree, replacing its root by a leaf labeled with the majority class for that subtree
- Replace subtree that most increases validation set accuracy (greedy approach)

Decision Tree Based Classification

Advantages:

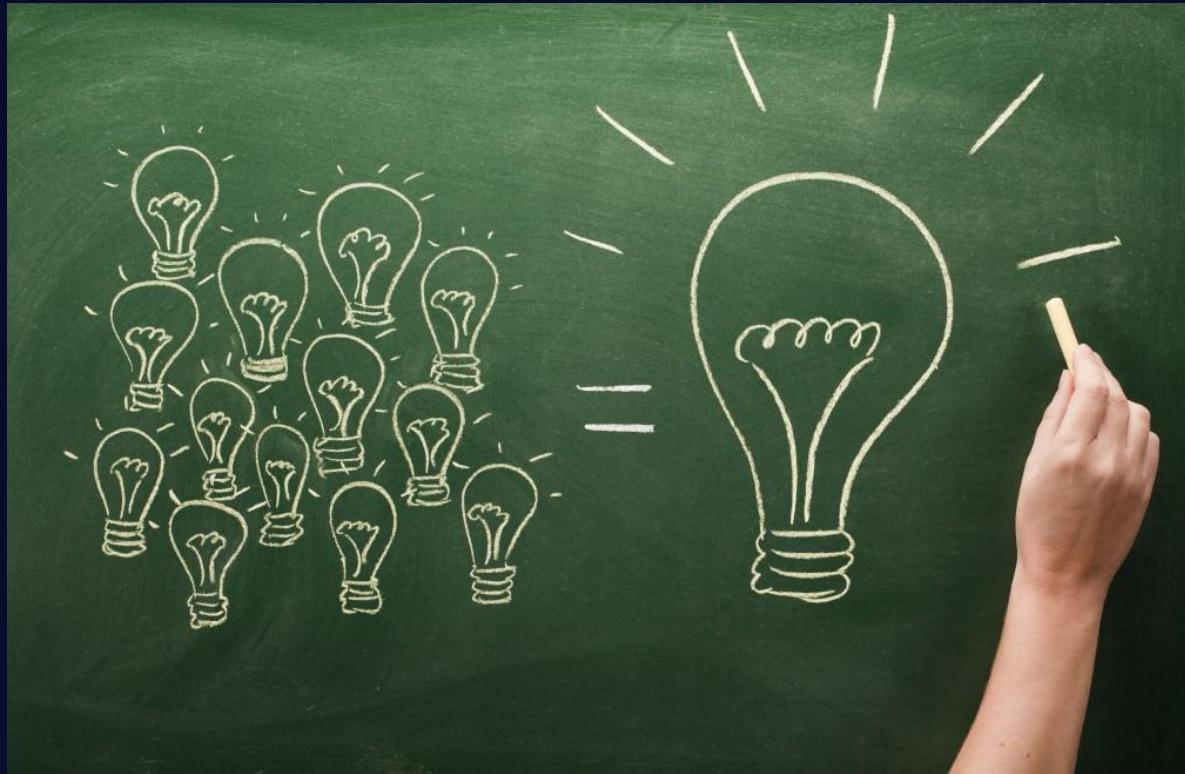
- Inexpensive to construct
- Extremely fast at classifying unknown records
- Easy to interpret for small-sized trees

Disadvantages:

- Axis-parallel decision boundaries
- Redundancy
- Need data to fit in memory
- Need to retrain with new data

Power of the crowds

- Wisdom of the crowds



Ensemble methods

- A single decision tree does not perform well
- But, it is super fast
- What if we learn multiple trees?

We need to make sure they do not all just learn the same

Bagging

If we split the data in random different ways, decision trees give different results, **high variance**.

Bagging: Bootstrap aggregating is a method that result in low variance.

If we had multiple realizations of the data (or multiple samples) we could calculate the predictions multiple times and take the average of the fact that averaging multiple onerous estimations produce less uncertain results

Bagging

Say for each sample b , we calculate $f^b(x)$, then:

$$\hat{f}_{avg}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x)$$

How?

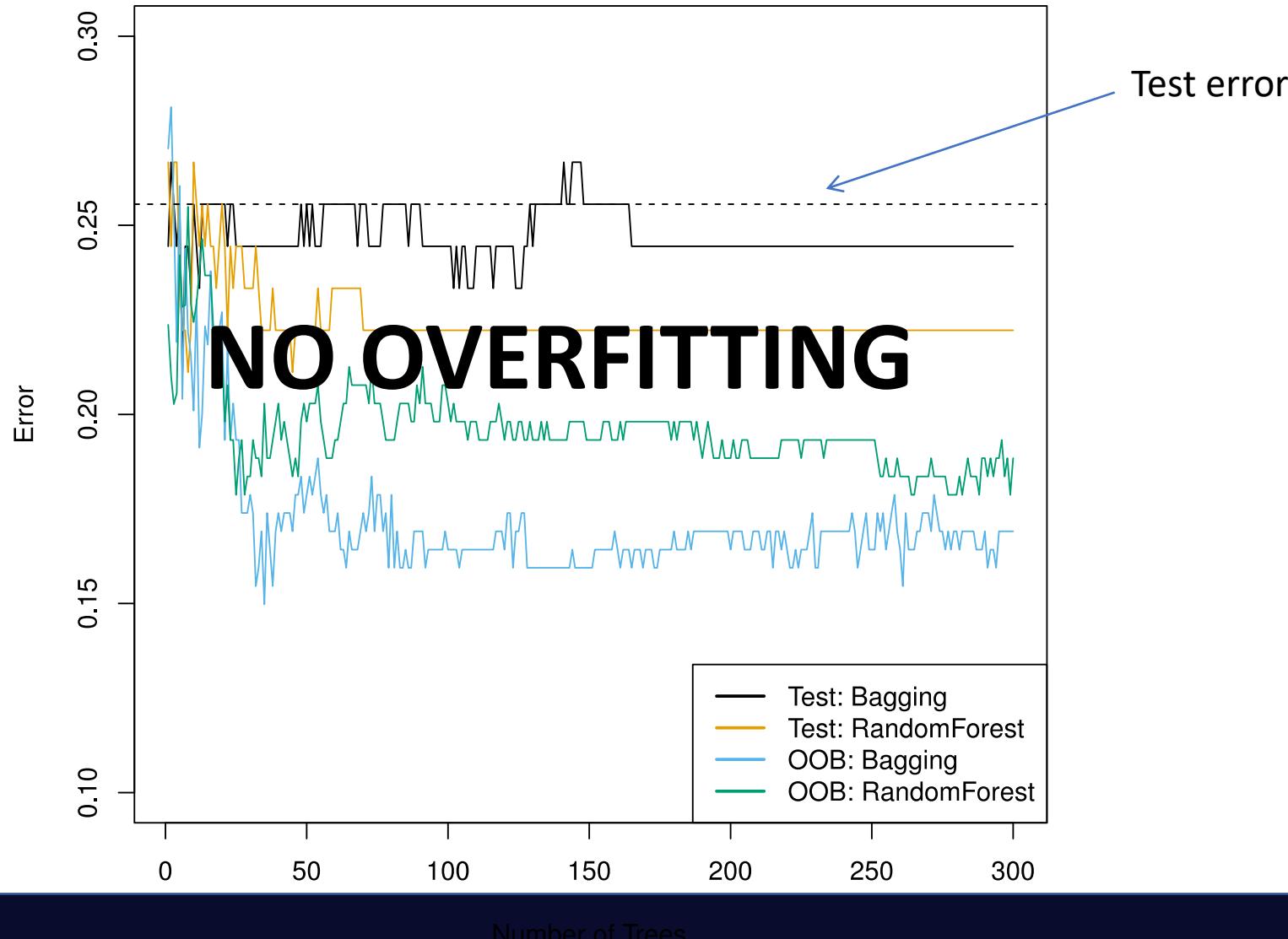
Bootstrap

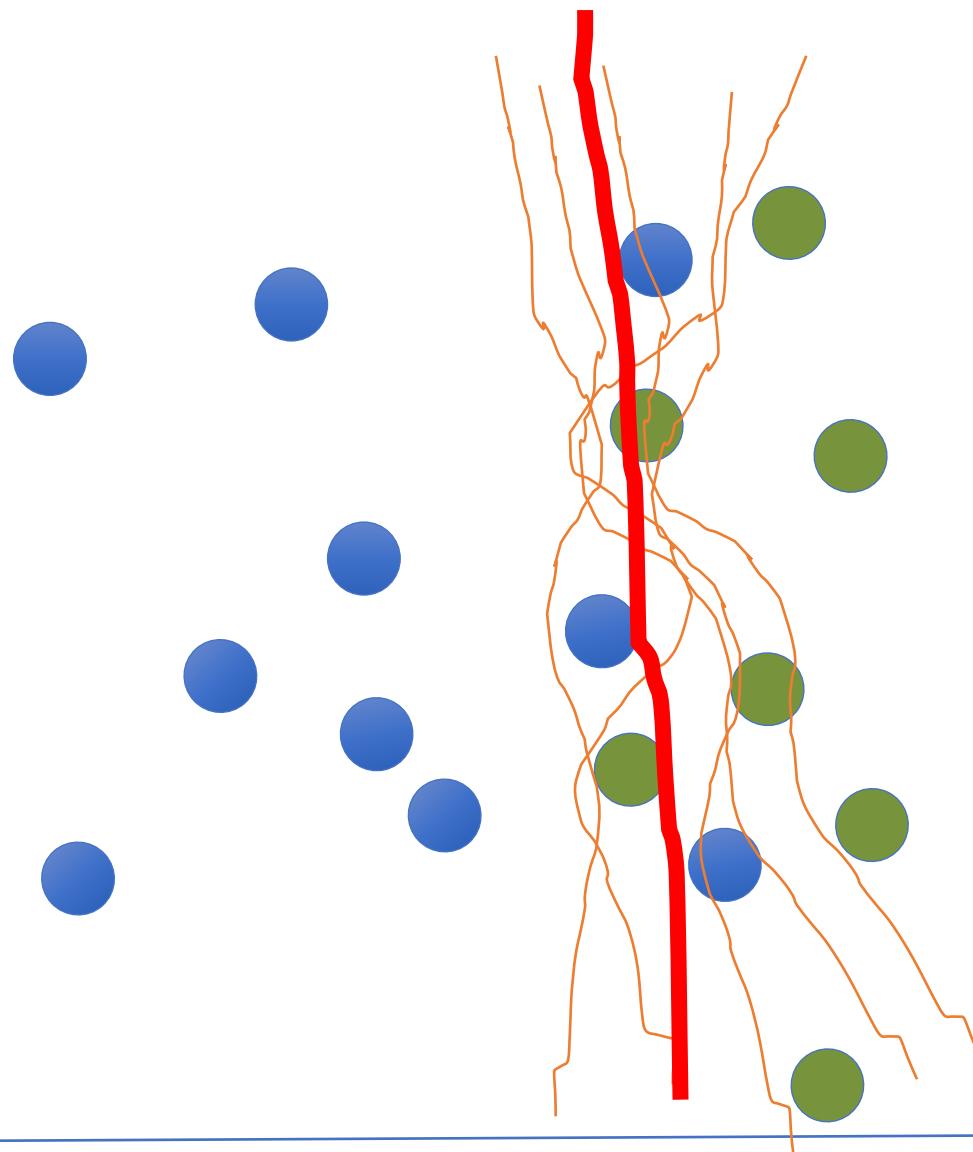
Construct B (hundreds) of trees (no pruning)

Learn a classifier for each bootstrap sample and
average them

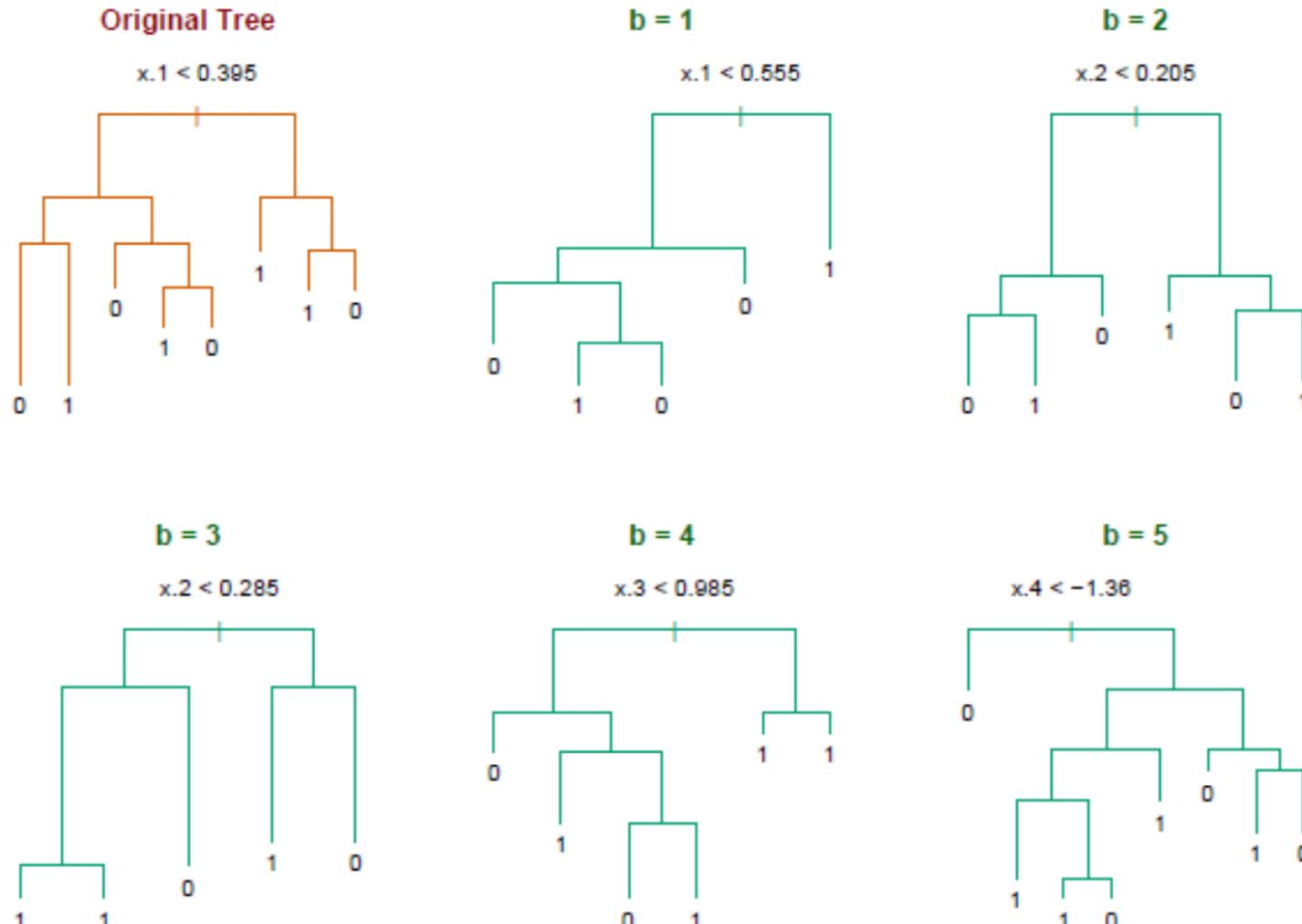
Very effective

Bagging for classification: Majority vote



x_2  x_1 

Bagging decision trees



Out-of-Bag Error Estimation

- No cross validation?
- Remember, in bootstrapping we sample with replacement, and therefore **not all observations are used for each bootstrap sample**. On average 1/3 of them are not used!
- We call them out-of-bag samples (OOB)
- We can predict the response for the i -th observation using each of the trees in which that observation was OOB and do this for n observations
- Calculate overall OOB MSE or classification error

Bagging

- Reduces overfitting (variance)
- Normally uses one type of classifier
- Decision trees are popular
- Easy to parallelize

Bagging - issues

Each tree is identically distributed (i.d.):

the expectation of the average of B such trees is the same as the expectation of any one of them

→ the bias of bagged trees is the same as that of the individual trees

i.d. and not i.i.d

Bagging - issues

An average of B i.i.d. random variables, each with variance σ^2 , has variance: σ^2/B

If i.d. (identical but not independent) and pair correlation ρ is present, then the variance is:

$$\rho \sigma^2 + \frac{1 - \rho}{B} \sigma^2$$

As B increases the second term disappears but the first term remains

Why does bagging generate correlated trees?

Bagging - issues

Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors.

Then all bagged trees will select the strong predictor at the top of the tree and therefore all trees will look similar.

How do we avoid this?

Bagging - issues

We can penalize the splitting (like in pruning) with a penalty term that depends on the number of times a predictor is selected at a given length

We can restrict how many times a predictor can be used

We only allow a certain number of predictors

Bagging - issues

Remember we want i.i.d such as the bias to be the same and variance to be less?

Other ideas?

What if we consider only a subset of the predictors at each split?

We will still get correlated trees unless

we **randomly** select the subset !

Random Forests

As in bagging, we build a number of decision trees on bootstrapped training samples each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors.

Note that if $m = p$, then this is bagging.

Random Forests Algorithm

For $b = 1$ to B :

- (a) Draw a bootstrap sample Z^* of size N from the training data.
- (b) Grow a random-forest tree to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
 - i. Select m variables at random from the p variables.
 - ii. Pick the best variable/split-point among the m .
 - iii. Split the node into two daughter nodes.

Output the ensemble of trees.

To make a prediction at a new point x we do:

For regression: average the results

For classification: majority vote

Random Forests Tuning

Good practices are:

- For classification, the default value for m is \sqrt{p} and the minimum node size is one.
- For regression, the default value for m is $p/3$ and the minimum node size is five.

In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters.

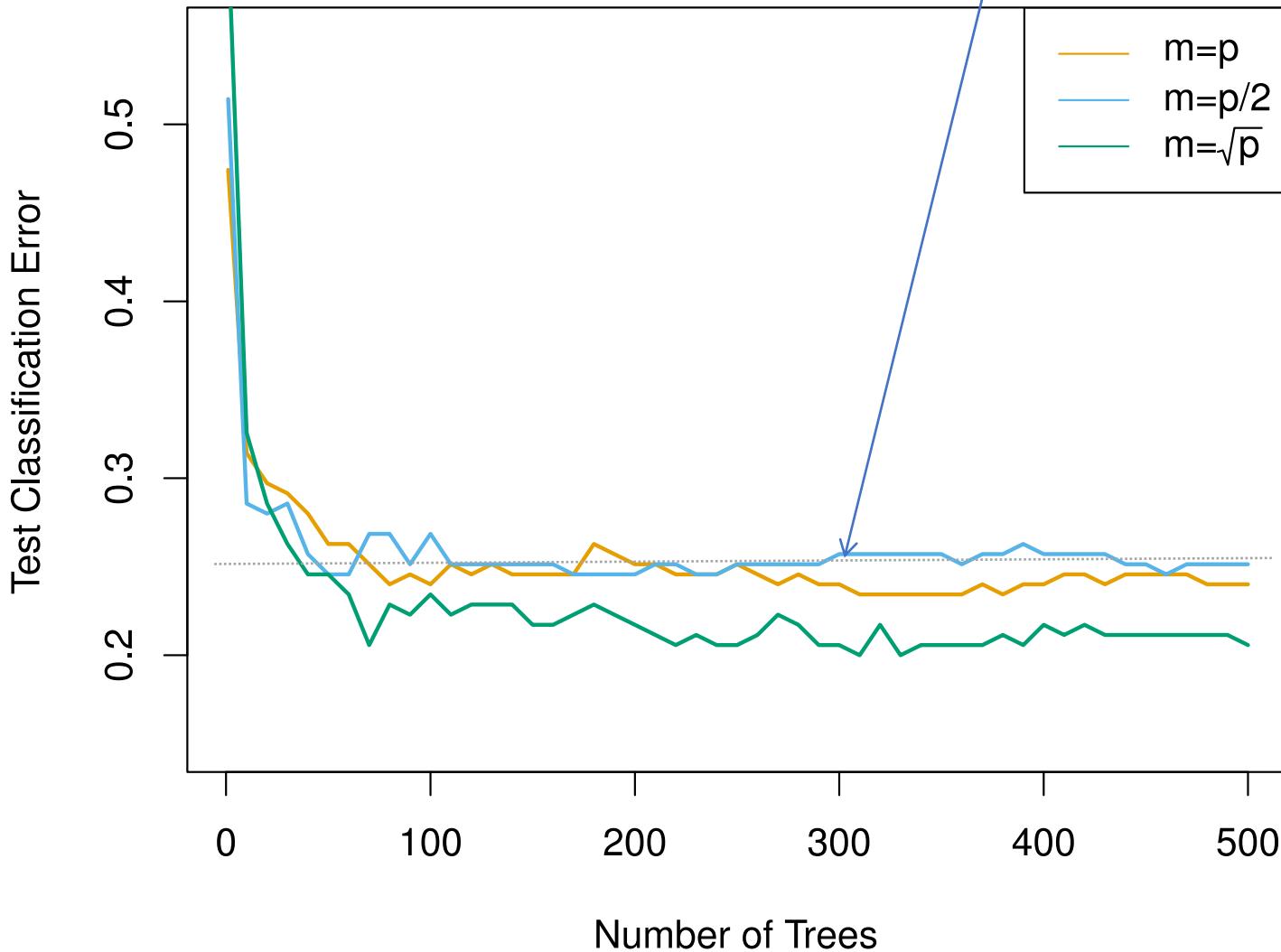
Like with Bagging, we can use OOB and therefore RF can be fit in one sequence, with cross-validation being performed along the way. Once the OOB error stabilizes, the training can be terminated.

Example

- 4,718 genes measured on tissue samples from 349 patients.
- Each gene has different expression
- Each of the patient samples has a qualitative label with 15 different levels: either normal or 1 of 14 different types of cancer.

Use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.

Null choice (Normal)

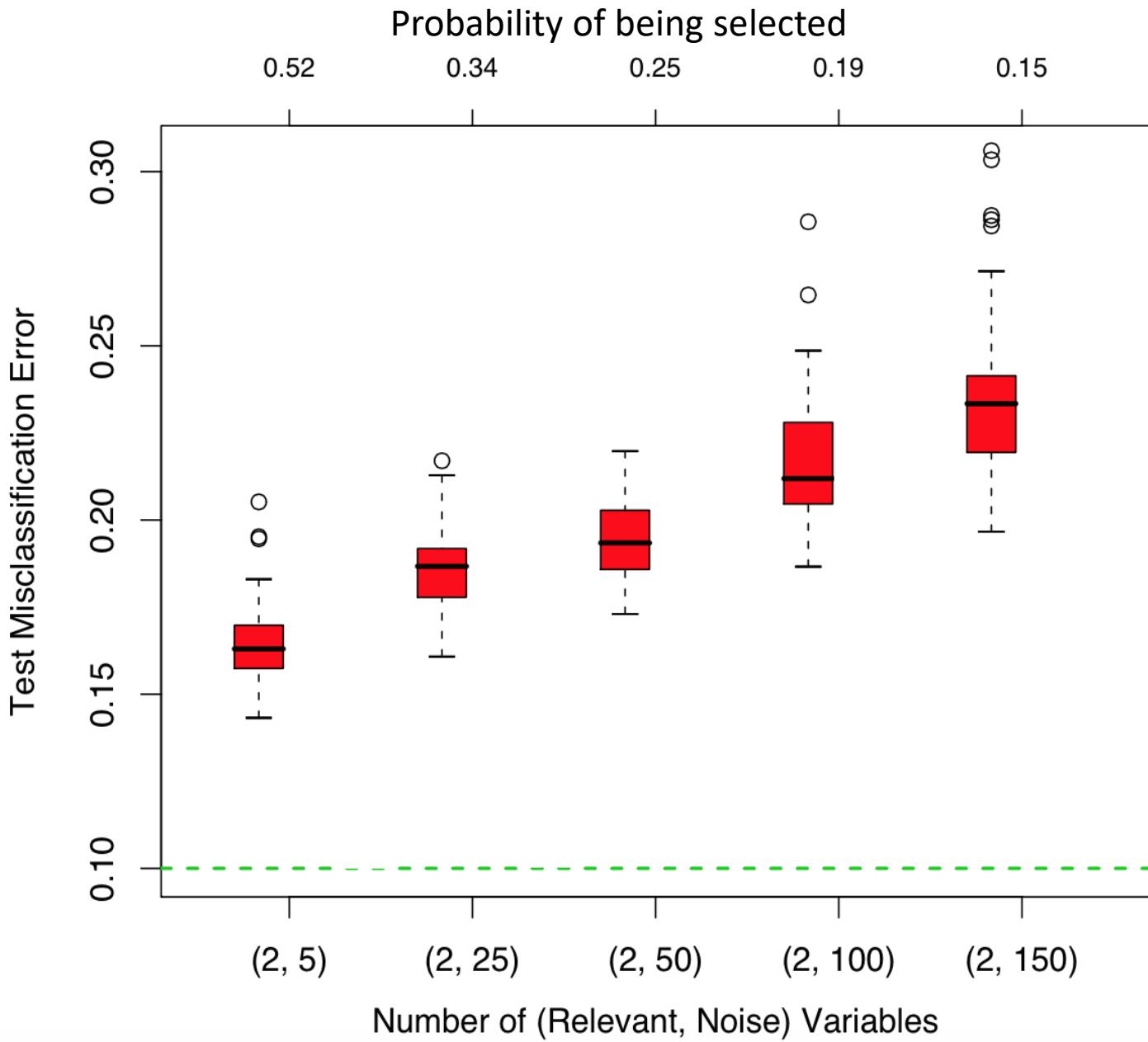


Random Forests Issues

When the number of variables is large, but the fraction of relevant variables is small, random forests are likely to perform poorly when m is small

At each split the chance can be small that the relevant variables will be selected

For example, with 3 relevant and 100 not so relevant variables the probability of any of the relevant variables being selected at any split is ~ 0.25



Can RF overfit?

Random forests “cannot overfit” the data wrt to number of trees.

Why?

The number of trees, B does not mean increase in the flexibility of the model

Boosting

- The term ‘Boosting’ refers to a family of algorithms which converts weak learner to strong learners.

Weak learners

- How would you classify an email as SPAM or not? Like everyone else, our initial approach would be to identify ‘spam’ and ‘not spam’ emails using following criteria. If:
 - Email has only one image file (promotional image), It’s a SPAM
 - Email has only link(s), It’s a SPAM
 - Email body consist of sentence like “You won a prize money of \$ xxxxxx”, It’s a SPAM
 - Email from known source, Not a SPAM
- Above, we’ve defined multiple rules to classify an email into ‘spam’ or ‘not spam’. But, do you think these rules individually are strong enough to successfully classify an email? No.
- Individually, these rules are not powerful enough to classify an email into ‘spam’ or ‘not spam’. Therefore, these rules are called as **weak learner** (slightly better than a random guess).

Weak learners

- To convert weak learner to strong learner, we'll combine the prediction of each weak learner using methods like:
 - Using average/ weighted average
 - Considering prediction has higher vote
- For example: Above, we have defined 5 weak learners. Out of these 5, 3 are voted as 'SPAM' and 2 are voted as 'Not a SPAM'. In this case, by default, we'll consider an email as SPAM because we have higher(3) vote for 'SPAM'.

Boosting

- To find weak rule, we apply base learning algorithms with a different distribution. Each time base learning algorithm is applied, it generates a new weak prediction rule. This is an iterative process. After many iterations, the boosting algorithm combines these weak rules into a single strong prediction rule.
- For choosing the right distribution, here are the following steps:
 - Step 1:* The base learner takes all the distributions and assign equal weight or attention to each observation.
 - Step 2:* If there is any prediction error caused by first base learning algorithm, then we pay higher attention to observations having prediction error. Then, we apply the next base learning algorithm.
 - Step 3:* Iterate Step 2 till the limit of base learning algorithm is reached or higher accuracy is achieved.
- Finally, it combines the outputs from weak learner and creates a strong learner which eventually improves the prediction power of the model. Boosting pays higher focus on examples which are misclassified or have higher errors by preceding weak rules.

Types of Boosting Algorithms

- Underlying engine used for boosting algorithms can be anything. There are many boosting algorithms which use other types of engine such as:
 - AdaBoost (**Adaptive Boosting**)
 - Gradient Tree Boosting
 - *GentleBoost*
 - *LPBoost*
 - *BrownBoost*
 - XGBoost
 - CatBoost
 - Lightgbm

Gradient Tree Boosting

Bagging: Generate multiple trees from bootstrapped data and average the trees.

Recall bagging results in i.d. trees and not i.i.d.

RF produces i.i.d (or more independent) trees by randomly selecting a subset of predictors at each step

Gradient Tree Boosting

Unlike Bagging and RF Boosting does not involve bootstrap sampling

1. Trees are grown sequentially: each tree is grown using information from previously grown trees
2. Like bagging, boosting involves combining a large number of decision trees, f^1, \dots, f^B

Gradient Tree Boosting

Given the current model,

- we fit a decision tree to the **residuals** from the model. Response variable now is the residuals and not Y
- We then add this new decision tree into the fitted function in order to update the residuals
- The learning rate has to be controlled

Boosting for regression

1. Set $f(x)=0$ and $r_i = y_i$ for all i in the training set.
2. For $b=1,2,\dots,B$, repeat:
 - a. Fit a tree with d splits (+1 terminal nodes) to the training data (X, r) .
 - b. Update the tree by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

- c. Update the residuals,

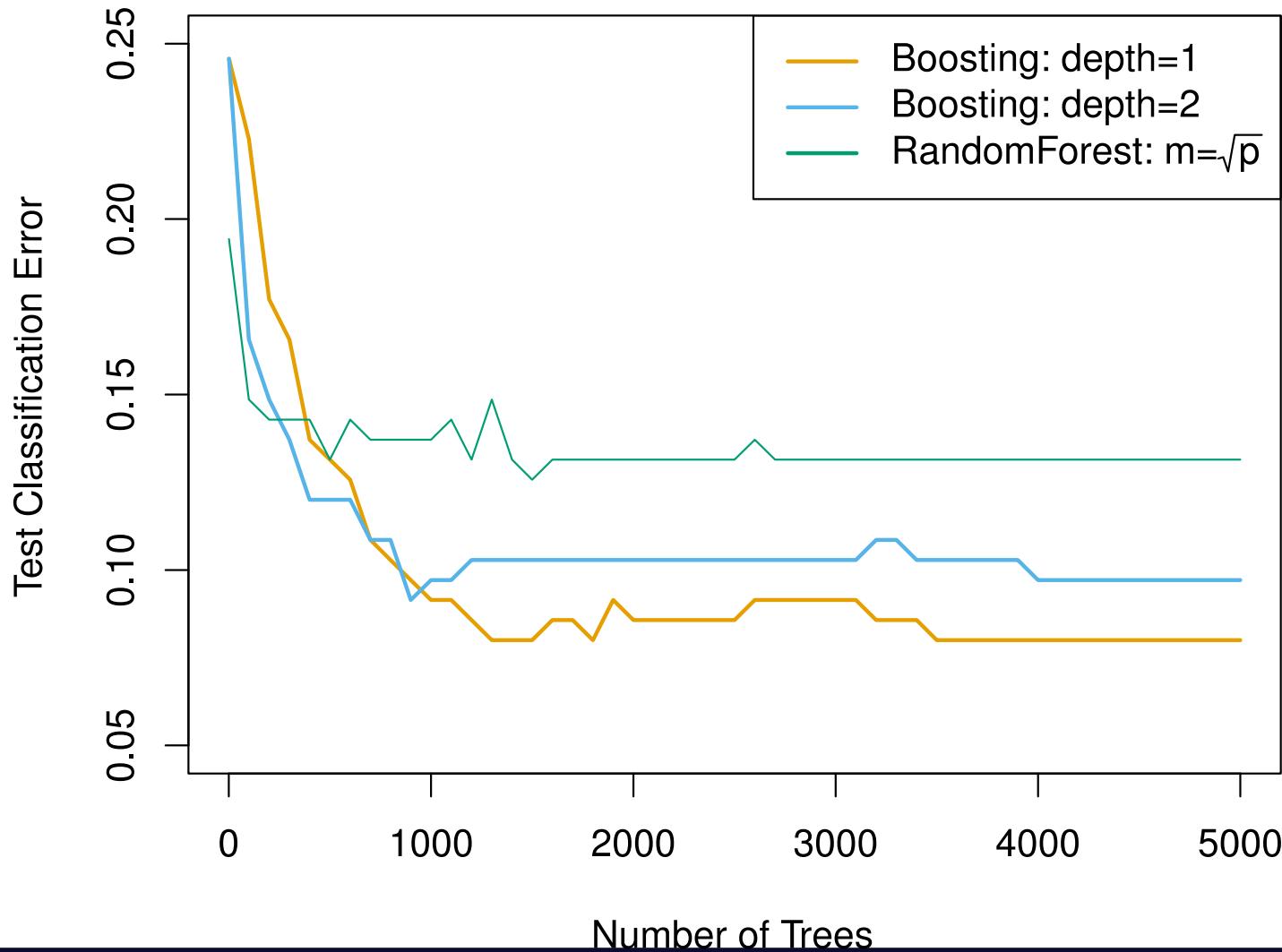
$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

3. Output the boosted model.

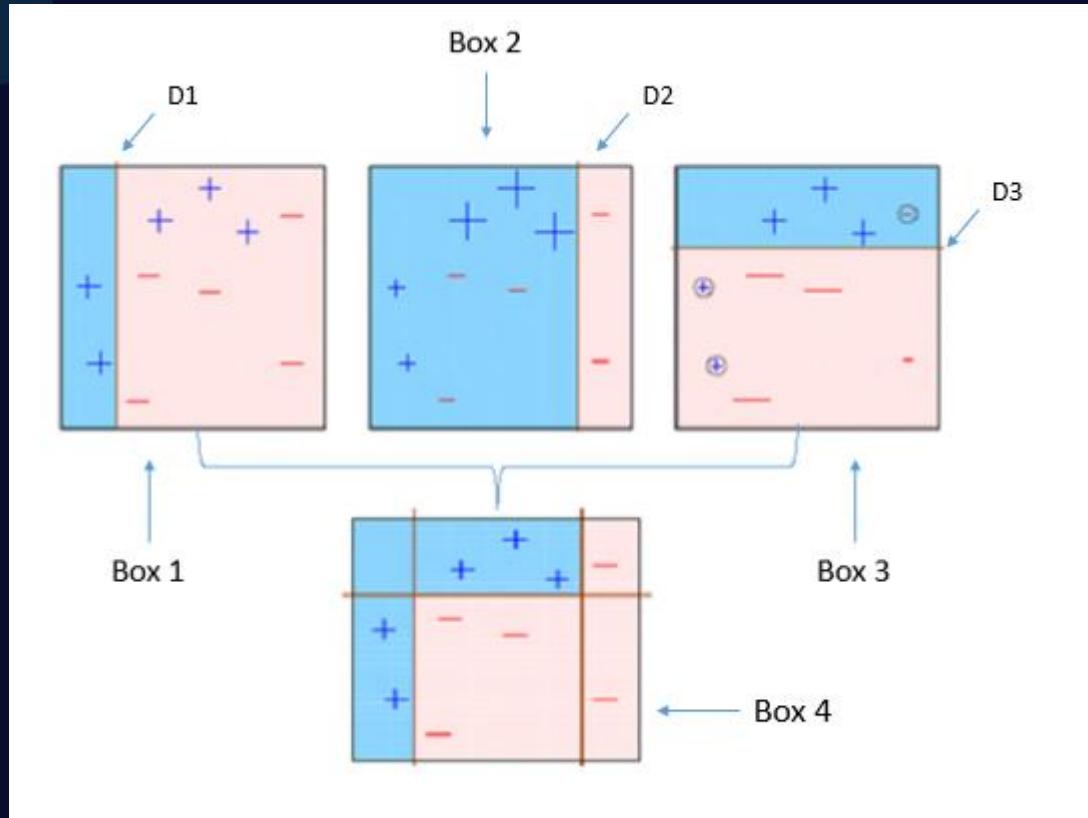
$$\hat{f}(x) = \sum_{b=1}^B \lambda \hat{f}^b(x)$$

Boosting tuning parameters

- The number of trees B . RF and Bagging do not overfit as B increases.
Boosting can overfit! **Cross Validation**
- The shrinkage parameter λ , a small positive number. Typical values are 0.01 or 0.001 but it depends on the problem. λ only controls the learning rate
- The number d of splits in each tree, which controls the complexity of the boosted ensemble. Stumpy trees, $d = 1$ works well.



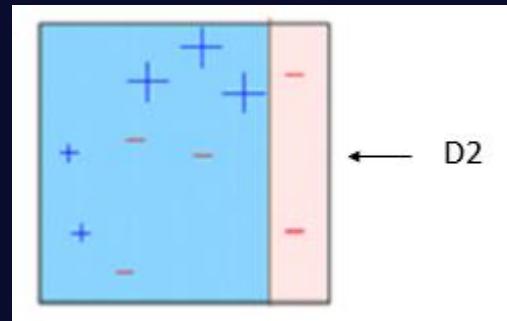
AdaBoost



- Box 1: You can see that we have assigned equal weights to each data point and applied a decision stump to classify them as + (plus) or - (minus). The decision stump (D1) has generated vertical line at left side to classify the data points. We see that, this vertical line has incorrectly predicted three + (plus) as - (minus). In such case, we'll assign higher weights to these three + (plus) and apply another decision stump.

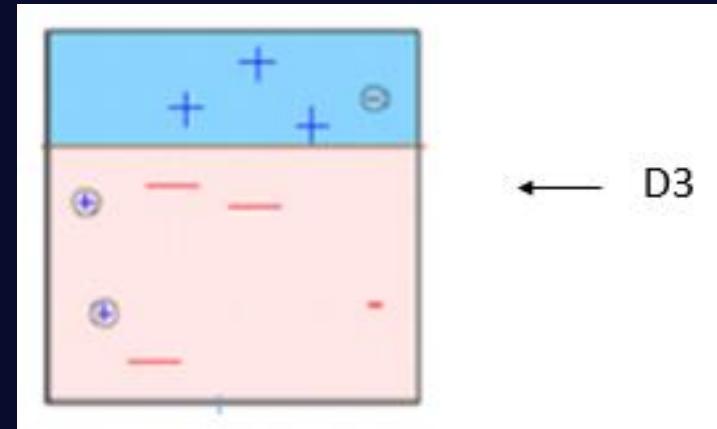
AdaBoost

- Box 2: The size of three incorrectly predicted + (plus) is bigger as compared to rest of the data points. In this case, the second decision stump (D2) will try to predict them correctly. Now, a vertical line (D2) at right side of this box has classified three misclassified + (plus) correctly. But again, it has caused misclassification errors. This time with three - (minus). Again, we will assign higher weight to three – (minus) and apply another decision stump.



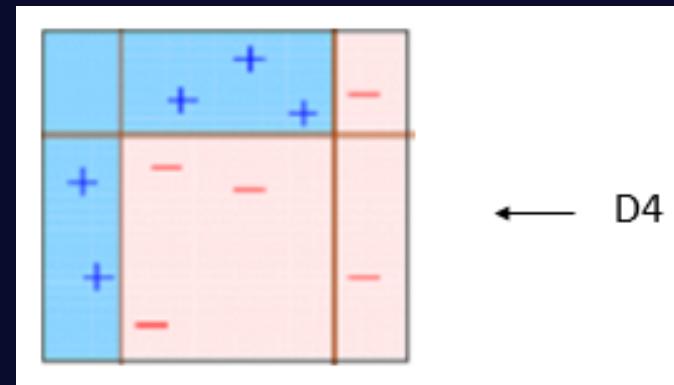
Adaboost

- Box 3: Here, three – (minus) are given higher weights. A decision stump (D3) is applied to predict these misclassified observation correctly. This time a horizontal line is generated to classify + (plus) and – (minus) based on higher weight of misclassified observation.



Adaboost

- Box 4: Here, we have combined D1, D2 and D3 to form a strong prediction having complex rule as compared to individual weak learner. You can see that this algorithm has classified these observation quite well as compared to any of individual weak learner.

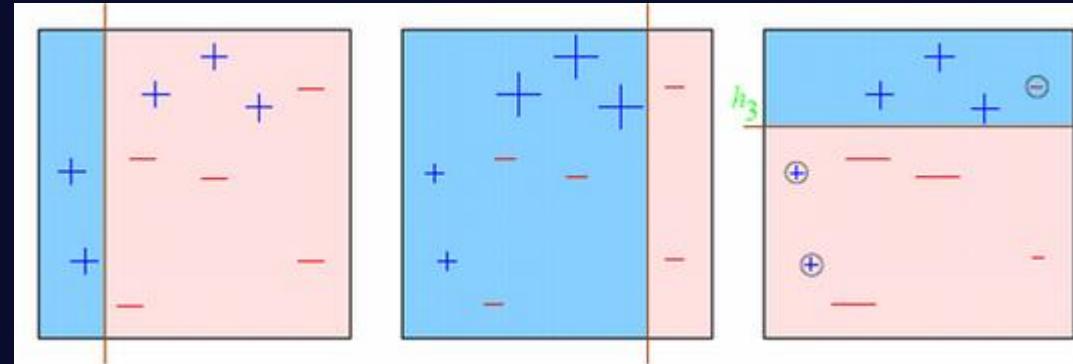


AdaBoost (Adaptive Boosting)

- It works on similar method as discussed above. It fits a sequence of weak learners on different weighted training data. It starts by predicting original data set and gives equal weight to each observation. If prediction is incorrect using the first learner, then it gives higher weight to observation which have been predicted incorrectly. Being an iterative process, it continues to add learner(s) until a limit is reached in the number of models or accuracy.
- Mostly, we use decision stamps with AdaBoost. But, we can use any machine learning algorithms as base learner if it accepts weight on training data set. We can use AdaBoost algorithms for both classification and regression problem.

Let's try to visualize one Classification Problem

- Look at the below diagram :



We start with the first box. We see one vertical line which becomes our first weak learner. Now in total we have 3/10 mis-classified observations. We now start giving higher weights to 3 plus mis-classified observations. Now, it becomes very important to classify them right. Hence, the vertical line towards right edge. We repeat this process and then combine each of the learner in appropriate weights.

Explaining underlying mathematics

- How do we assign weight to observations?
- We always start with a uniform distribution assumption. Lets call it as D_1 which is $1/n$ for all n observations.
- Step 1 . We assume an $\alpha(t)$
- Step 2: Get a weak classifier $h(t)$
- Step 3: Update the population distribution for the next step

where

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

$$Z_t = \sum_{i=1}^m D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

Explaining underlying mathematics

- Alpha is kind of learning rate, y is the actual response (+ 1 or -1) and h(x) will be the class predicted by learner. Essentially, if learner is going wrong, the exponent becomes $1 * \alpha$ and else $-1 * \alpha$. The weight will probably increase, if the prediction went wrong the last time.
- Step 4 : Use the new population distribution to again find the next learner
- Step 5 : Iterate step 1 – step 4 until no hypothesis is found which can improve further.
- Step 6 : Take a weighted average of the frontier using all the learners used till now. But what are the weights? Weights are simply the alpha values. Alpha is calculated as follows:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

Explaining underlying mathematics

- Output the final hypothesis

$$H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right).$$



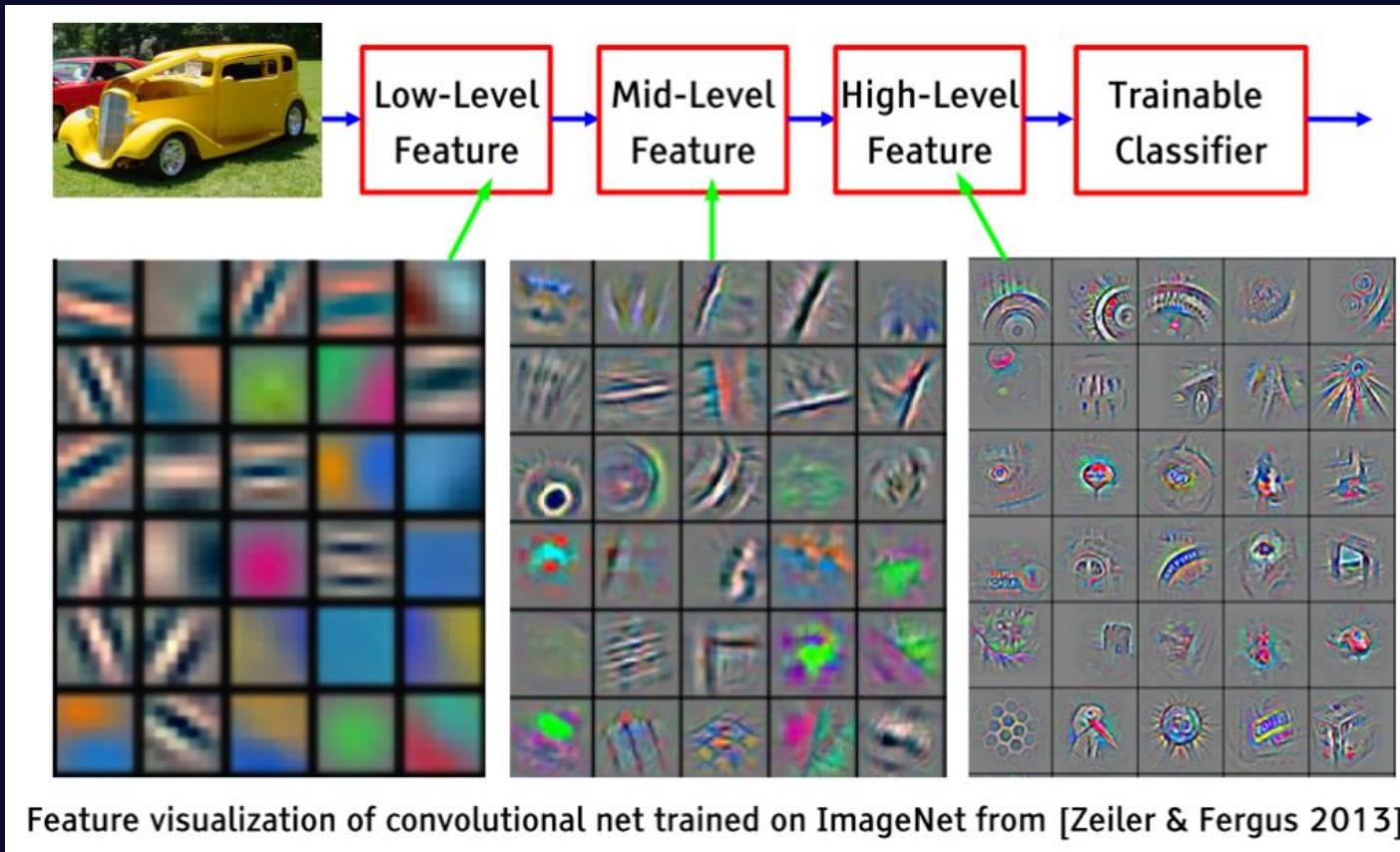
Introduction to deep learning

Background

- “Traditional machine learning” is about:
 - Feature engineering
 - Creating models and representations
 - Optimizing and predicting

Feature Learning

- Can we create a model to learn and extract hierarchical feature representations from data?

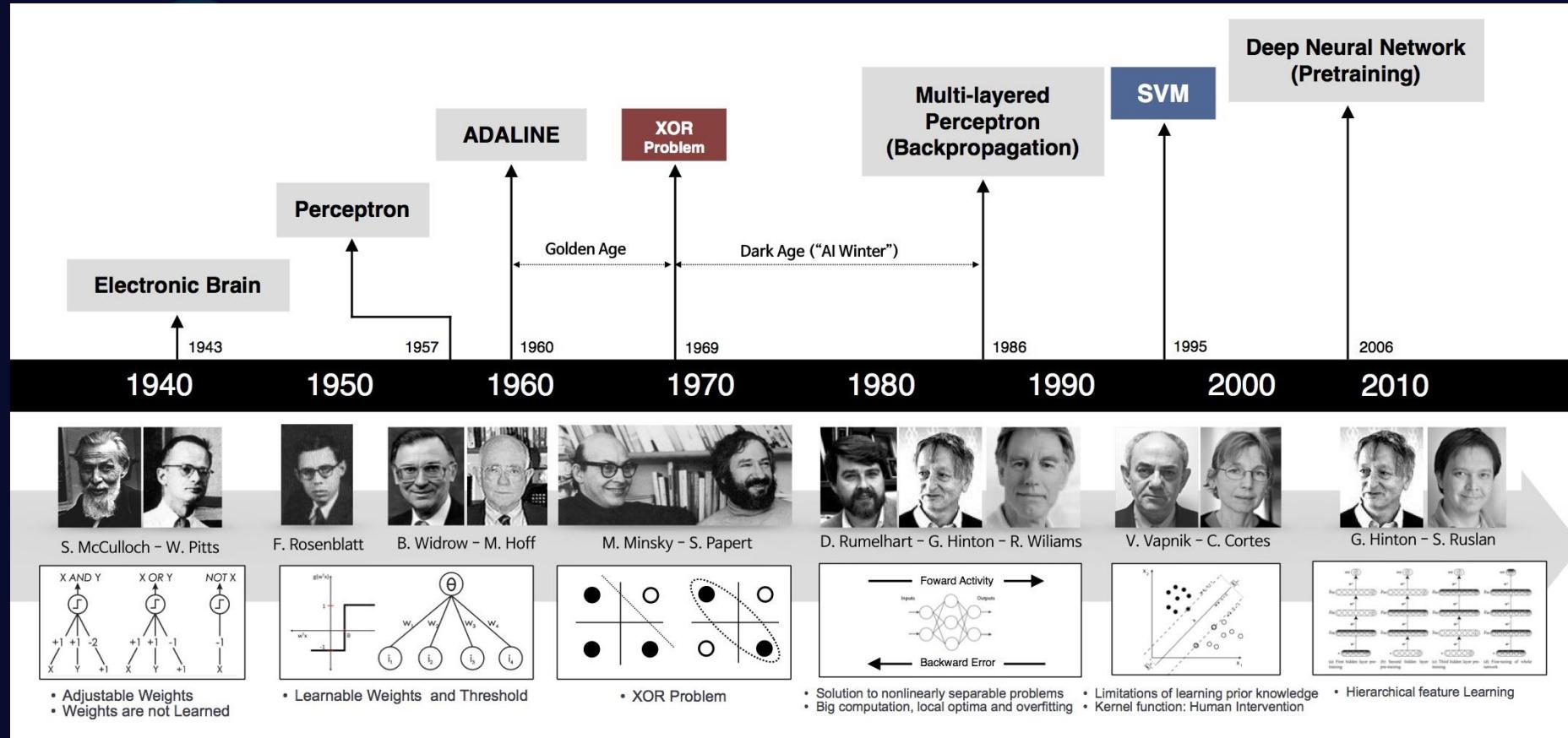


What is Deep Learning?

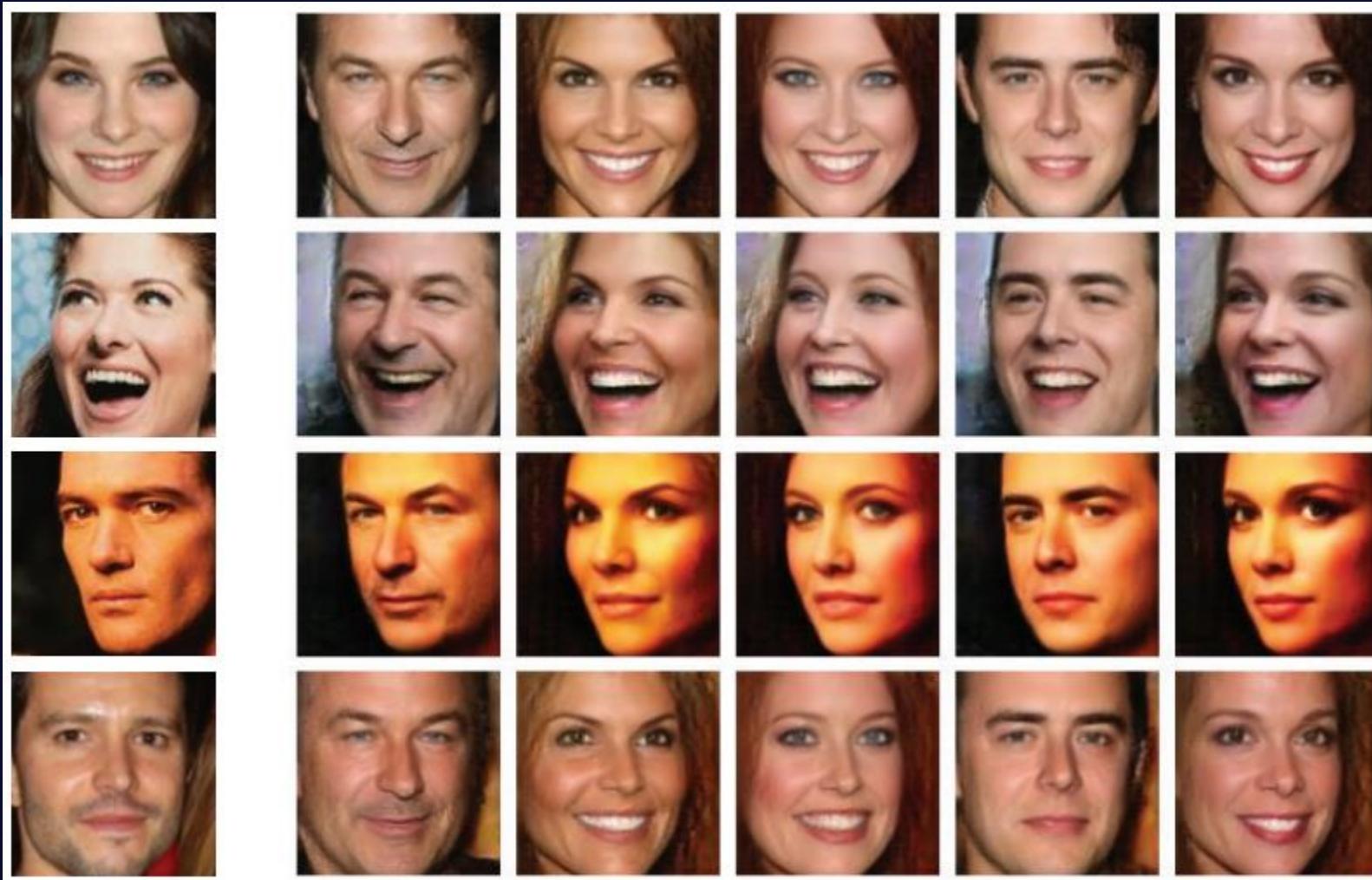
- **Deep learning** (deep structured learning or hierarchical learning) is a set of algorithms that attempt to model *high-level abstractions* in data by using model architectures composed of *multiple non-linear transformations*.

Usually a deep learning model is simply a: Neural networks with **more than 1 hidden layers**.

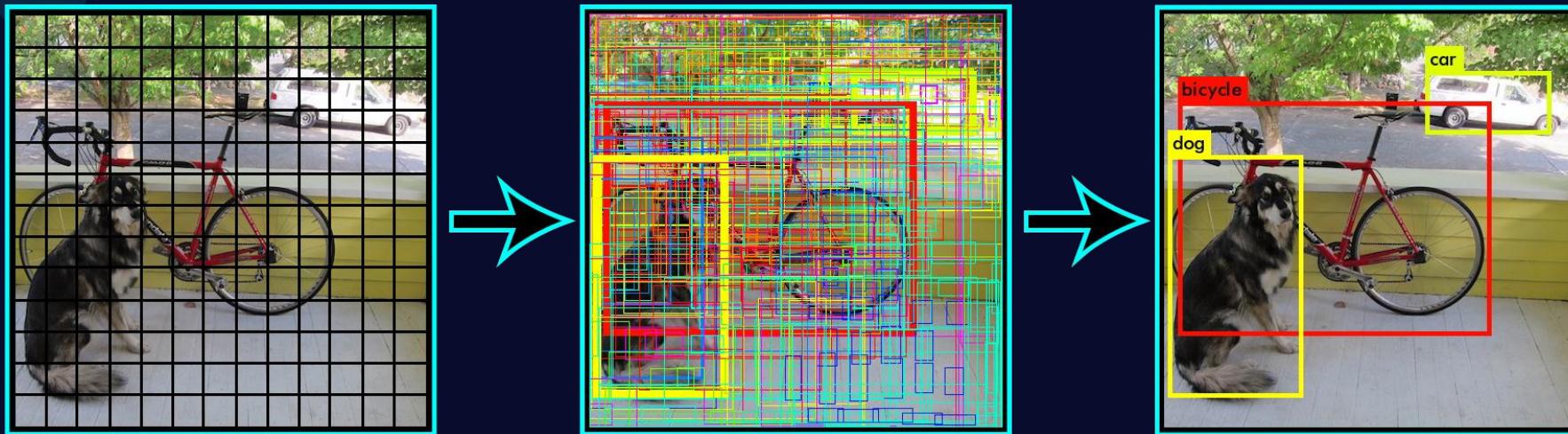
A Brief History of Neural Networks



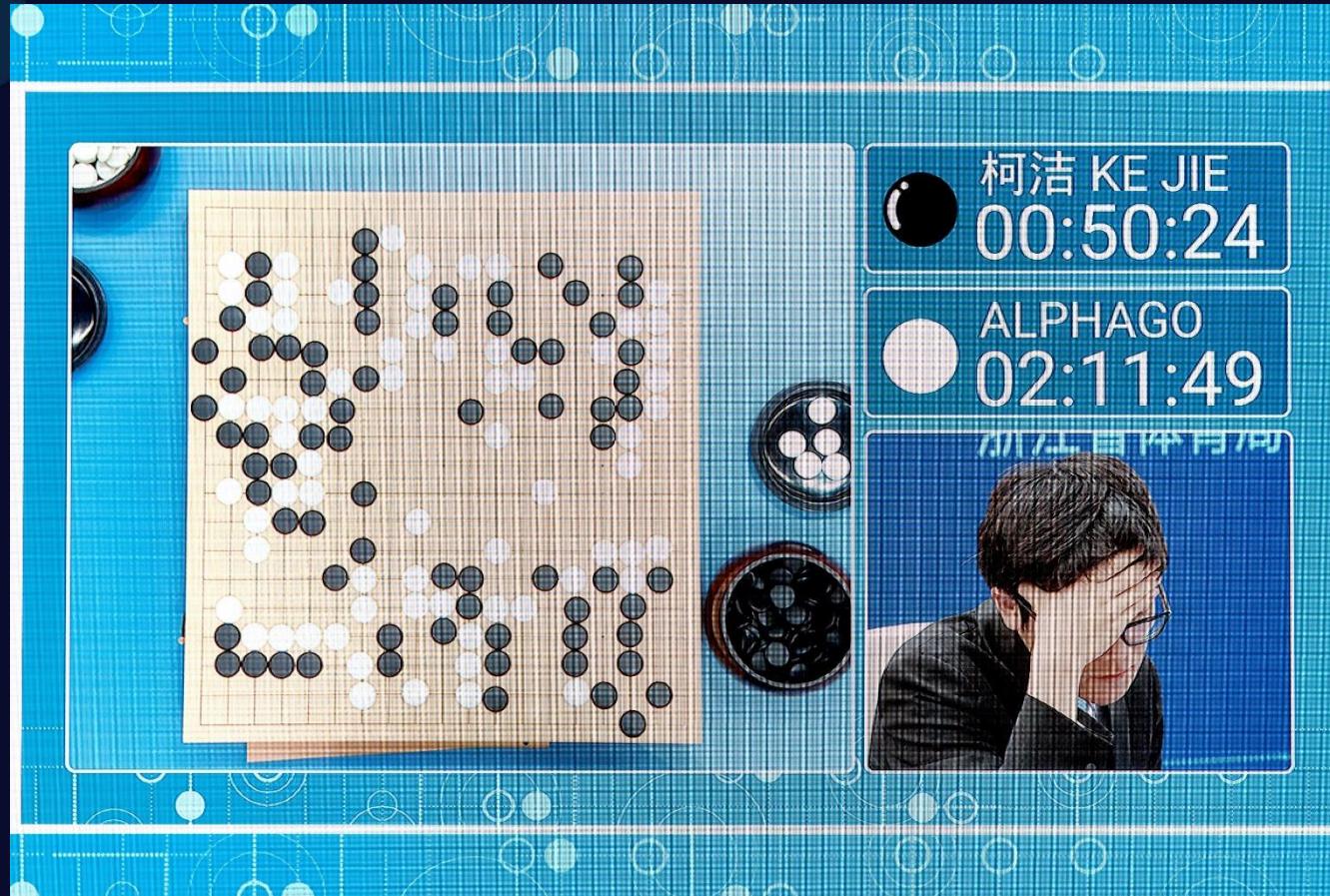
Motivation slides...



Object Detection



Play Go

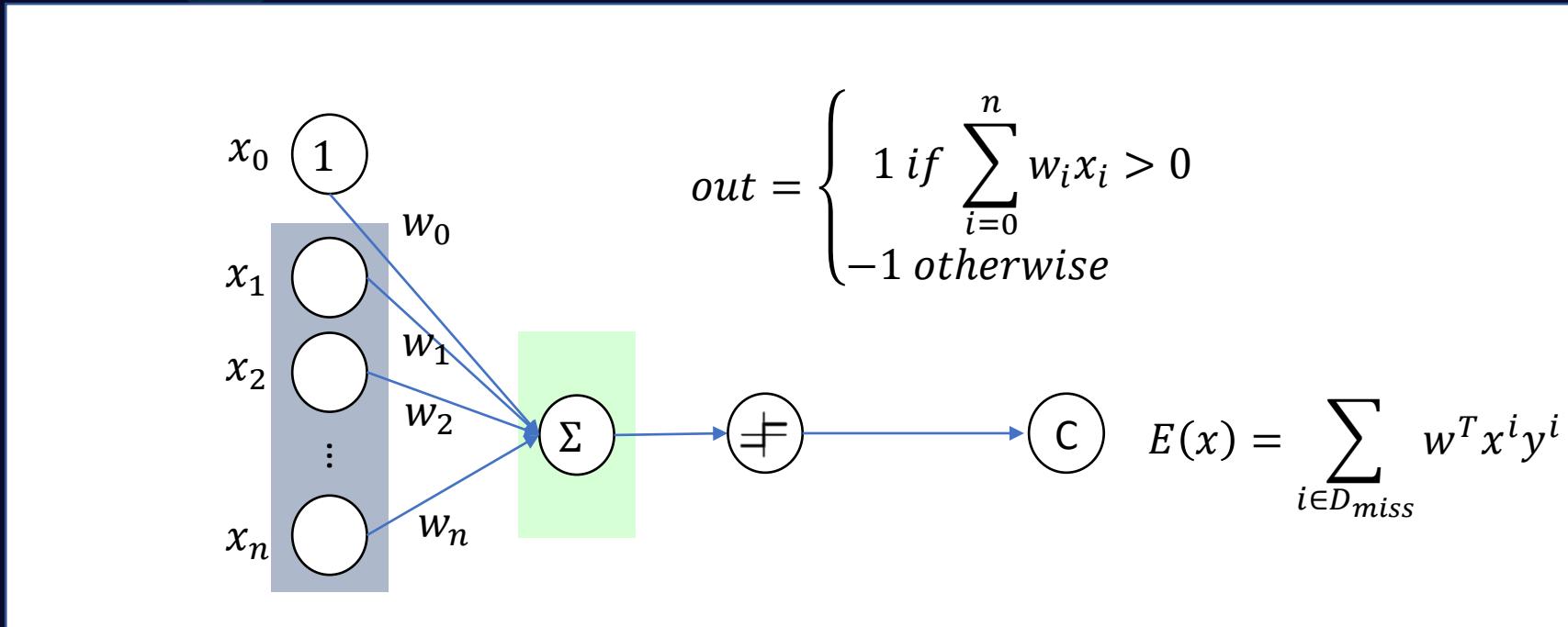




Building a neural network

Perceptron

- Recall the basic perceptron:



- Training using SGD
- Is it really SGD...?

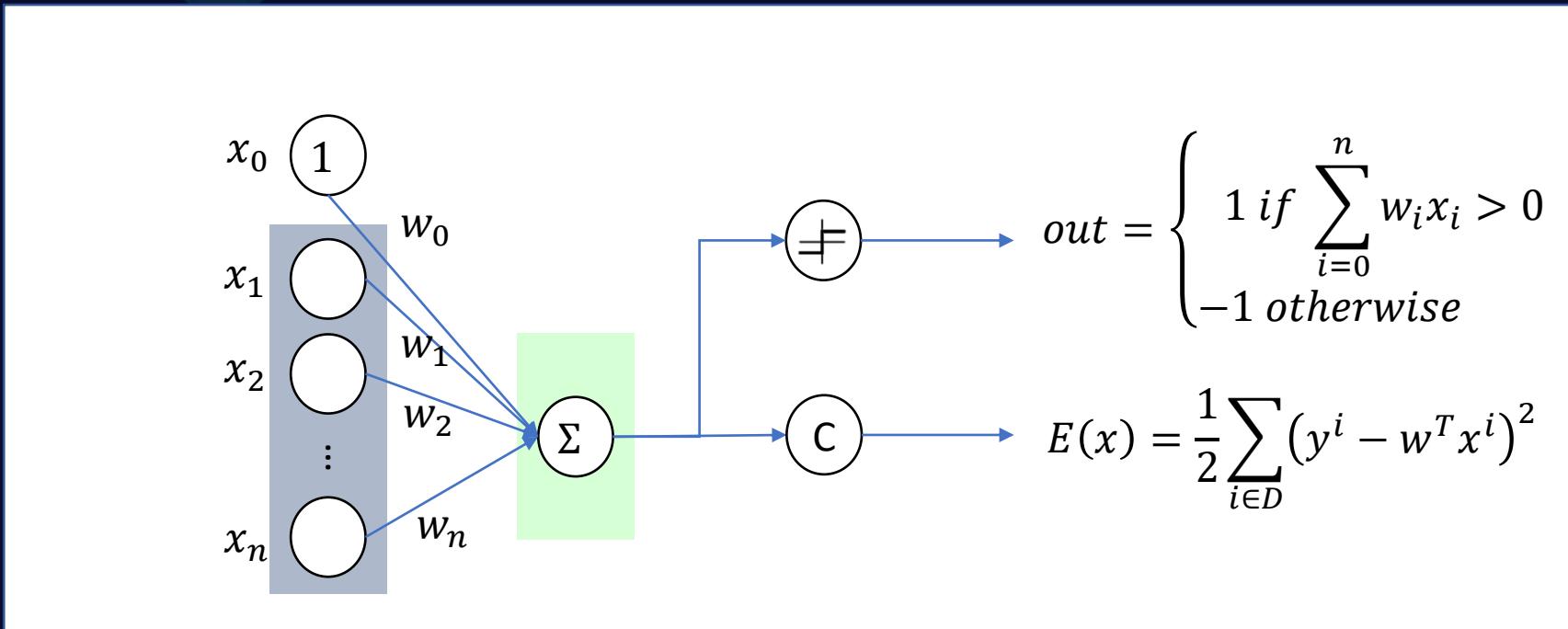
Training a perceptron

Algorithm 1 Perceptron Rule

```
1:  $\vec{w} \leftarrow \vec{w}_0$                                 ▷ Take  $\vec{w}_0$  to be a random vector with small components
2:  $ErrorFound \leftarrow True$ 
3: while  $ErrorFound$  do
4:    $ErrorFound \leftarrow False$ 
5:   for  $i = 1, \dots, N$  do
6:      $\hat{y}_i \leftarrow sgn(\vec{x}_i \cdot \vec{w}_t)$ 
7:     if  $\hat{y}_i \neq y_i$  then
8:        $ErrorFound \leftarrow True$ 
9:        $\vec{w} \leftarrow \vec{w} + \eta y_i \vec{x}_i$ 
10:      end if
11:    end for
12:  end while
13: return  $\vec{w}$ 
```

Adaptive Linear Neuron (Adaline)

- Same principle, different error function



- Can we use SGD?
- Yes!

Reminder - Modular approach (Lego)

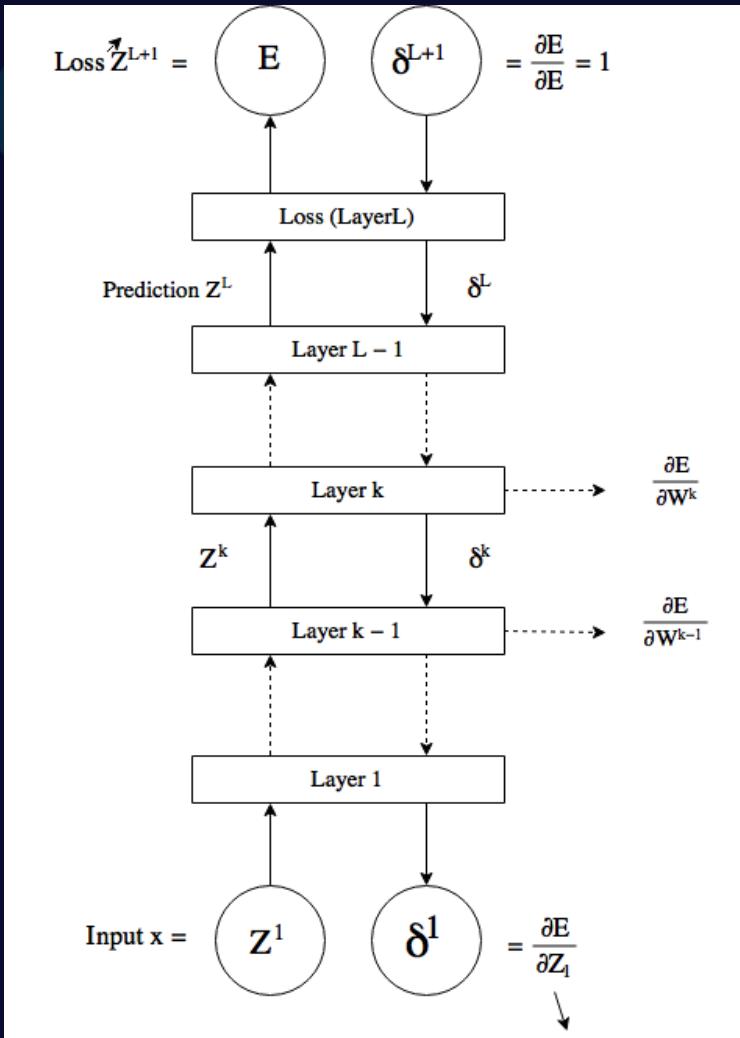
We code layers not networks!

Layer specification

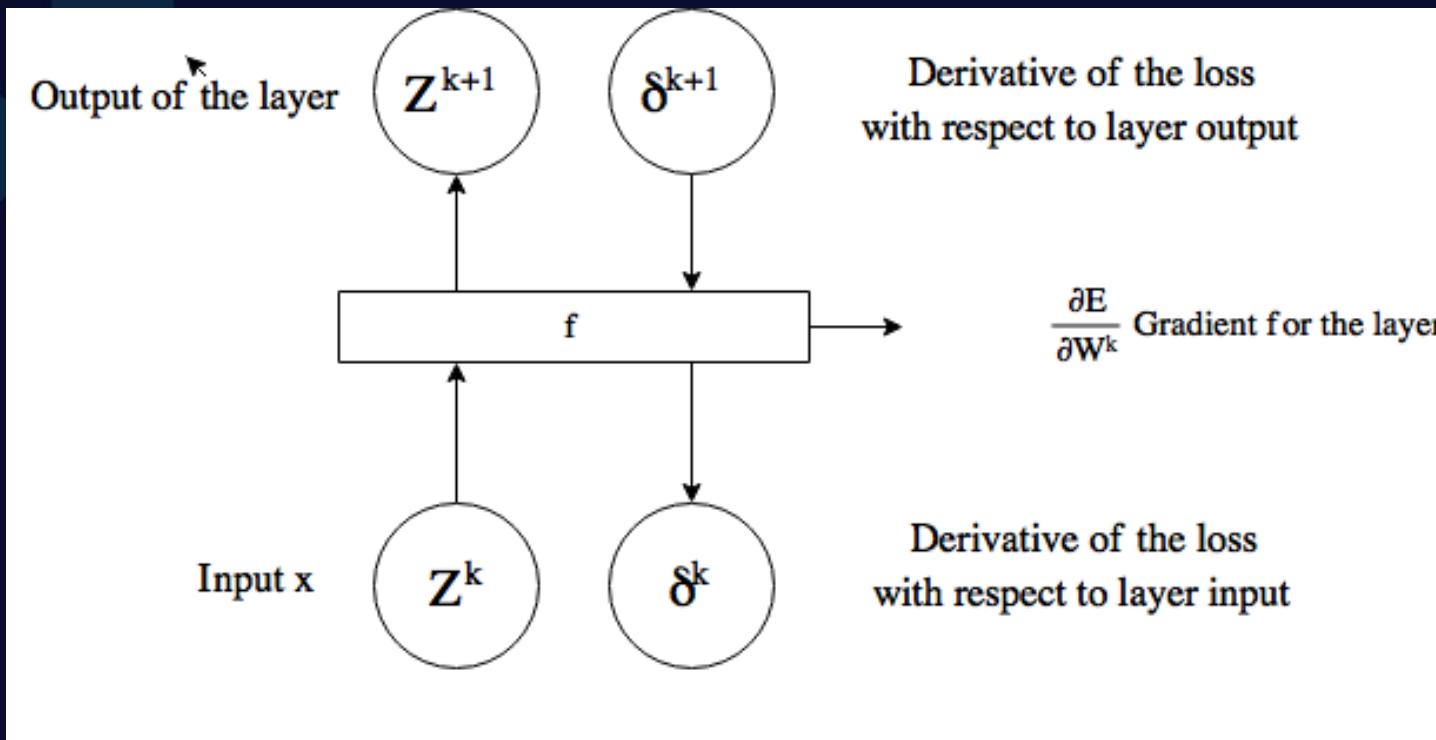
Each layer needs to provide three functions:

1. What is the layer output given its input (Forward)
2. Derivative with respect to the input
3. Derivative with respect to parameters

NN layer view



Single layer



Backpropagation

- Forward pass

$$Z^{k+1} = f(z^k)$$

- Backward pass

$$\delta^L = \frac{\partial E}{\partial Z^L}$$

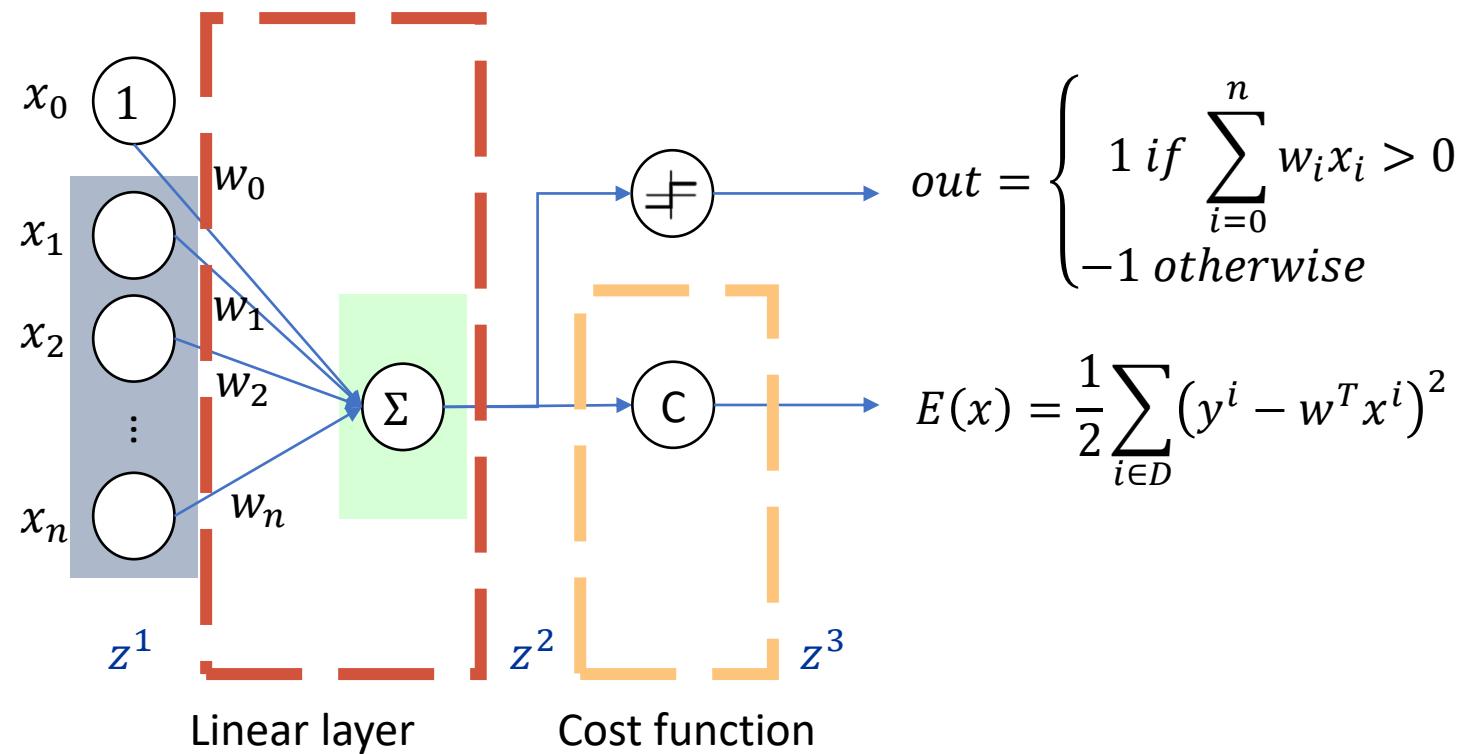
- Applying chain rule for a single layer

$$\frac{\partial E}{\partial Z^k} = \frac{\partial E}{\partial Z^{k+1}} \frac{\partial Z^{k+1}}{\partial Z^k} = \delta^{k+1} \frac{\partial Z^{k+1}}{\partial Z^k}$$

- Gradient with respect to layer parameters if it has parameters

$$\frac{\partial E}{\partial W^k} = \frac{\partial E}{\partial Z^{k+1}} \frac{\partial Z^{k+1}}{\partial W^k} = \delta^{k+1} \frac{\partial Z^{k+1}}{\partial W^k}$$

Adaline – Layer View



Adaline derivatives

- Error layer:

$$E(z) = \frac{1}{2} \sum_{i \in D} (y^i - z^i)^2 = \frac{1}{2} \|Y - Z\|^2$$

$$\frac{\partial E}{\partial z} = - \sum_{i \in D} (y^i - z^i) \nabla z^i = -(Y - Z) \nabla Z$$

- Linear layer:

$$z = f(x) = w^T X$$

$$\frac{\partial f}{\partial w} = \sum_{i \in D} x^i = X$$

Together:

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial z} \frac{\partial z}{\partial w} = - \sum_{i \in D} (y^i - w^T x^i) x^i = -(Y - w^T X)$$

Stochastic Gradient Descent

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k .

Require: Initial parameter θ

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

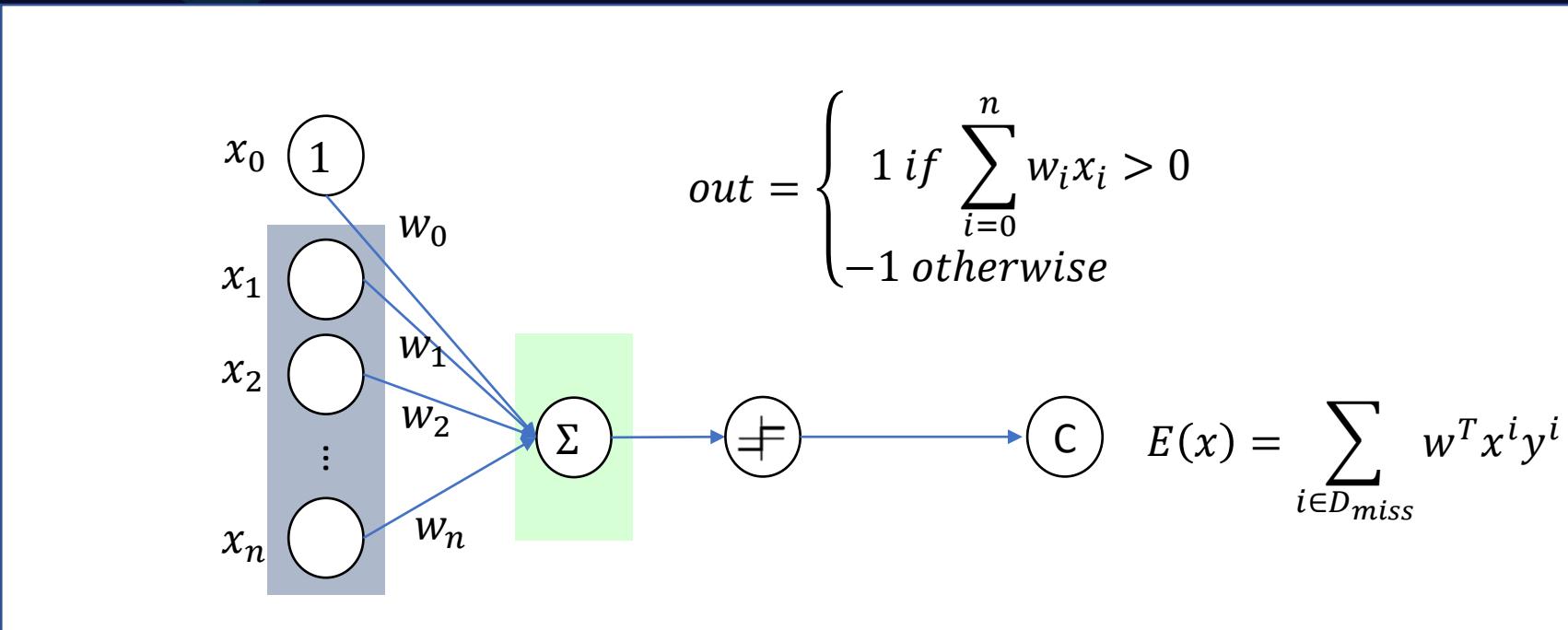
 Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 Apply update: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

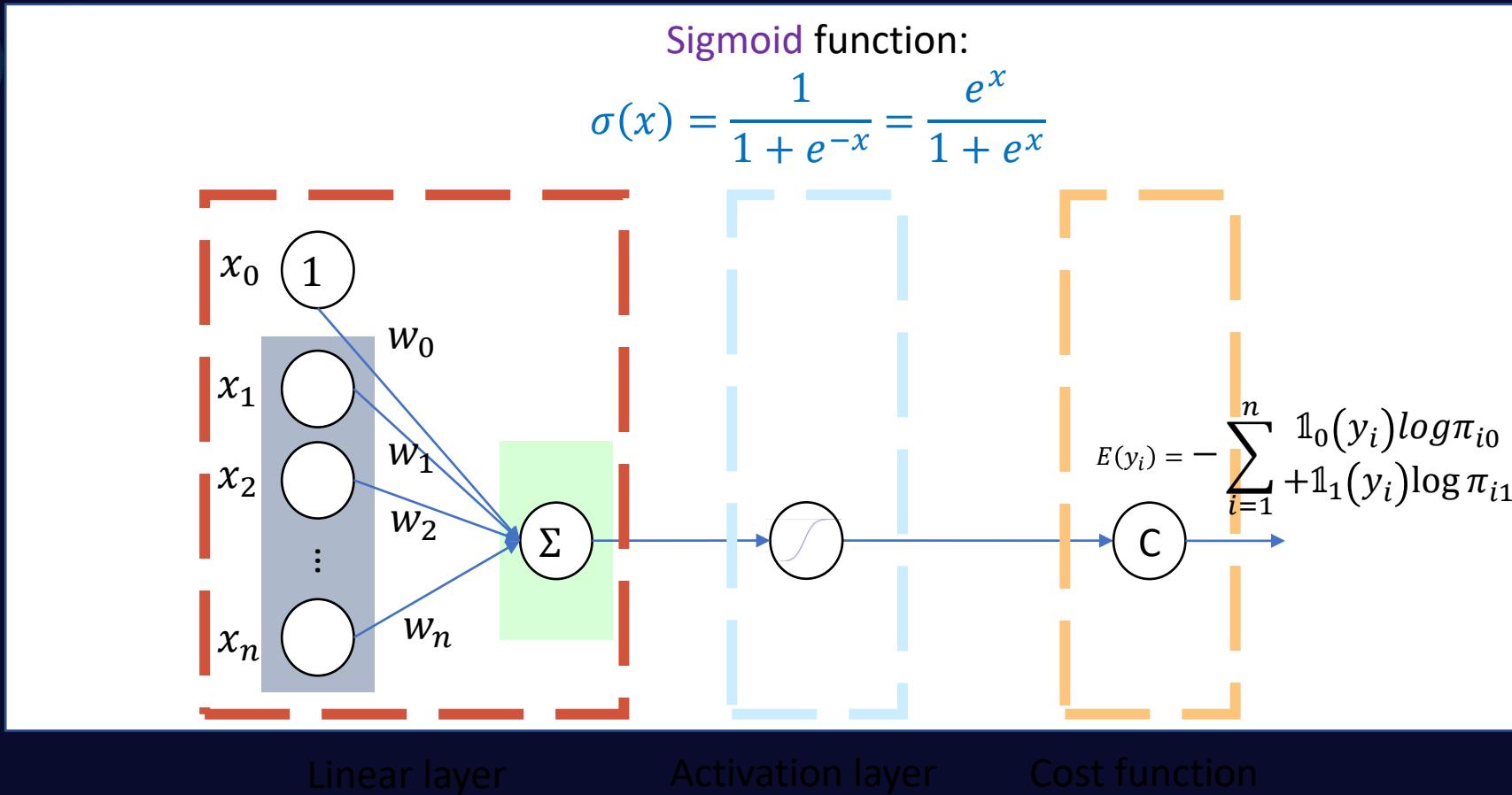
end while

The Perceptron – a variation

- We can replace the step function with a sigmoid:



The Perceptron – a variation



Perceptron variation output

- For convenience we'll use the following notations:

$$P(y_i|x_i, w) = \begin{cases} \pi_{i1} \equiv \sigma(w^T x) = \frac{1}{1 - e^{-w^T x}}, & \text{if } y_i = 1 \\ \pi_{i0} \equiv 1 - \sigma(w^T x) = 1 - \frac{1}{1 - e^{-w^T x}}, & \text{if } y_i = 0 \end{cases}$$

Logistic Regression

- Expanding the expression

$$P(y|x, \theta) = Ber(y|\sigma(w^T x))$$

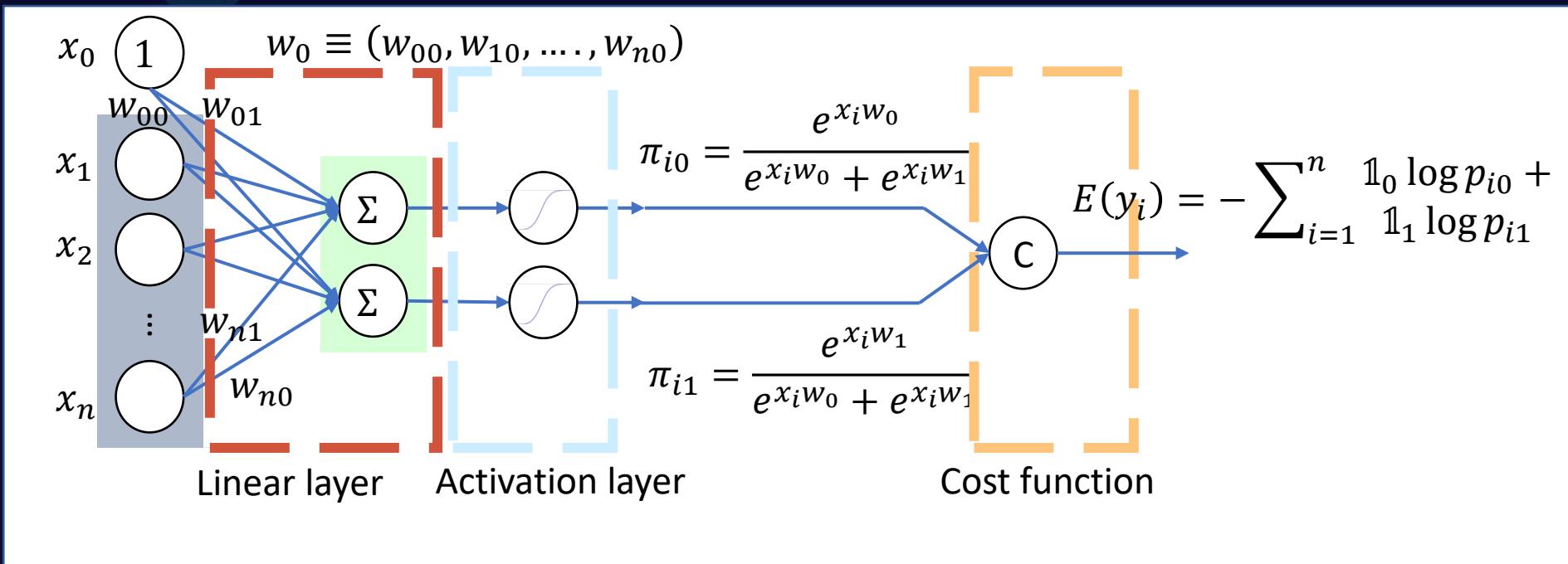
- we get:

$$\begin{aligned} NLL(\theta) &= \sum_{i=1}^n \log \pi_{i1}^{y_i} \pi_{i0}^{1-y_i} \\ &= - \left(\sum_{i=1}^n y_i \log \pi_{i1} + (1 - y_i) \log \pi_{i0} \right) \end{aligned}$$

- The expression we get is known as **cross-entropy**

The Perceptron – another variation

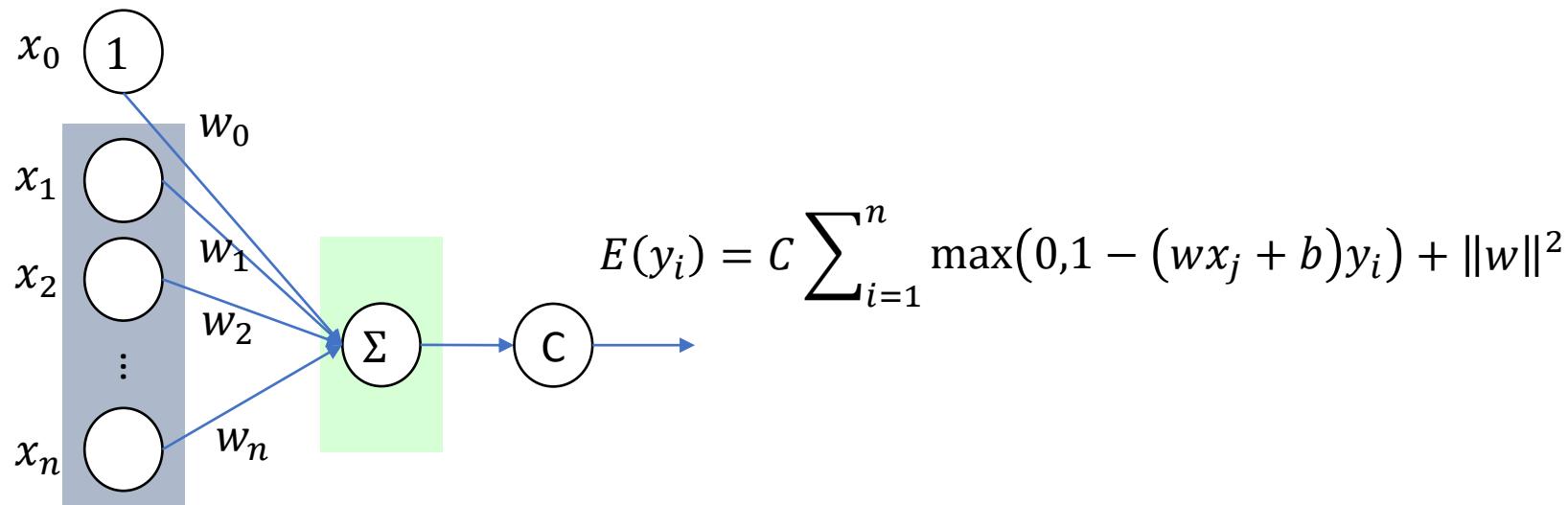
- Let's create an output for each class



- Why use this “architecture”?

Perceptron – yet another variation

- Remove sigmoid and replace error function with hinge loss (and add L_2 regularization):



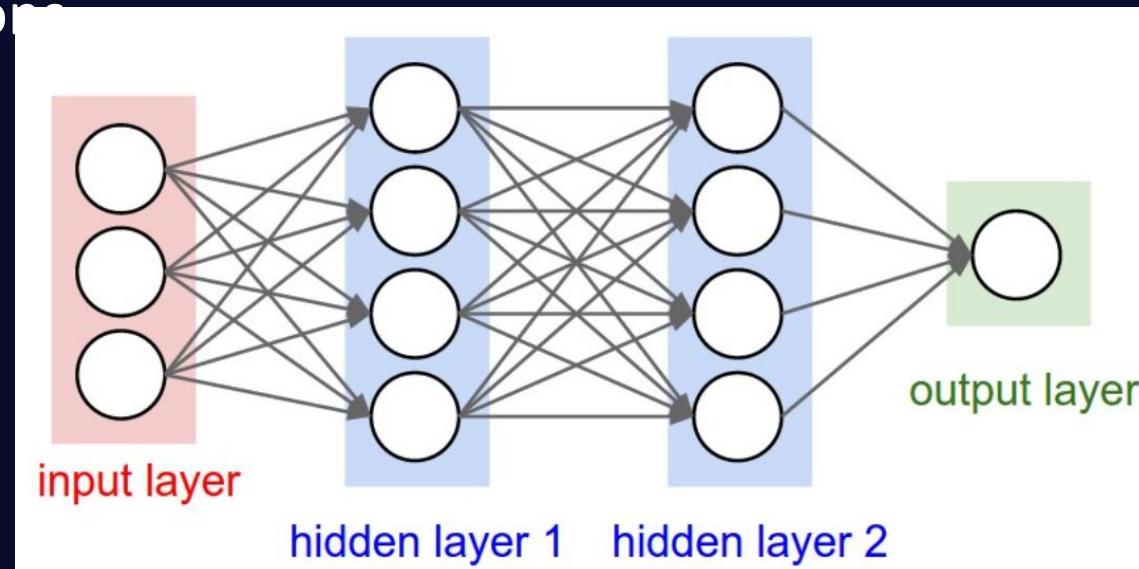
- And we get SVM!



Multi-layers perceptron

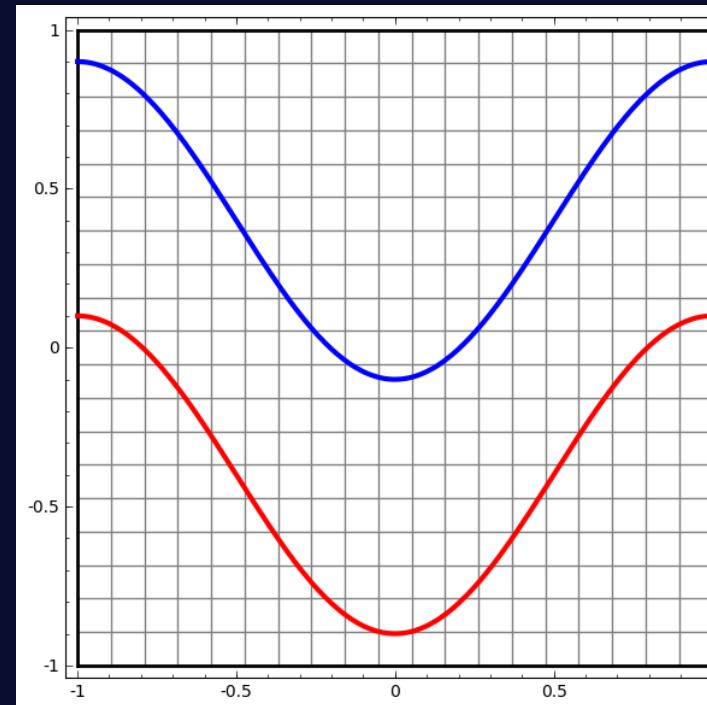
Multi-Layer Perceptron

- Using the “building blocks”, we can stack multiple layers to form a simple network
 - Networks are fed with inputs and trained to output the corresponding targets according to a loss function.
 - Hidden layers, are said to represent the data, by applying non-linear transformations



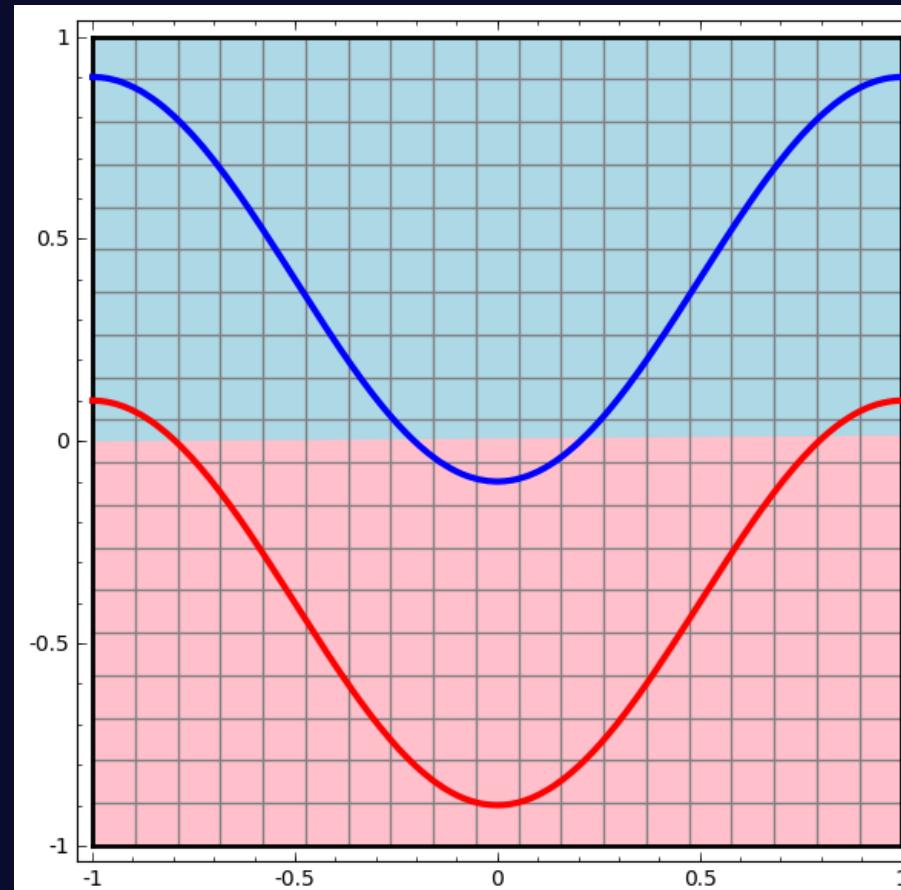
Multi-layer intuition

- The problem below is separable, but not linearly



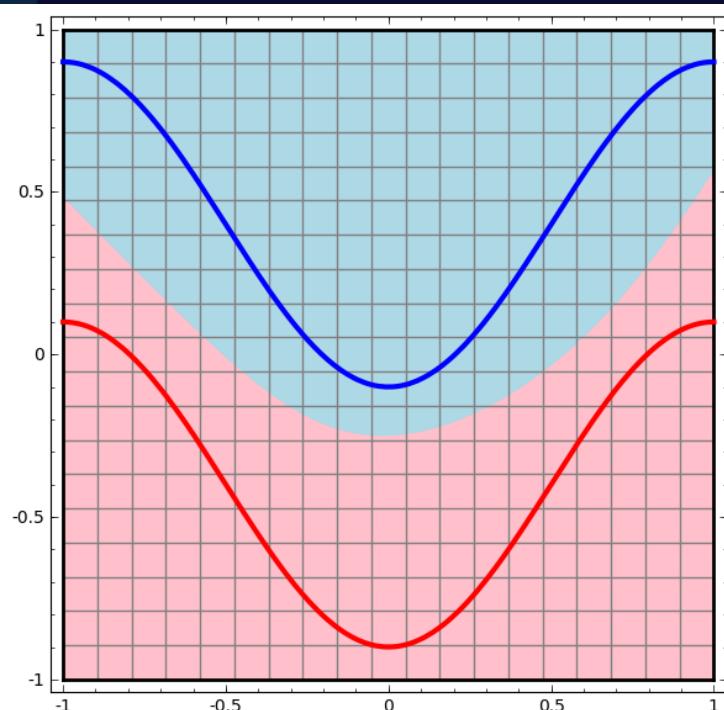
Multi-layer intuition

- The problem below is separable, but not linearly

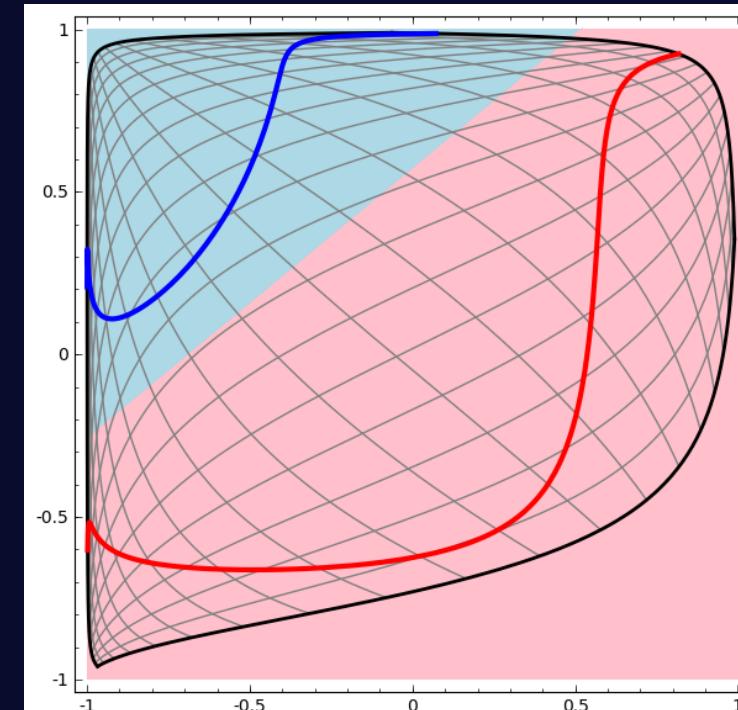


Multi-layer intuition – cont.

- The multi layers can be thought of transforming the new data to a new representation



Non-linear separator in original space



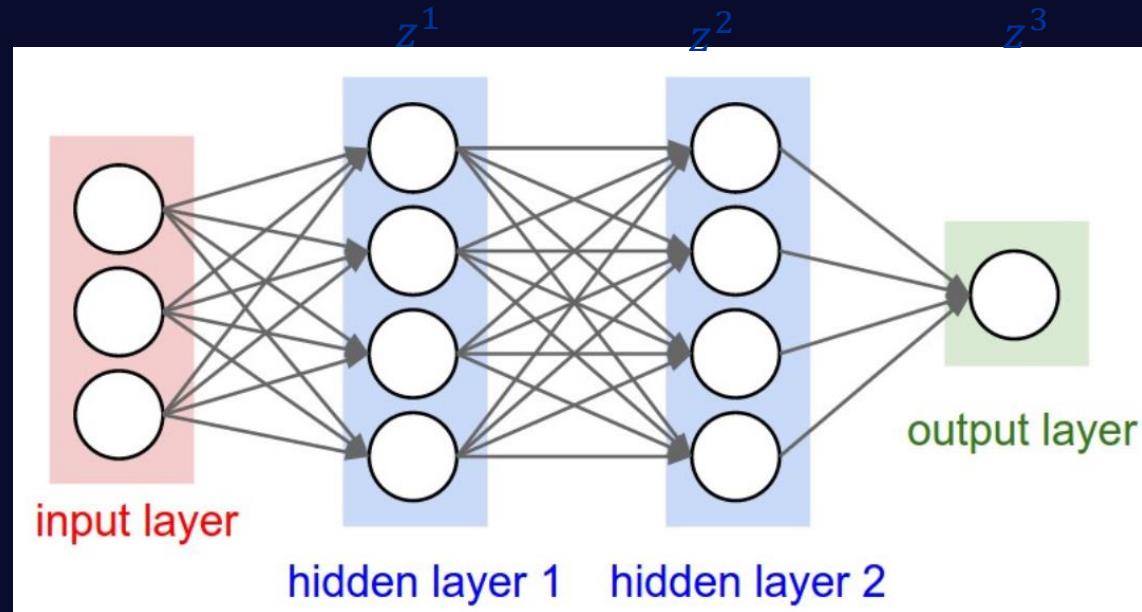
Linear separator in transformed space

Why are non-linearities important?

- Why bother with non-linear units?

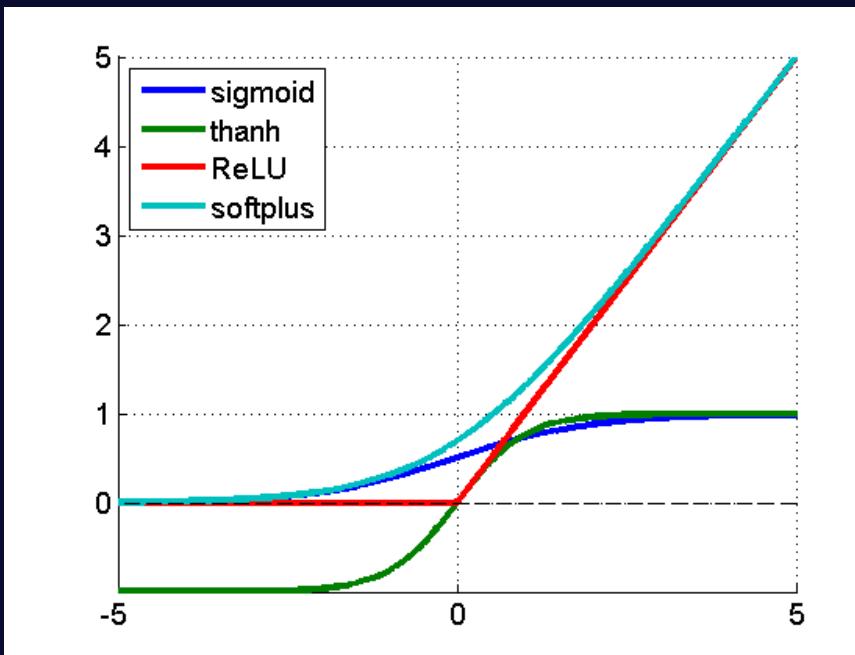
$$Z^3 = W_2^T z^2 = W_2^T W_1^T X = \tilde{W}^T X$$

The two linear layers “collapsed” to a single linear layer.



Common non-linearities

- Sigmoid - $\sigma(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic tangent - $\tanh(x) = \frac{1-e^{-x}}{1+e^x}$
- Rectified Linear Unit - $ReLU(x) = \max(0, x)$



MNIST Visualization

- <http://scs.ryerson.ca/~aharley/vis/fc/>



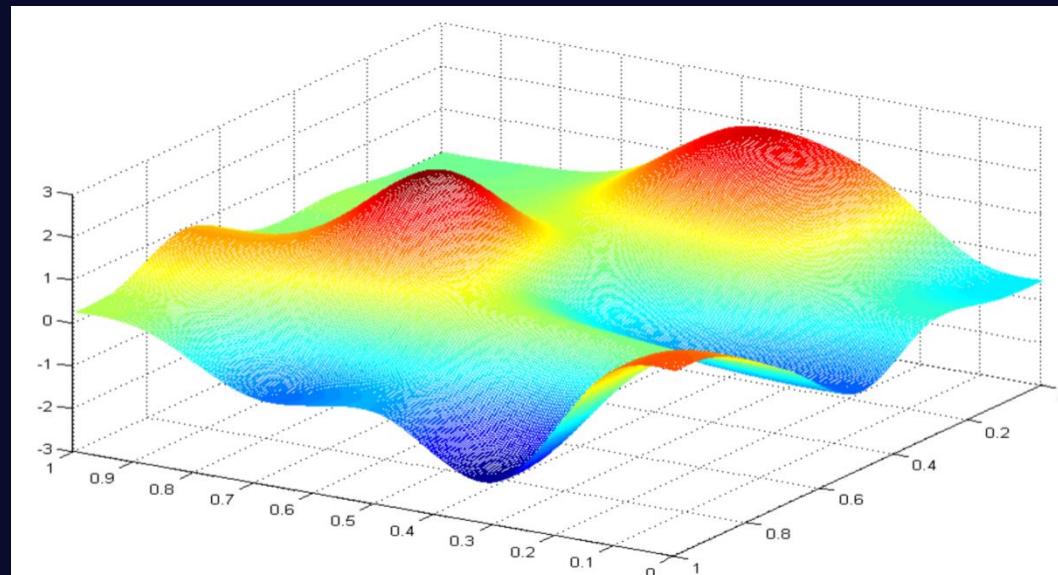
Optimization in neural networks

Stochastic Gradient Descent

- Training a neural network with a large number of parameters requires a simple optimization technique.
 - Usually, a variant of *Stochastic gradient descent (SGD)* is used, using a noisy subset estimation of the gradient (mini-batch).
 - It requires $O(n)$ number of computations and memory use, where n is the number of parameters.

Efficient optimization of NNs

- Optimizing neural networks can be very challenging, as we're dealing with a highly non-convex problem, residing in a high dimensional space.
Problems can be observed even in a very simple non-convex error landscape:



Efficient optimization of NNs – cont.

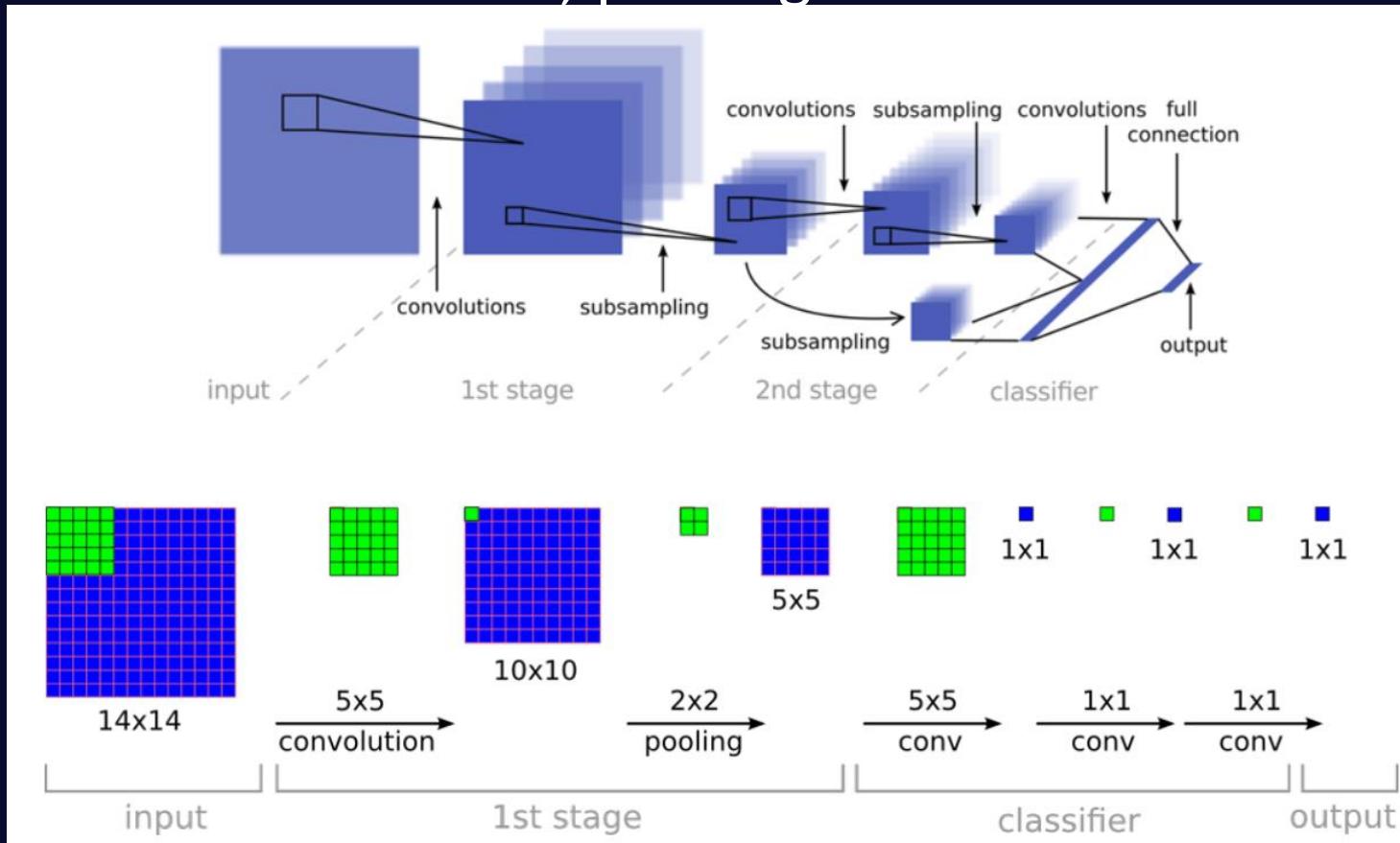
- Some obvious difficulties:
 - Strong dependency on initial weights
 - Possibly poor error estimation leading to noisy gradients
 - Exploring a (very) high-dimensional space using only local information
- Updating the network is done using backpropagation (chain-rule)



Convolutional neural networks

ConvNets

A convolutional network (ConvNet/CNN), is composed of multiple layers of convolutions, pooling and non-linearities.



Background

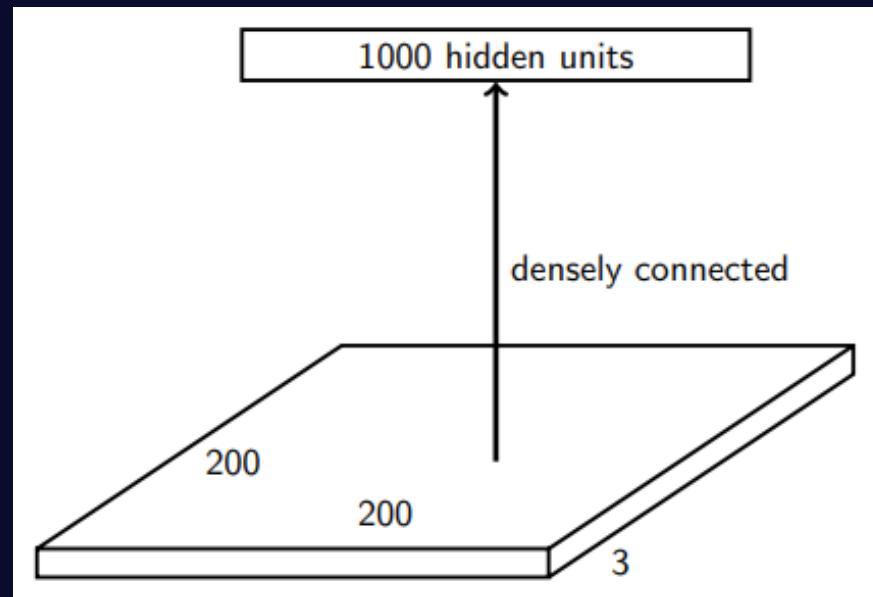
- Convolutional networks are rooted in ideas from “NeoCognitron” (Fukushima 80’) and inspired from Hubel’s and Wiesel’s work on visual primary cortex (59’)
- Their first modern form appeared in LeCun’s 98’ work to recognize handwritten digits (LeNet).
- They reached their current status as visual recognition de-facto standard, after beating traditional computer-vision techniques in 2012 ImageNet competition by huge margin (AlexNet).

Spatial Convolution

- Natural images have the property that many patches share statistical properties, and can often be described with a fairly small set of the same features
- This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations.
- This is a widely used property in computer-vision and image-processing related tasks.

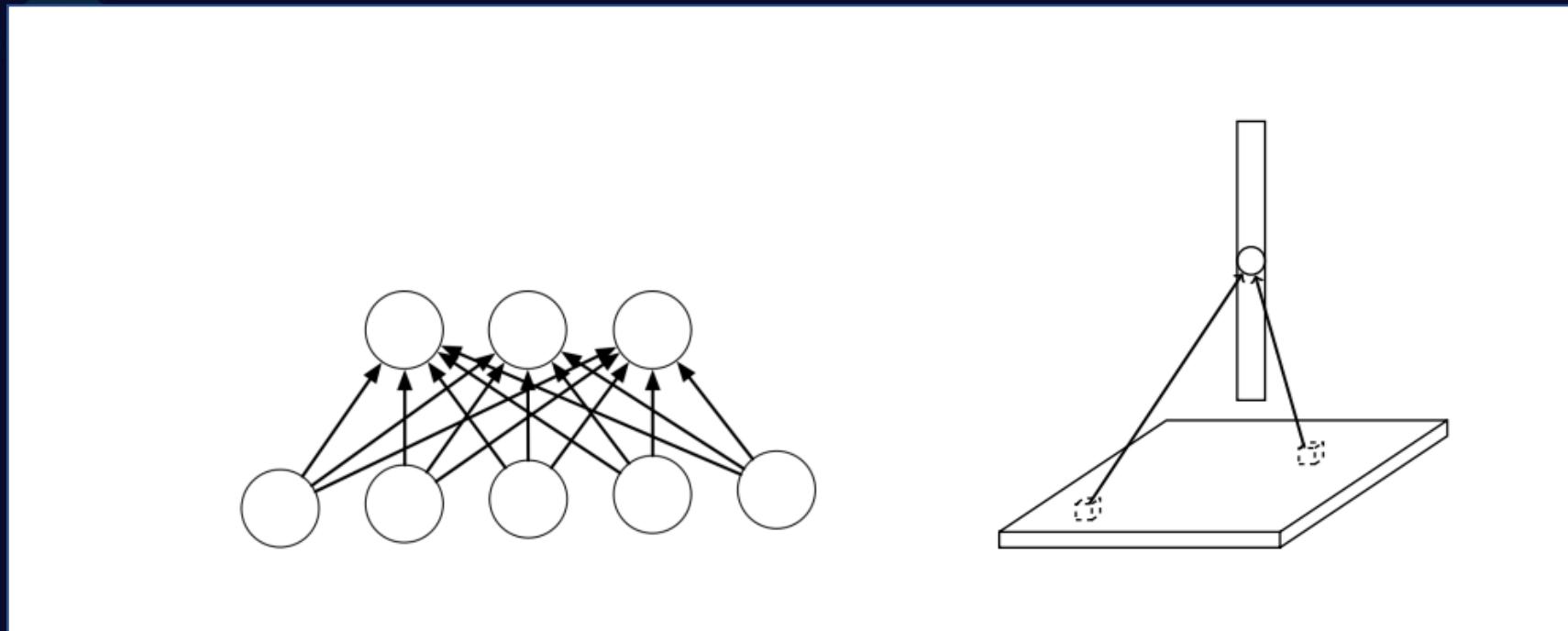
Linear layer for images

- Let take an example of a small image: 200x200x3 that is fully connected (linear connection) to 1000 units.
- This layer has $200 \times 200 \times 3 \times 1000 = 120M$ parameters!
- Furthermore, it isn't shift invariant!



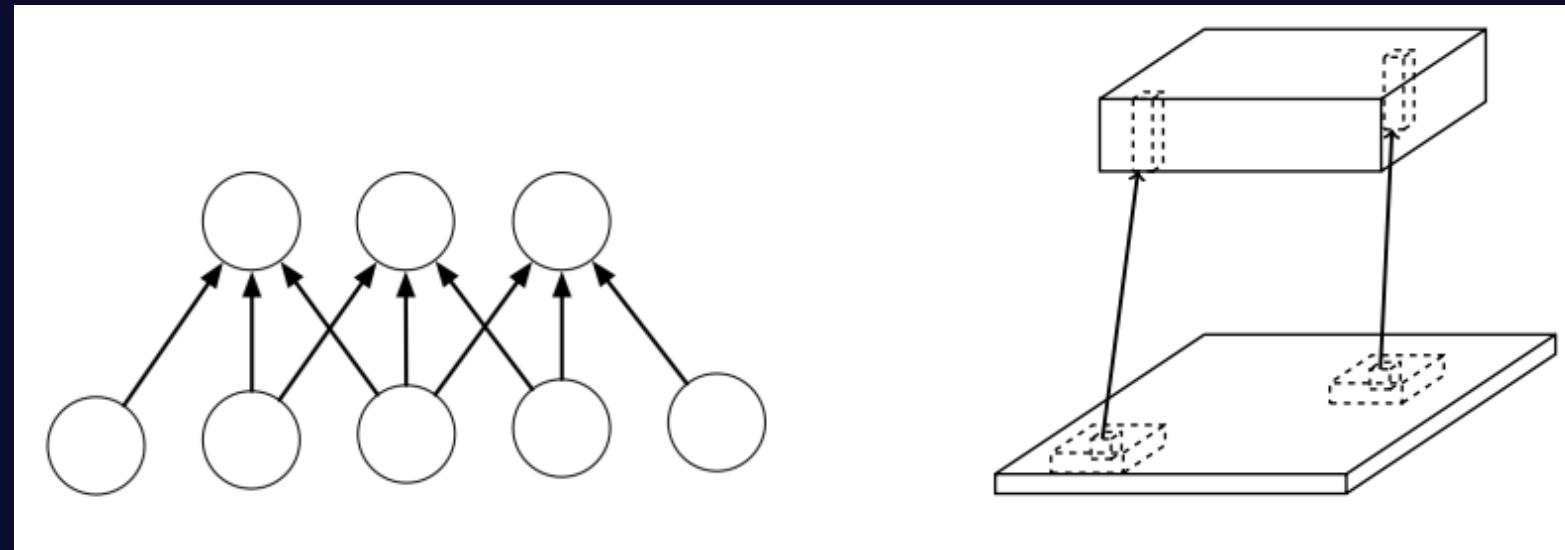
Linear layer for images

- Every “neuron” in the output is connected to every pixel in the input



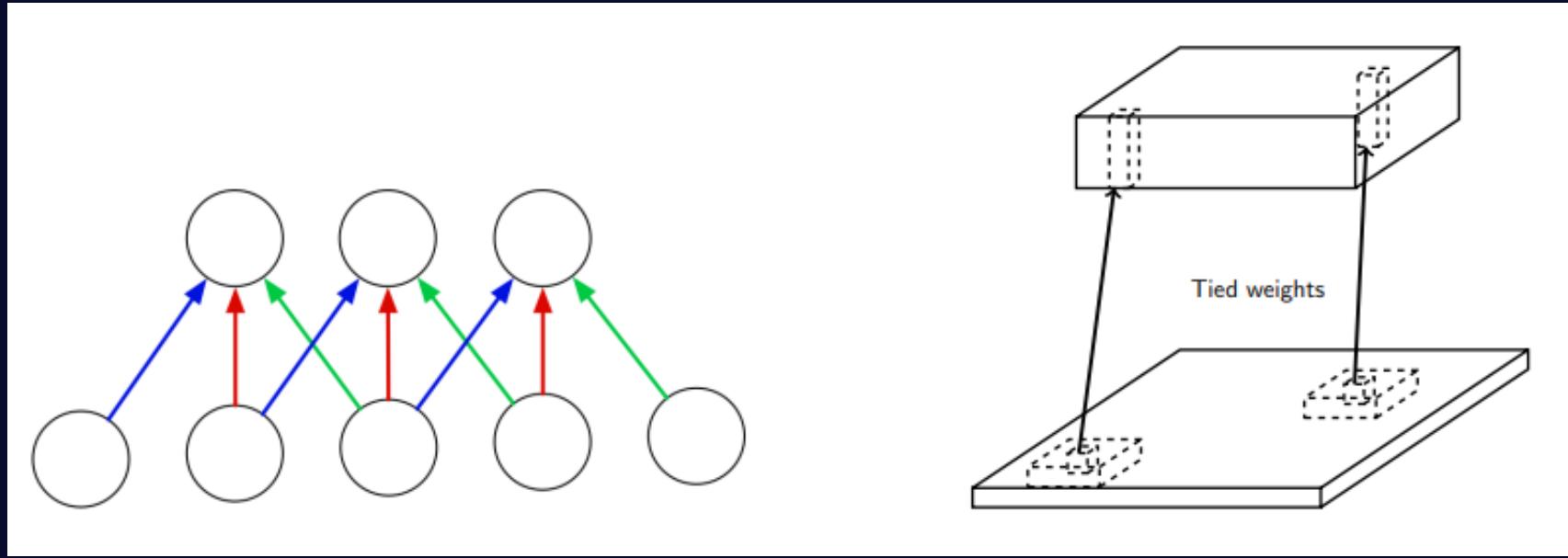
Improvement #1

- Every “neuron” is connected only to a local neighborhood of neurons (pixels) of the input



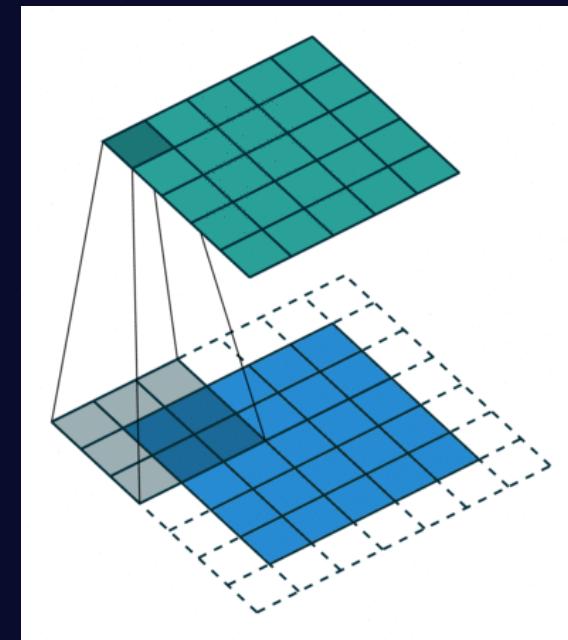
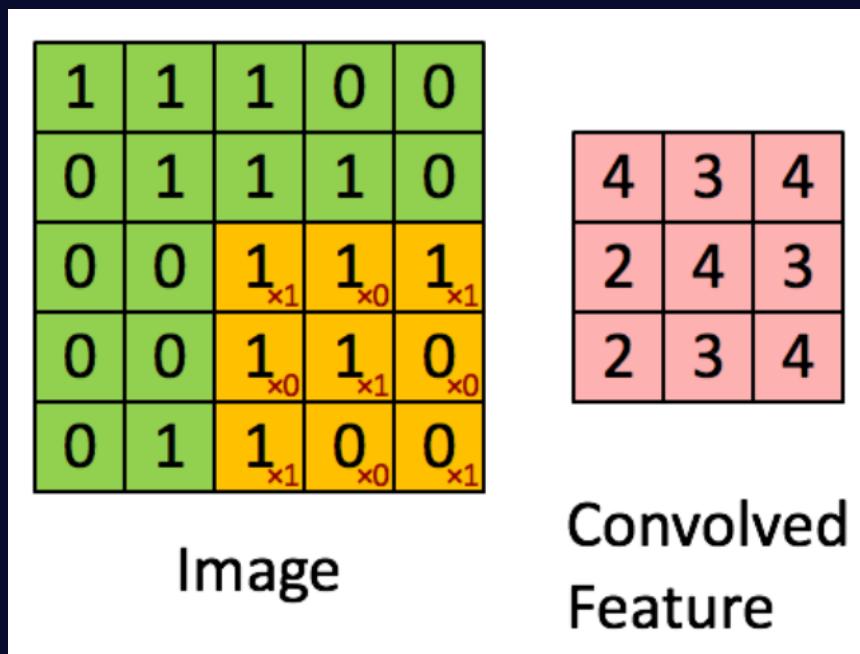
Improvement #2

- The weights connecting each output neuron to the inputs neurons are the same weights (shared)



Spatial Convolution

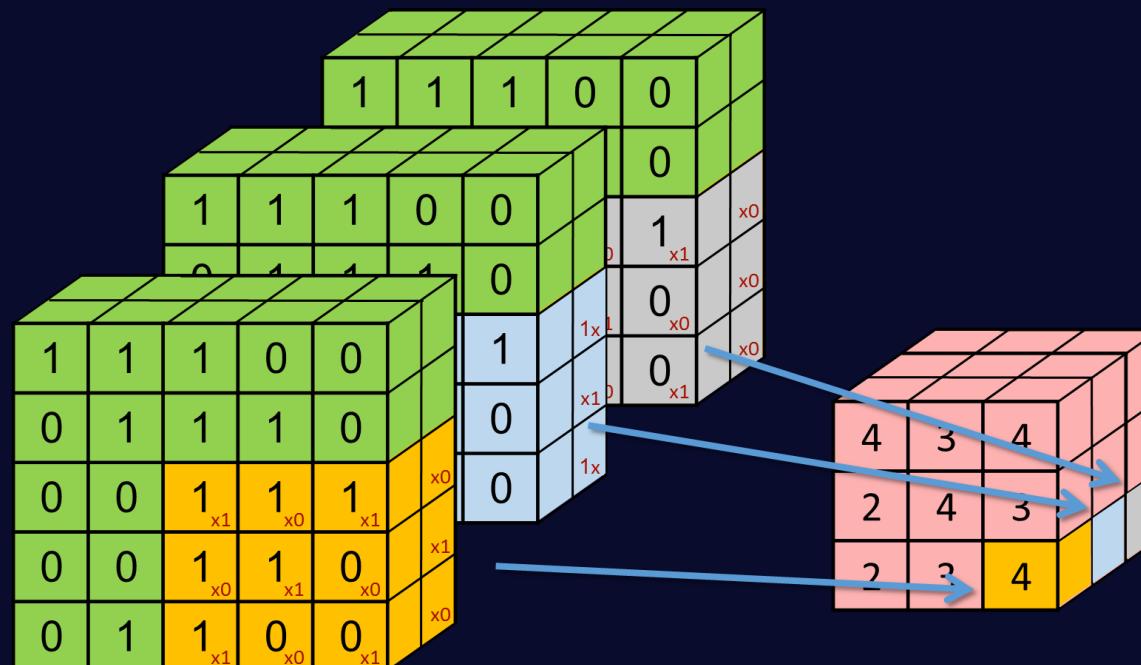
- For example, having learned features over small (say 5×5) patches sampled from the image, we can then convolve them with the larger image, thus obtaining a different feature activation value at each location in the image.



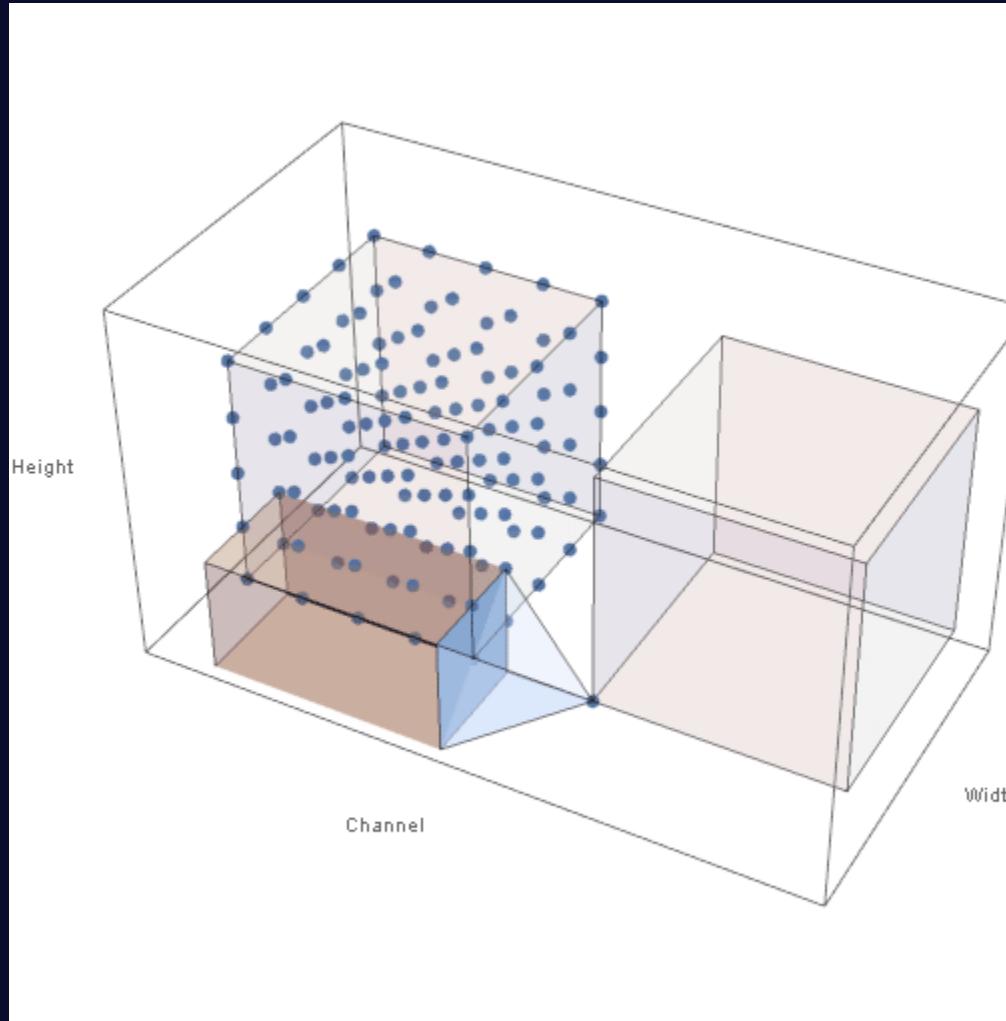
Spatial Convolution – cont.

- We perform spatial convolutions on volumes as well:
 - In the example below, we apply 3 kernels of size $3 \times 3 \times 2$
 - The output is a tensor of size $3 \times 3 \times 3$

Reduced dimension due to kernel size #kernels



2D convolution



Convention and terminology

- 2D grids that represent image space are called ***feature maps***. Each value in a feature map represent information about a specific location in the input image.
- The parameters learned for each convolution layer are also ordered as a 2D map - usually called “**kernels**” or “**filters**”
- Convolutions can have ***strides*** - moving the filters with a fixed step and ***padded*** - adding zero values to avoid spatial size decrease
- Bias values are added per-map (number of bias elements is the number of output feature maps)

Forward Function

It is most natural to think of this as correlation between two cubes:

- Our input is $X \in \mathbb{R}^{N \times W \times H}$ where N is the number of feature maps, each of spatial size $W \times H$.
- We wish to correlate them with $W \in \mathbb{R}^{M \times N \times K \times K}$ - M different filters of size $N \times K \times K$
- Output is $Y \in \mathbb{R}^{M \times (W-K+1) \times (H-K+1)}$ (no stride, no padding)

$$Y_{m,i,j} = \sum_{n=0}^{N-1} \sum_{k=0}^{K-1} \sum_{l=0}^{K-1} w_{m,k,l} \cdot x_{n,(i+k),(j+l)} + b_m$$

It can also be expressed as sum of M 2d-convolutions

$$Y_m = \sum_{n=0}^{N-1} W_m * X_n + b_n$$



Pooling layers

Pooling

- To describe a large image, one natural approach is to aggregate statistics of features at various locations. For example, one could compute the mean (or max) value of a particular feature over a region of the image.

8	1	13	8	7	13
1	5	6	8	-1	1
1	2	7	1	2	5
4	1	9	6	1	3
1	8	1	0	8	3
4	9	7	2	5	5

Max pooling
With 2x2 filters
Stride 2

8	13	13
4	9	5
9	7	8

Pooling – cont.

Pooling operations serve two main purposes:

- **Dimensionality reduction** - pooling is usually done with stride bigger than 1, effectively reducing the spatial size of the maps and allowing easier processing.
- **Local invariance to translation/transformation** - by pooling a spatial area to single point, we allow small invariance to location and deformations. This also reduces the ability to overfit on specific images (generalizes from patches).

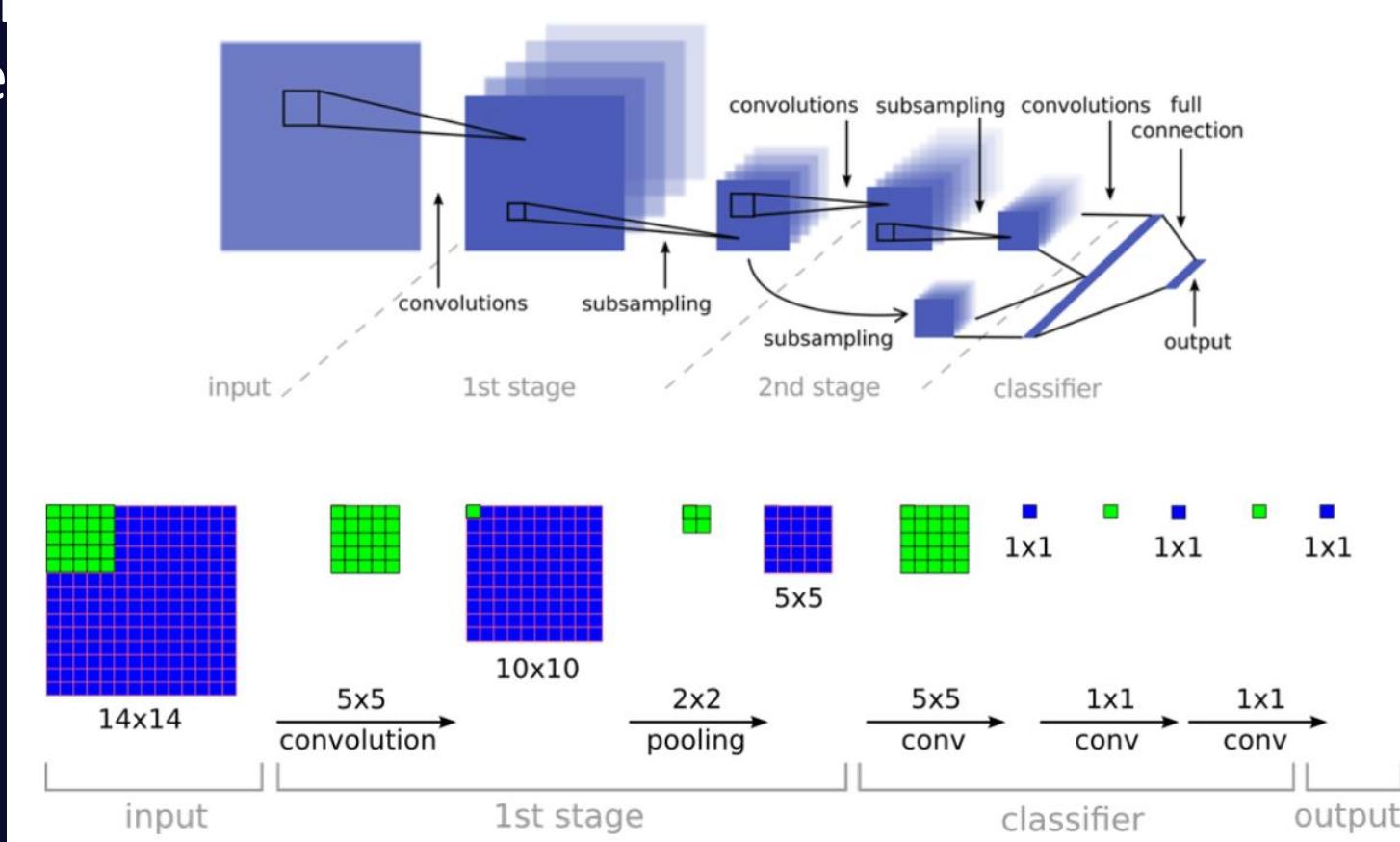
Pooling functions

Most effective pooling methods are

- Max Pooling - taking the maximum element from the pooled area.
- Average Pooling - taking the average of the pooled area.

ConvNets

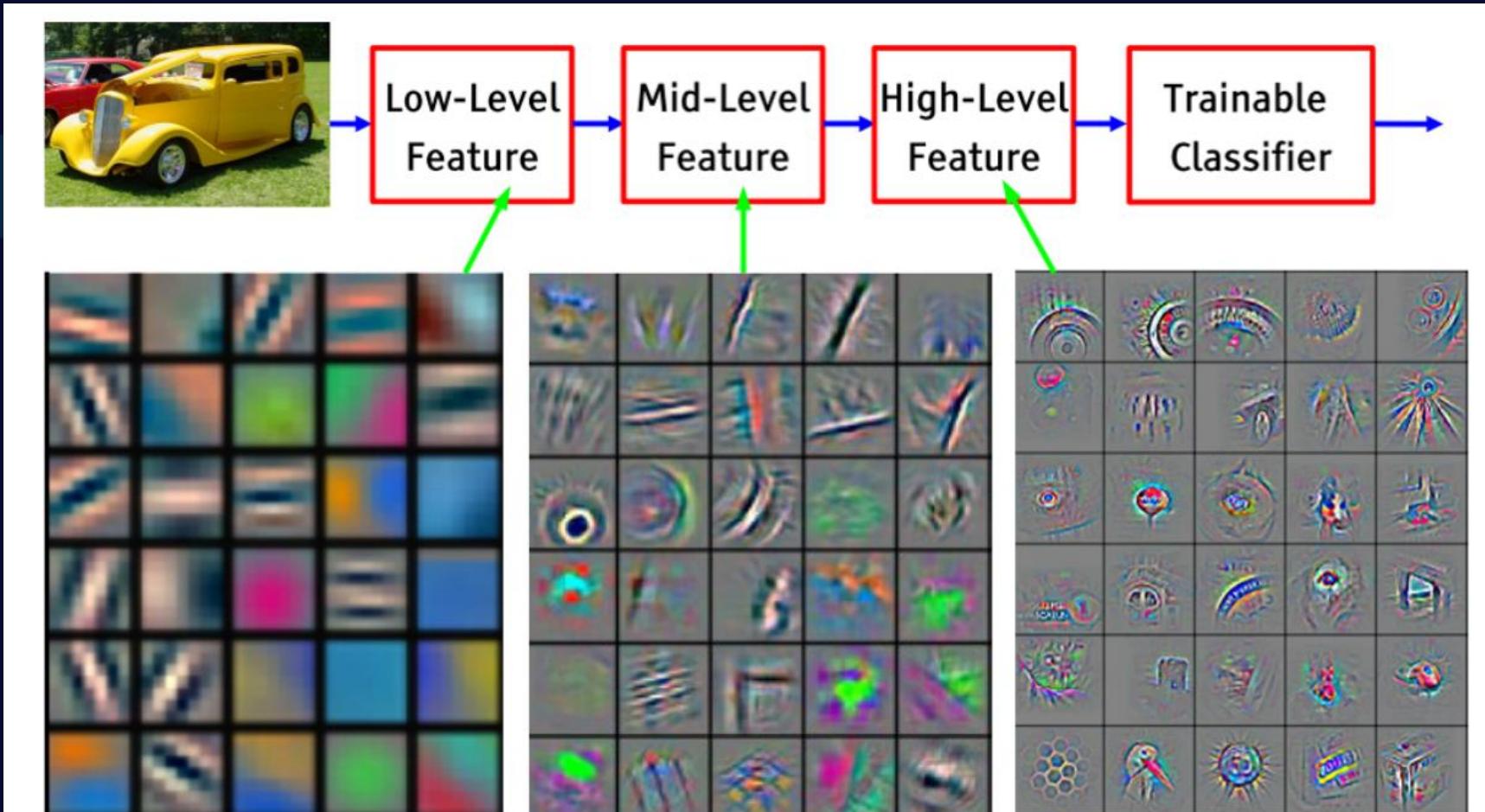
A convolutional network (ConvNet/CNN), is composed of multiple layers of convolutions, pooling and non-linearity.



MNIST Visualization

- <http://scs.ryerson.ca/~aharley/vis/conv/>

What do convolutional networks learn?

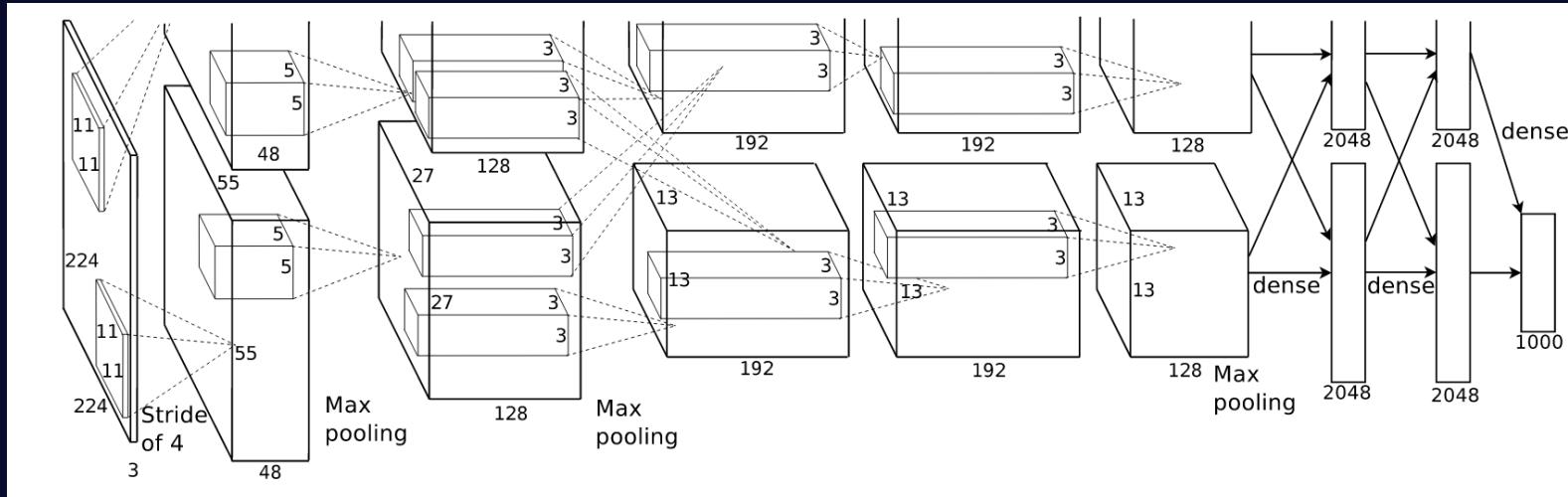


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

AlexNet

The first (recent) work that popularized Convolutional Networks in Computer Vision was the “AlexNet”, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton.

- AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% vs 26% error).
- The Network was deeper, bigger, than previous networks - 60M parameters, 8 trainable layers.

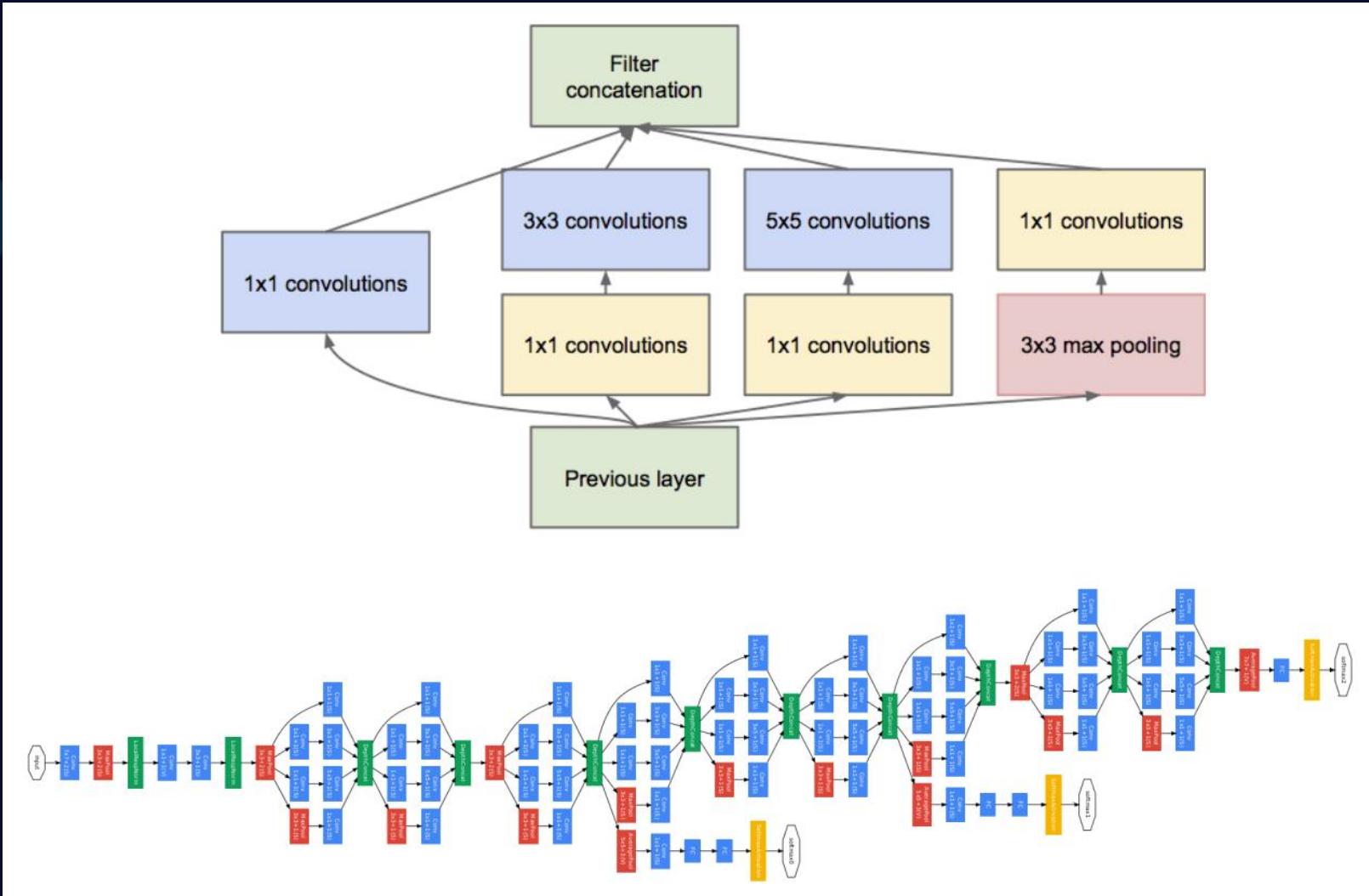


GoogLeNet

The ILSVRC 2014 winner was a Convolutional Network from Szegedy et al. from Google called *GoogLeNet*. It featured some new ideas:

- Inception Module - concatenation of several convolution sizes
- 1×1 convolutions + Average Pooling instead of Fully Connected layers (both introduced by Lin in “Network-in-network” paper from 2013).
- This dramatically reduced the number of parameters in the network (5M, compared to AlexNet with 60M).
- Google later improved this network by introducing batch-normalization + different inception configuration (Inception v2-v4)

GoogLeNet – cont.



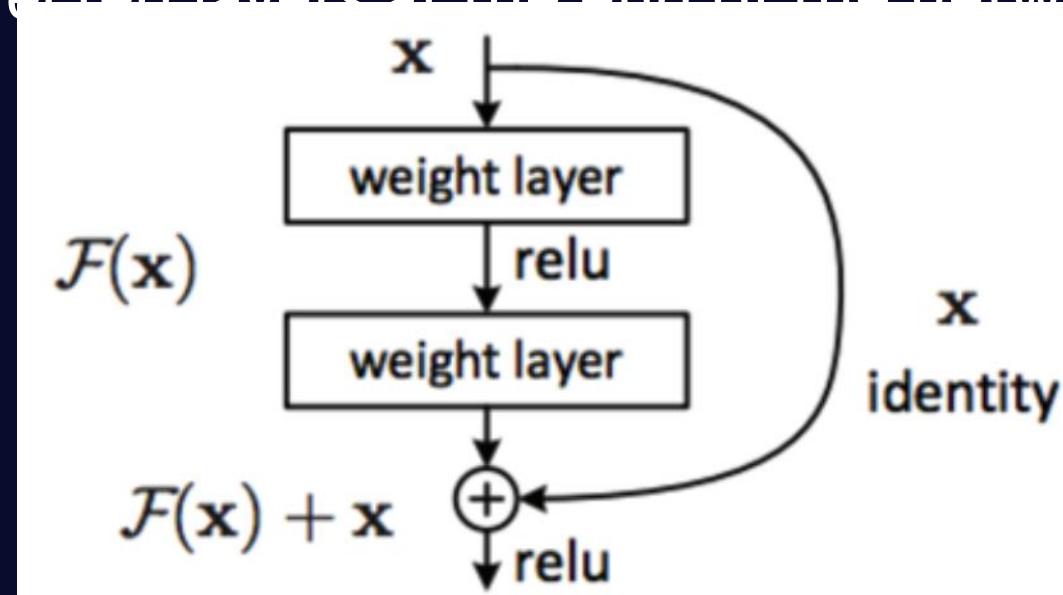
VGG Network

The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet. Its main contribution was in showing that the depth of the network is a critical component for good performance.

- Their best network contains 16 trainable layers.
- Features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end.
- Expensive to evaluate and uses a lot more memory and parameters (140M).

Residual networks

Residual Network developed by Kaiming He et al. was the winner of ILSVRC 2015. It features a very deep architecture (152 layers) with special skip connections and heavy use of batch normalization. The architecture is also missing fully connected layers at the end of the network.



Trends in designing CNNs

Some noticeable trends in recent CNN architectures:

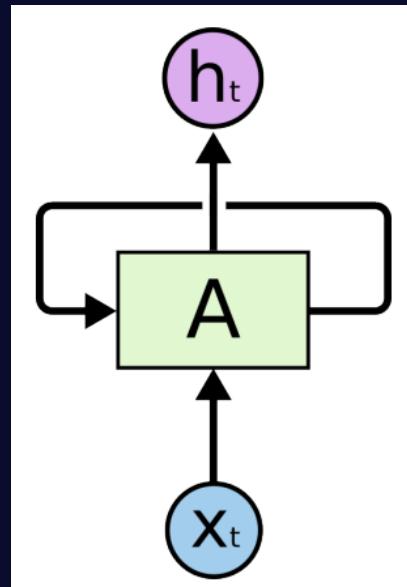
- Smaller convolution kernels - 3×3 is very dominant.
- Deeper - using smaller kernels is rectified by stacking more layers - effectively increasing the receptive field of the network.
- Less pooling - going deeper means we do not want to reduce spatial size too quickly. Modern network have small number (if any) of pooling layers.



RECURRENT NEURAL NETWORKS

Recurrent Neural Networks

- Recurrent neural networks (RNNs) are networks where connections between units form a directed cycle.



$$h_t = \phi(W_i x_t + W_r h_{t-1} + b)$$

$x_t \in \mathbb{R}^N$ - input at time t

$h_t \in \mathbb{R}^M$ - state at time t

$W_i \in \mathbb{R}^{M \times N}, W_r \in \mathbb{R}^{M \times M}, b \in \mathbb{R}^M$

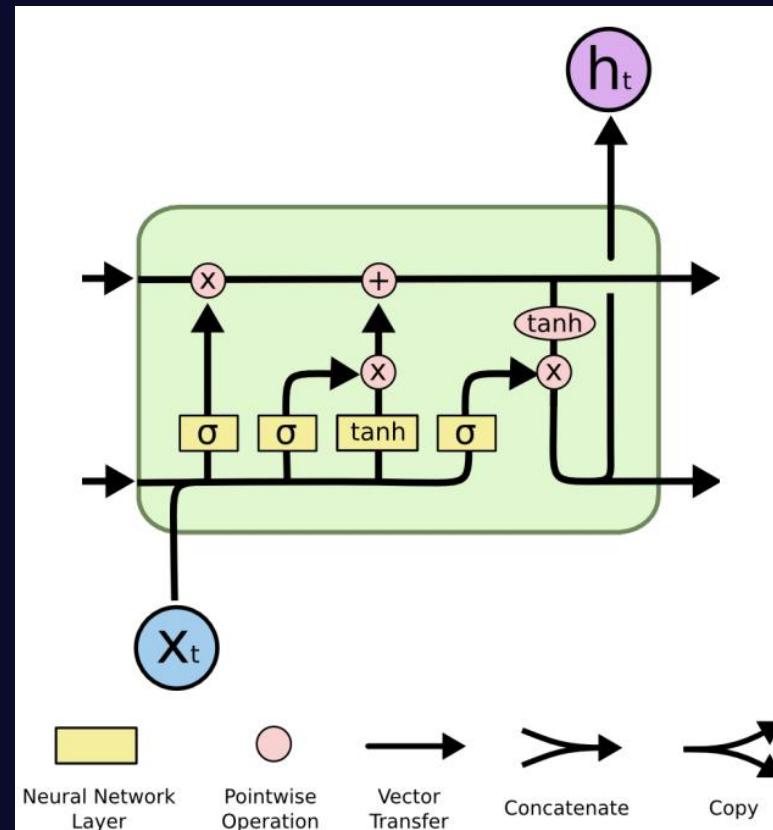
- ϕ is usually a bounded non-linearity (tanh, sigmoid)

RNN Properties

- RNN hidden states work as **memory** on previous inputs and states.
- **Temporal dependency** and internal memory allow processing of sequences of inputs
- Able to address wide range of time-dependencies
- Able to learn and infer sequences of varying length

Long- short- term memory

- Most successful recurrent model used today:
Long Short-Term Memory (LSTM) architecture (Hochreiter and Schmidhuber, '97)





Vision tasks & usages

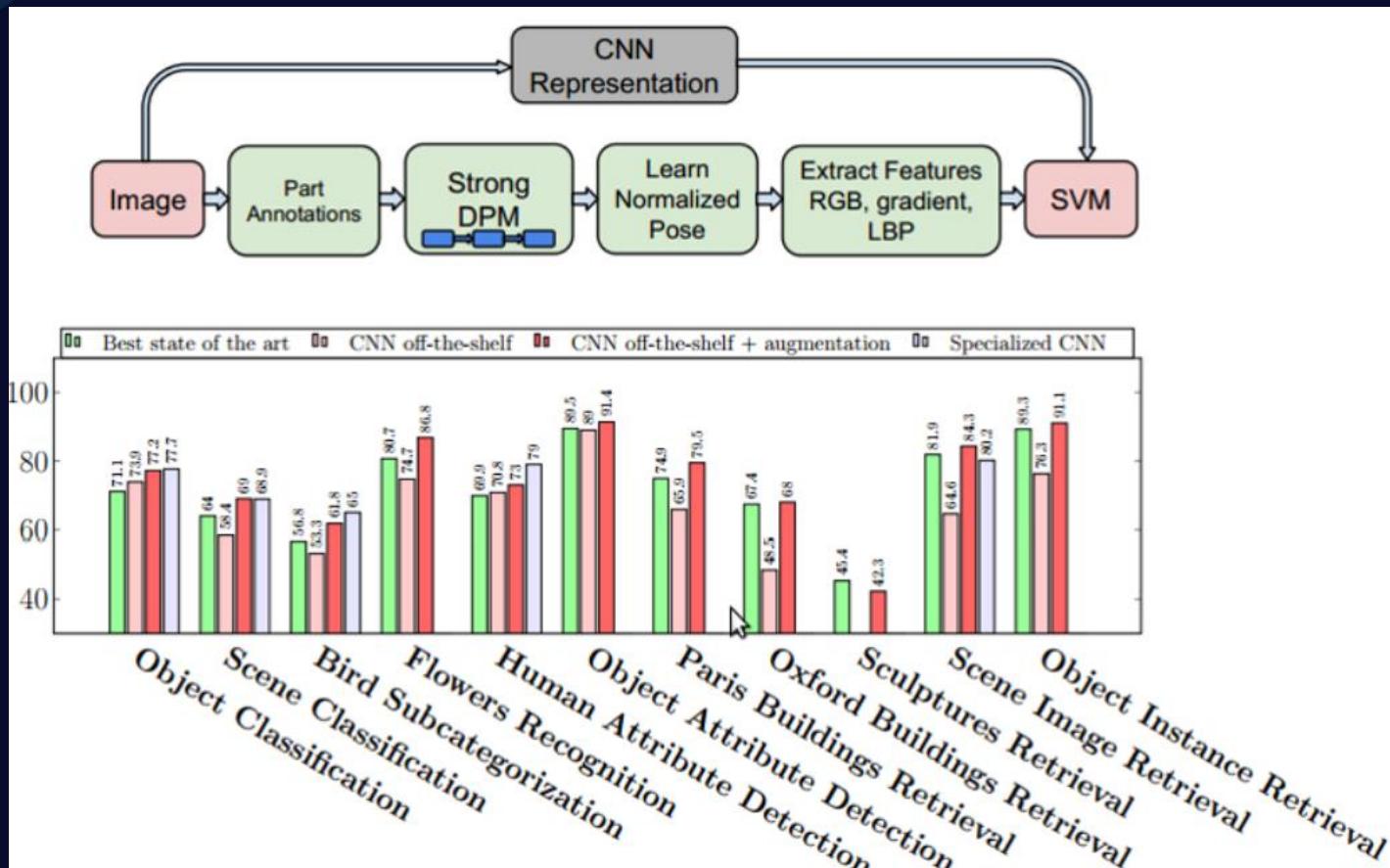
Transfer learning using deep networks

We've seen how, given enough data, we can train a convolutional network to learn complicated visual tasks.

- In practice, few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size.
- Instead, it is common to use a network that was pre-trained on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories),

Transfer learning

- This turned out to work pretty well in practice:

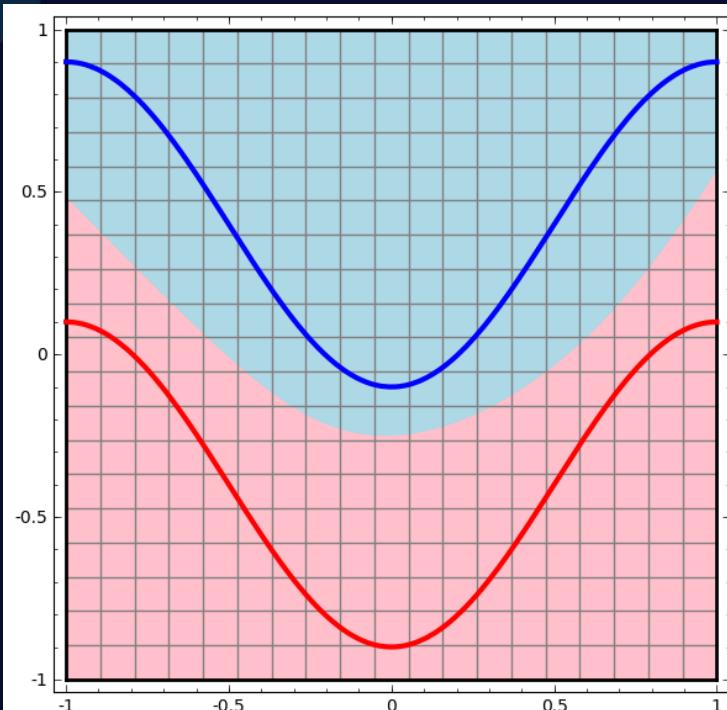


When transferring our model to the new task, our two main approaches for using the trained CNN are

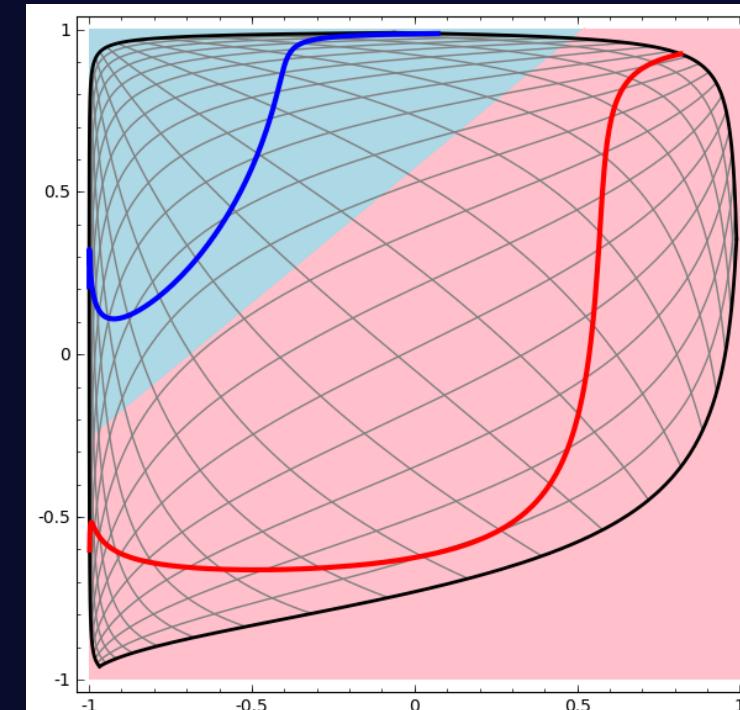
- *Generic feature extractor* - Take the pretrained network, remove the last fully-connected layer, then treat the rest of the ConvNet as a fixed feature extractor for the new dataset. You can then train a linear classifier (e.g. Linear SVM or logistic regression) on those features for the new dataset.
- *Fine-tuning* - Replace the classifier with one for the new task, then fine-tune the weights of the pretrained network by continuing the backpropagation. It is possible to fine-tune all the layers of the ConvNet, or it's possible to keep some of the layers fixed.

Multi-layer intuition – reminder

- The multi layers can be thought of transforming the new data to a new representation



Non-linear separator in original space



Linear separator in transformed space

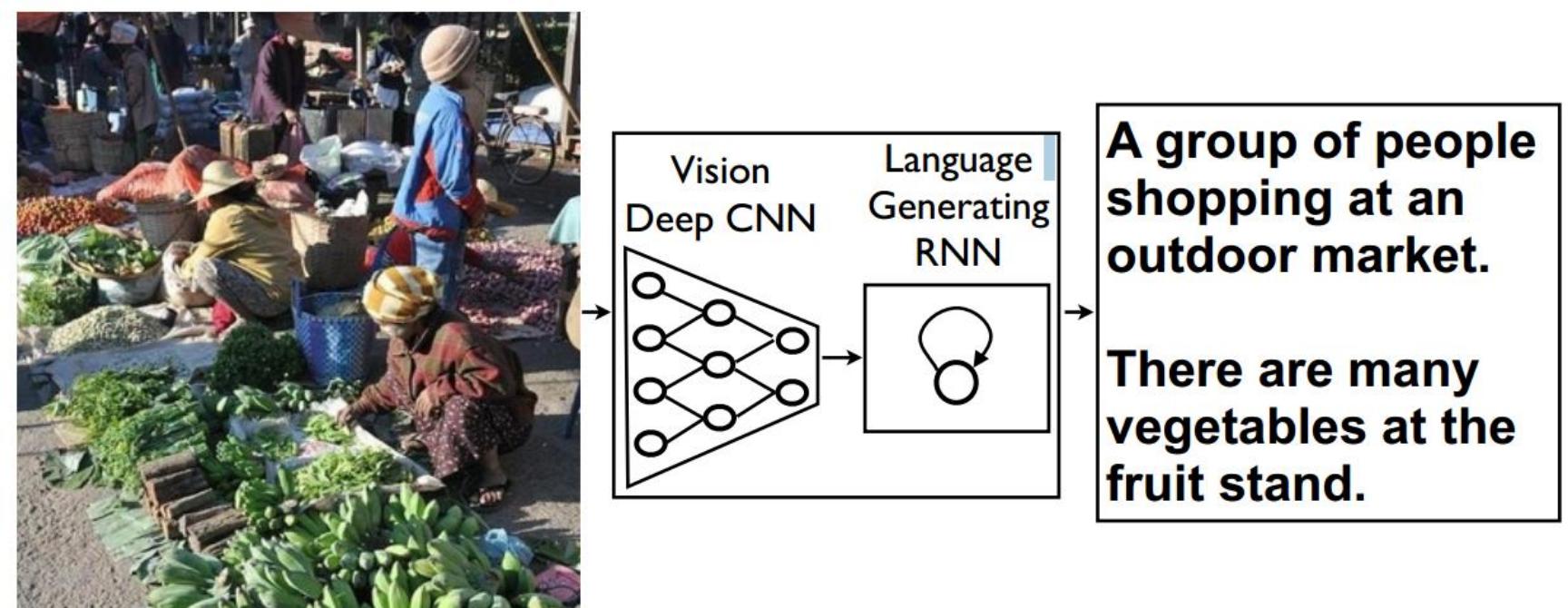
Segmentation using CNNs

- Image segmentation (*DeepMask Pinheiro 15'*)

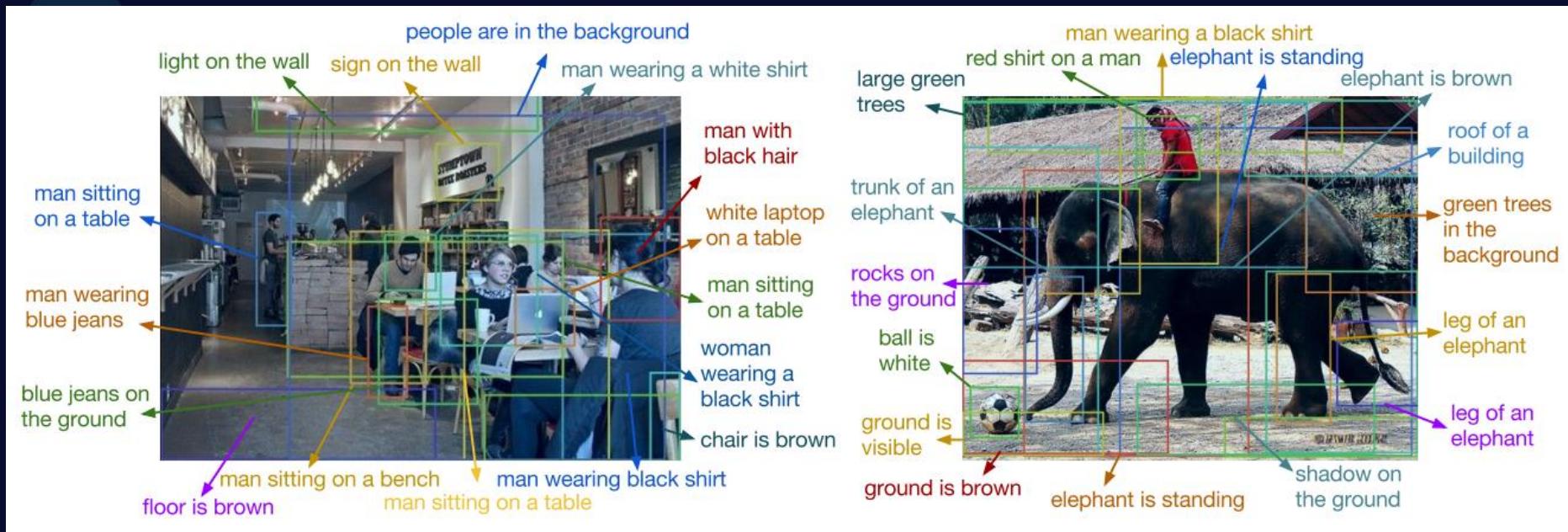


Caption Generation

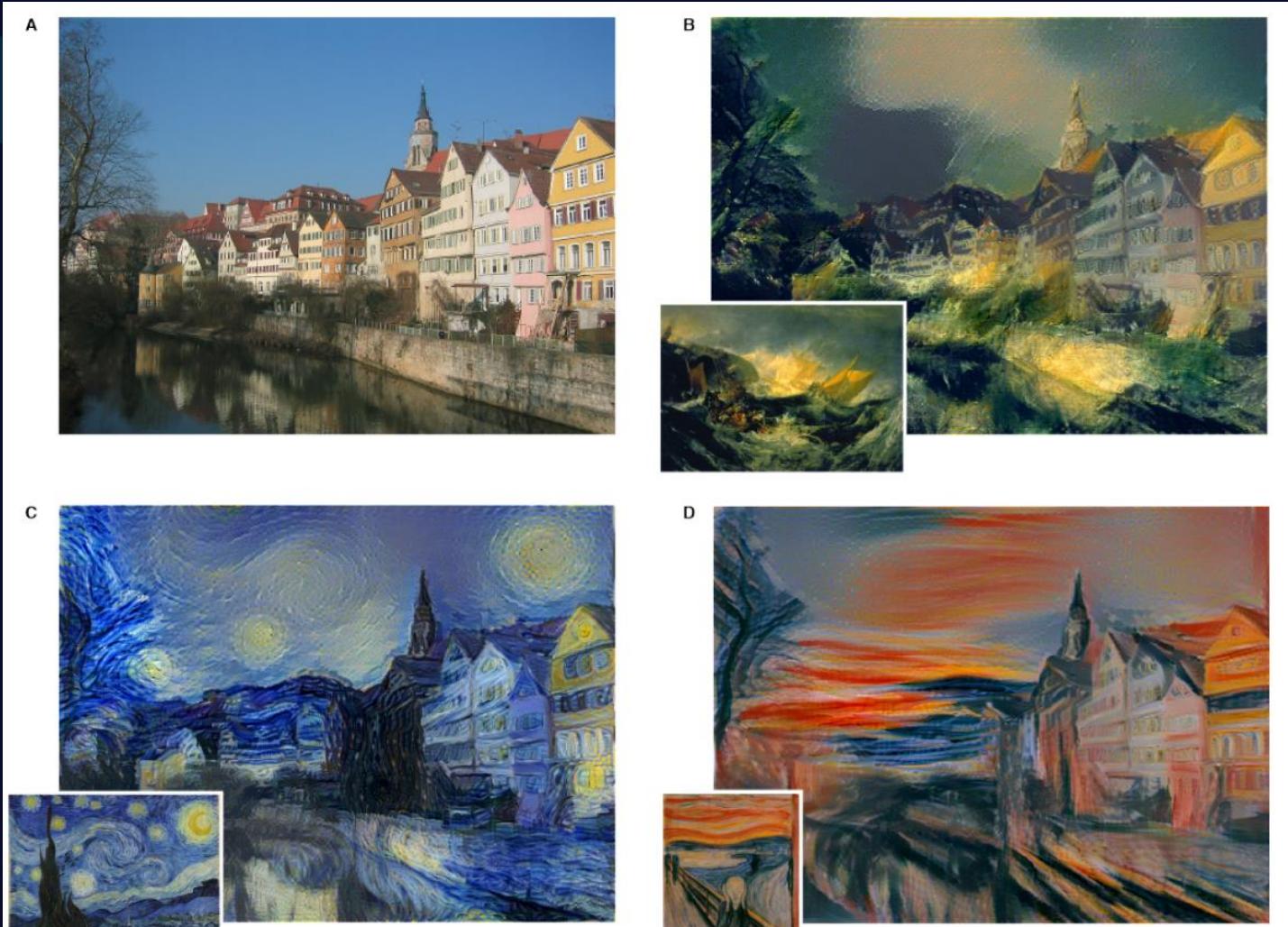
- Another usage is transferring visual knowledge into language domain
 - such as generating captions or question answering.
- This is done by augmenting a convolutional network with a recurrent one.



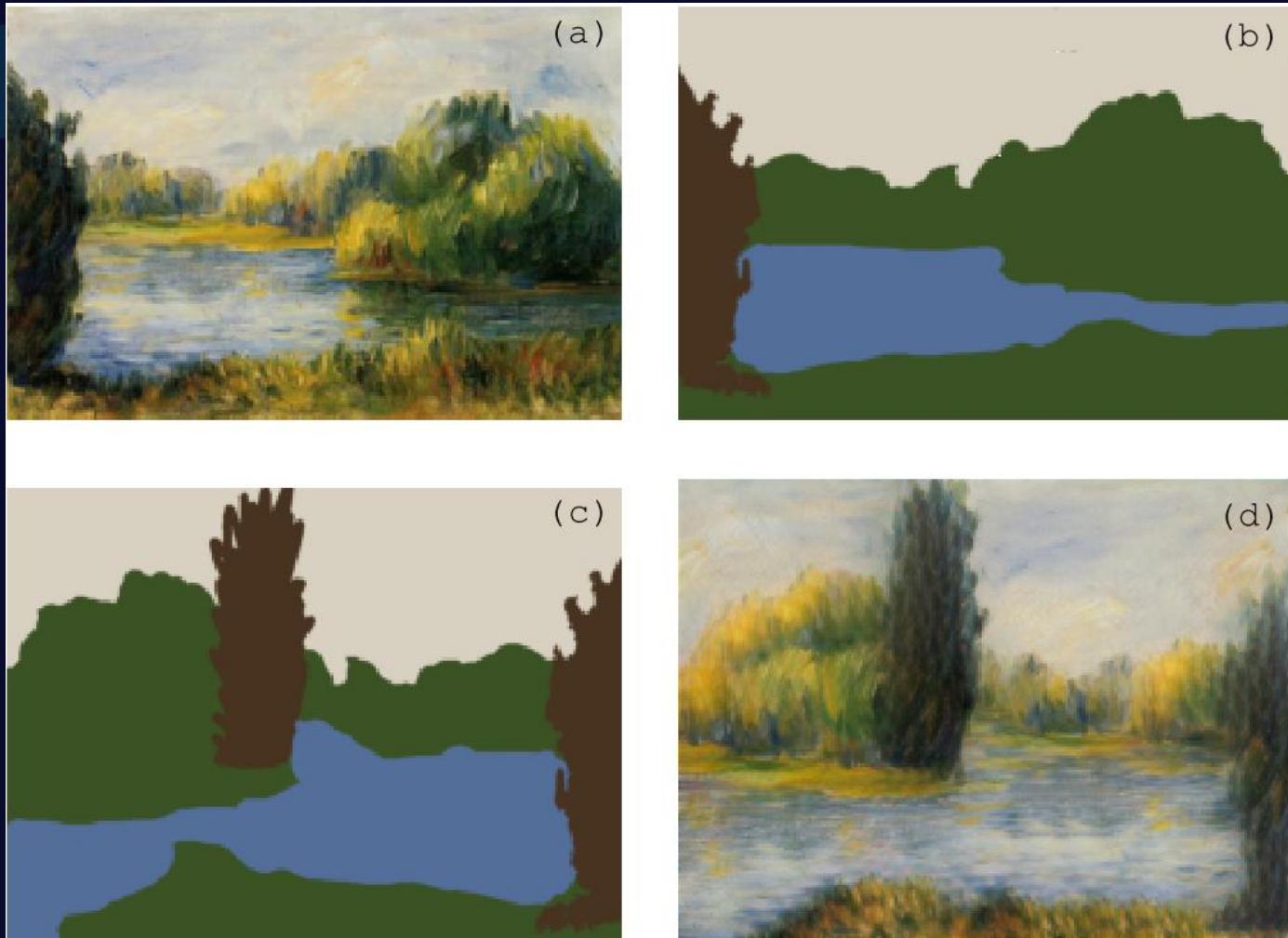
Caption Generation



Style transfer



Style transfer





Generative models

Generative Models

One new and exciting framework for unsupervised, generative models is the *Adversarial network* (Goodfellow 14').

It consists of two networks trained simultaneously:

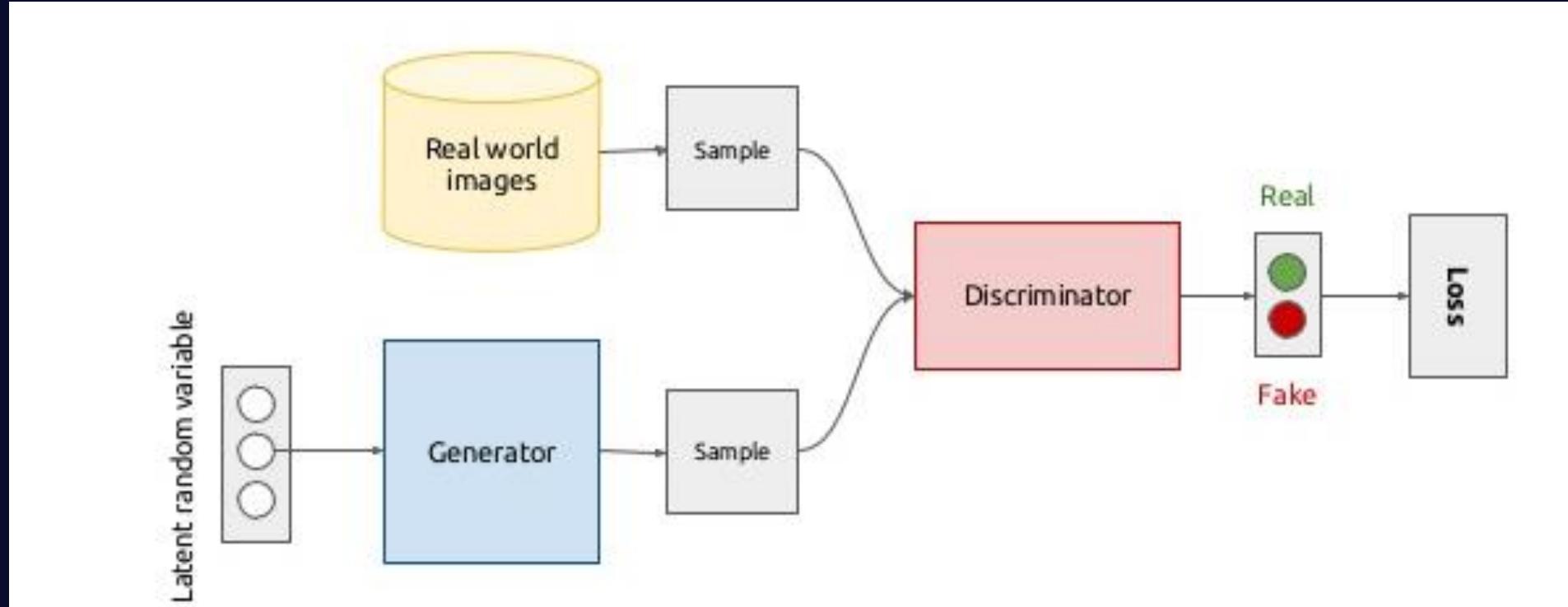
- A *generative* model G that tries to capture the data distribution and generate new samples.
- A *discriminative* model D that estimates the probability that a sample came from the training data rather than G .

Generative Adversarial Network

- The Loss function is defined as such:

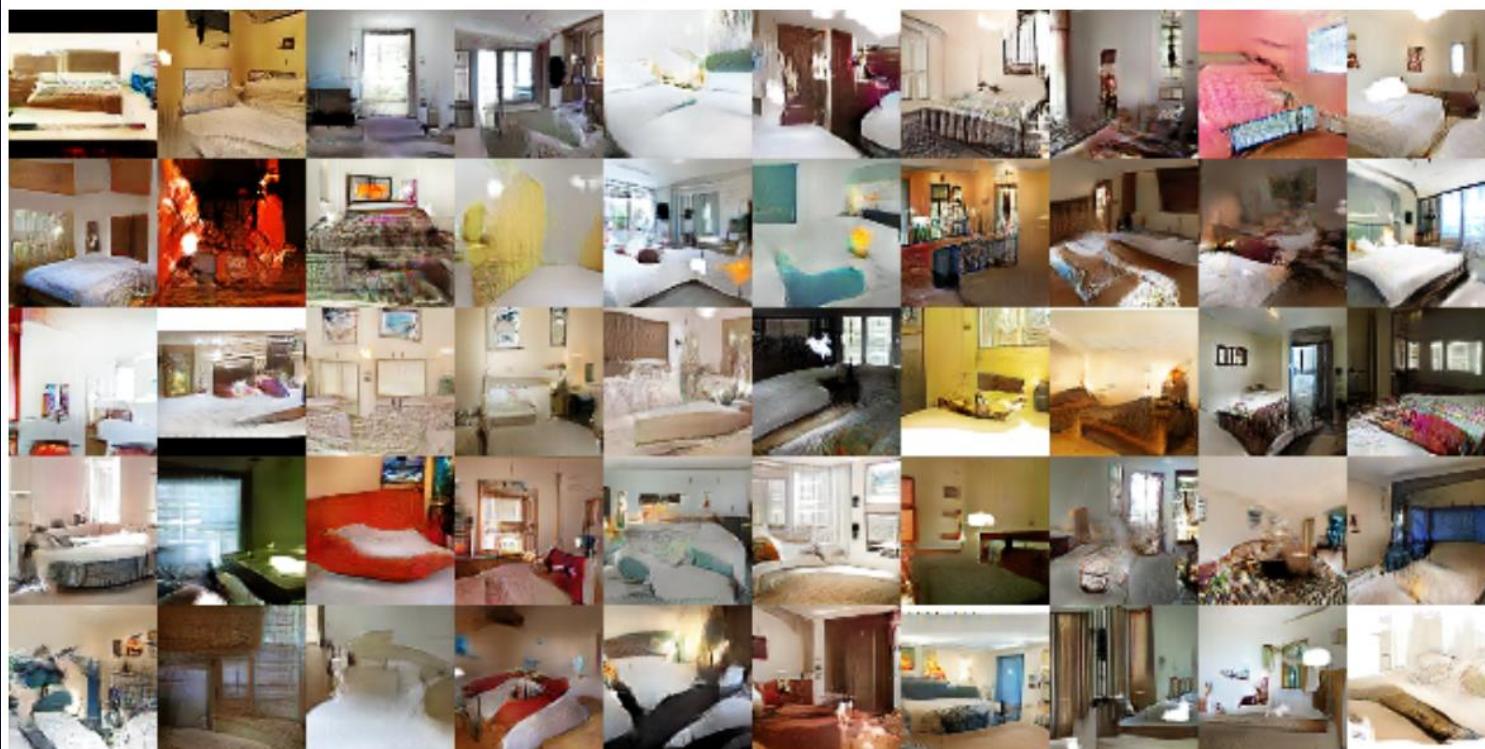
$$\min_G \max_D V(D, G)$$

$$V(D, G) = \mathbb{E}_{x \sim p_{data(x)}} [\log D(x)] + \mathbb{E}_{z \sim p_{z(z)}} [\log(1 - D(G(z)))]$$

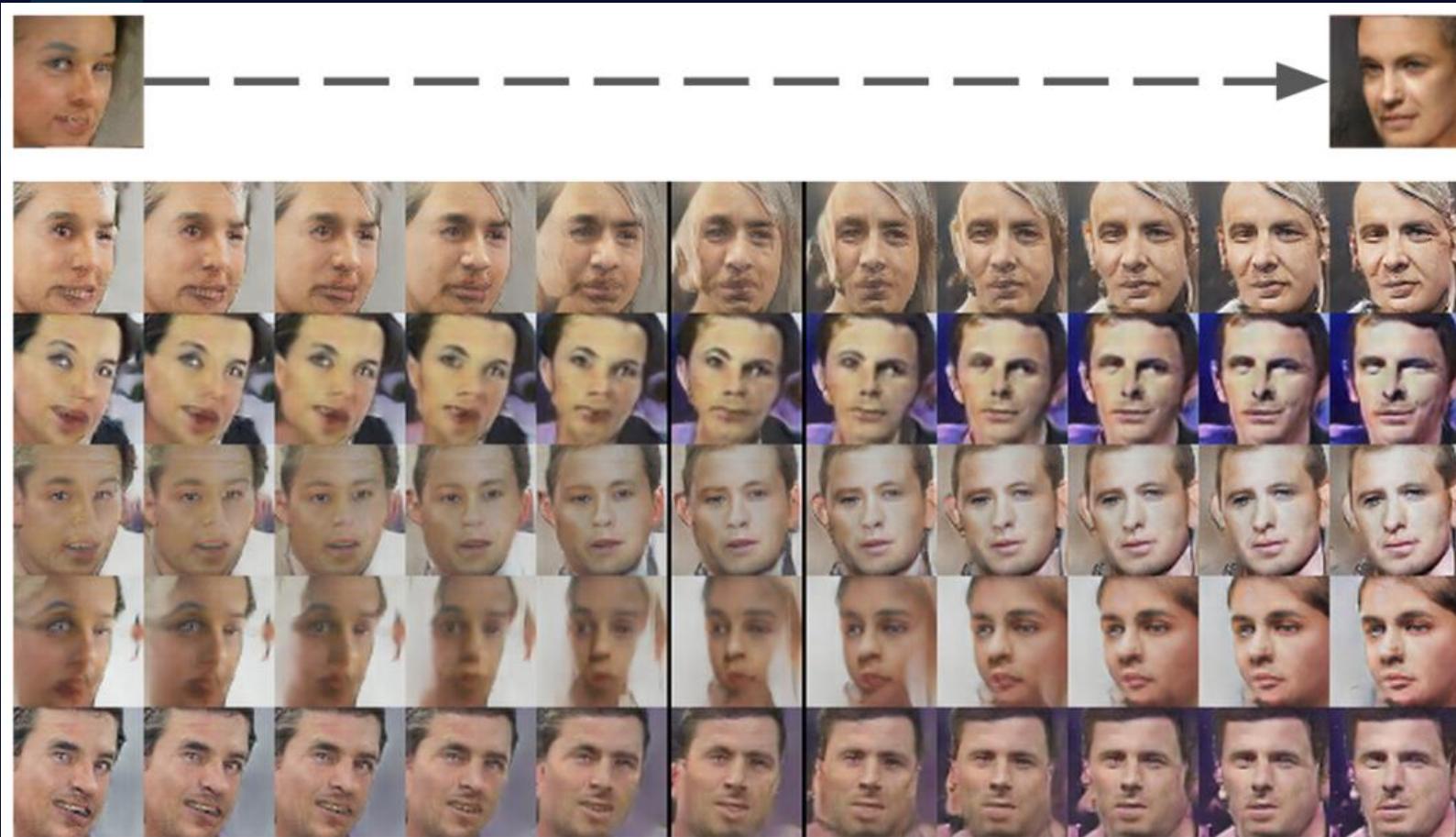


Generative adversarial networks

- “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks” (Alec Radford, Luke Metz, Soumith Chintala 15’)



Generative adversarial networks

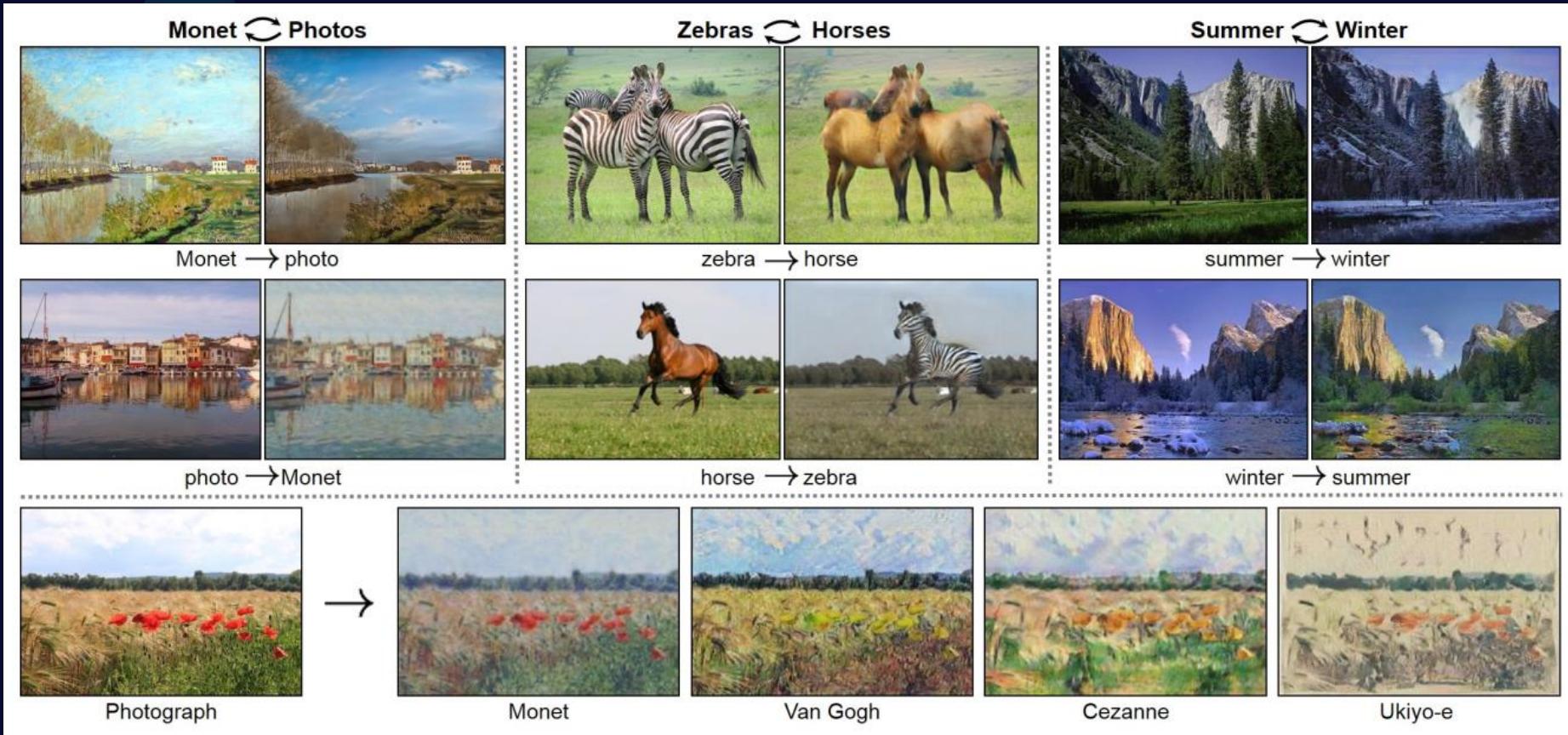


Generative adversarial networks

- Many GAN papers over the past year - with impressive improvement over time.



Cycle-GAN (Zhu et al. 17')



Deep dreaming

Another interesting usage is to try and maximize the L_2 norm of intermediate layer activations, by optimizing an input image. This can help visualize the role each layer play:

