CS4442/9542b
Artificial Intelligence II
Prof. Olga Veksler

Lecture 2

Introduction to ML

Basic Linear Algebra

Matlab

Some slides on Linear Algebra are from Patrick Nichols

Outline

- Introduction to Machine Learning
- Basic Linear Algebra
- Matlab Intro

Intro: What is Machine Learning?

- How to write a computer program that automatically improves its performance through experience
- Machine learning is useful when it is too difficult to come up with a program to perform a desired task
- Make computer to learn by showing examples (most frequently with correct answers)
  - “supervised” learning or learning with a teacher
- In practice: computer program (or function) which has a tunable parameters, tune parameters until the desirable behavior on the examples

Different Types of Learning

- **Learning from examples**
  - **Supervised Learning**: given training examples of inputs and corresponding outputs, produce the correct outputs for new inputs
    - study in this course
  - **Unsupervised Learning**: given only inputs as training, find structure in the world: e.g. discover clusters
- Other types, such as **reinforcement learning** are not covered in this course

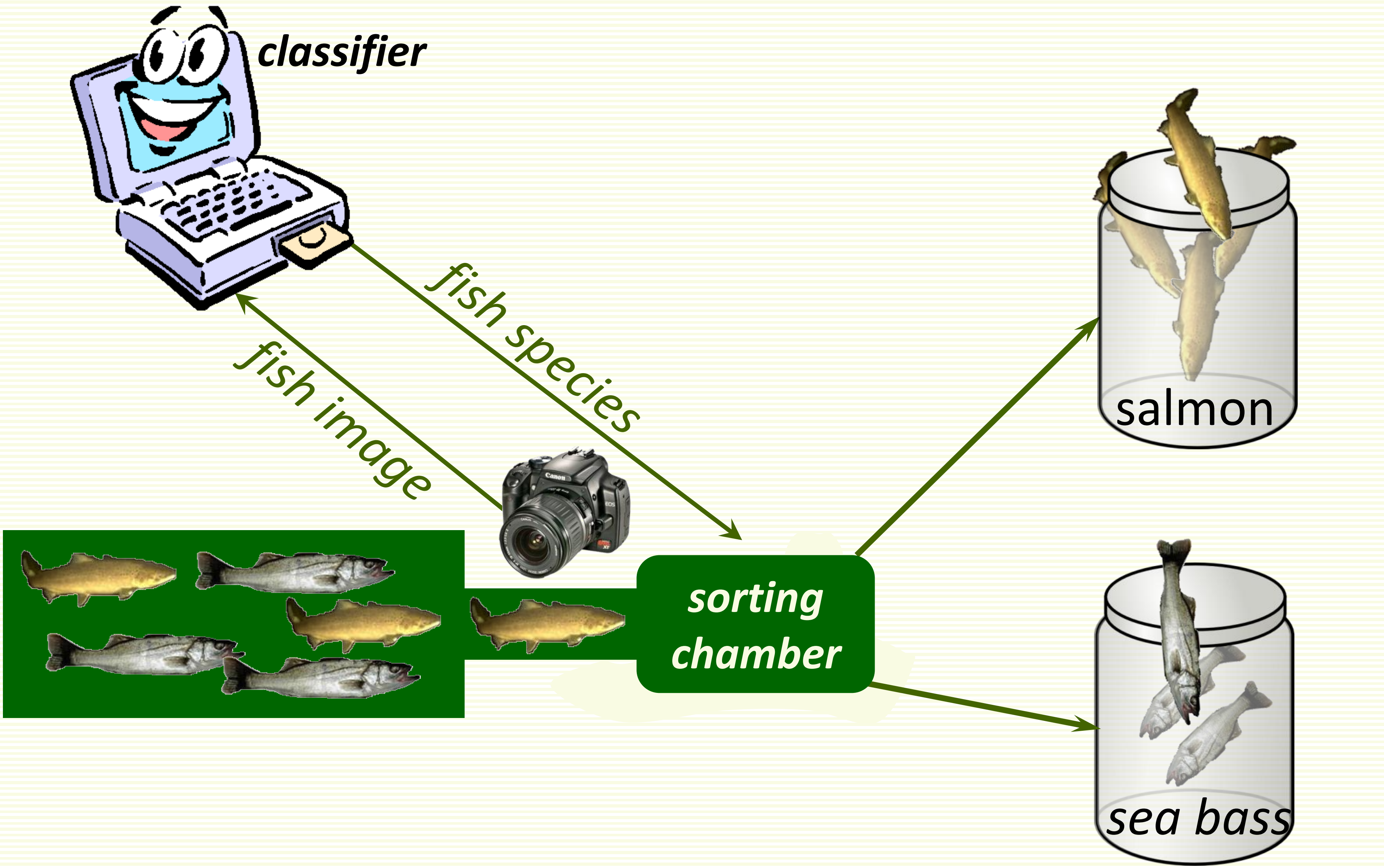
Supervised Machine Learning

- Training samples (or examples) $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^n$
- Each example $\mathbf{x}^i$ is typically multi-dimensional
  - $\mathbf{x}^i_1, \mathbf{x}^i_2, \ldots, \mathbf{x}^i_d$ are called *features*, $\mathbf{x}^i$ is often called a *feature vector*
  - Example: $\mathbf{x}^1 = [3, 7, 35]$, $\mathbf{x}^2 = [5, 9, 47]$, …
  - how many and which features do we take?
- Know desired output for each example $\mathbf{y}^1, \mathbf{y}^2, \ldots \mathbf{y}^n$
  - This learning is supervised ("teacher" gives desired outputs)
  - $\mathbf{y}^i$ are often one-dimensional
  - Example: $\mathbf{y}^1 = 1$ ("face"), $\mathbf{y}^2 = 0$ ("not a face")

Two Types of Supervised Machine Learning

- Classification
  - $\mathbf{y}^i$ takes value in finite set, called a *label* or a *class*
  - Example: $\mathbf{y}^i \in$ {“sunny”, ”cloudy”, ”raining”}
- Regression
  - $\mathbf{y}^i$ continuous, called an *output value*
  - Example: $\mathbf{y}^i$ = temperature $\in [-60,60]$

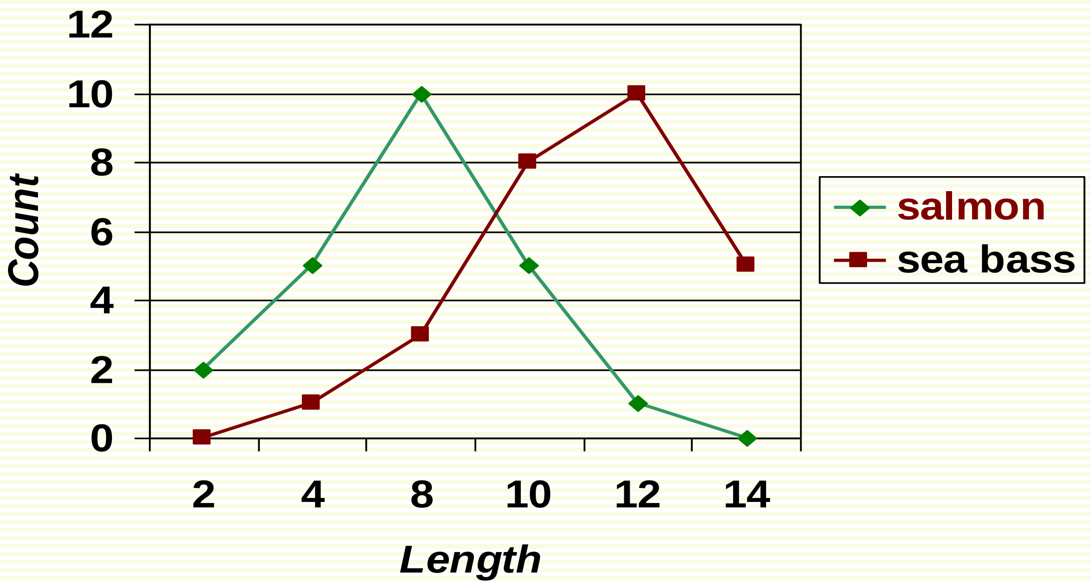
Toy Application: fish sorting



Classifier design

- Notice salmon tends to be shorter than sea bass
- Use *fish length* as the discriminating feature
- Count number of bass and salmon of each length

	2	4	8	10	12	14
bass	0	1	3	8	10	5
salmon	2	5	10	5	1	0



Single Feature (length) Classifier

- Find the best length ***L*** threshold

fish length < L → ***classify as salmon***

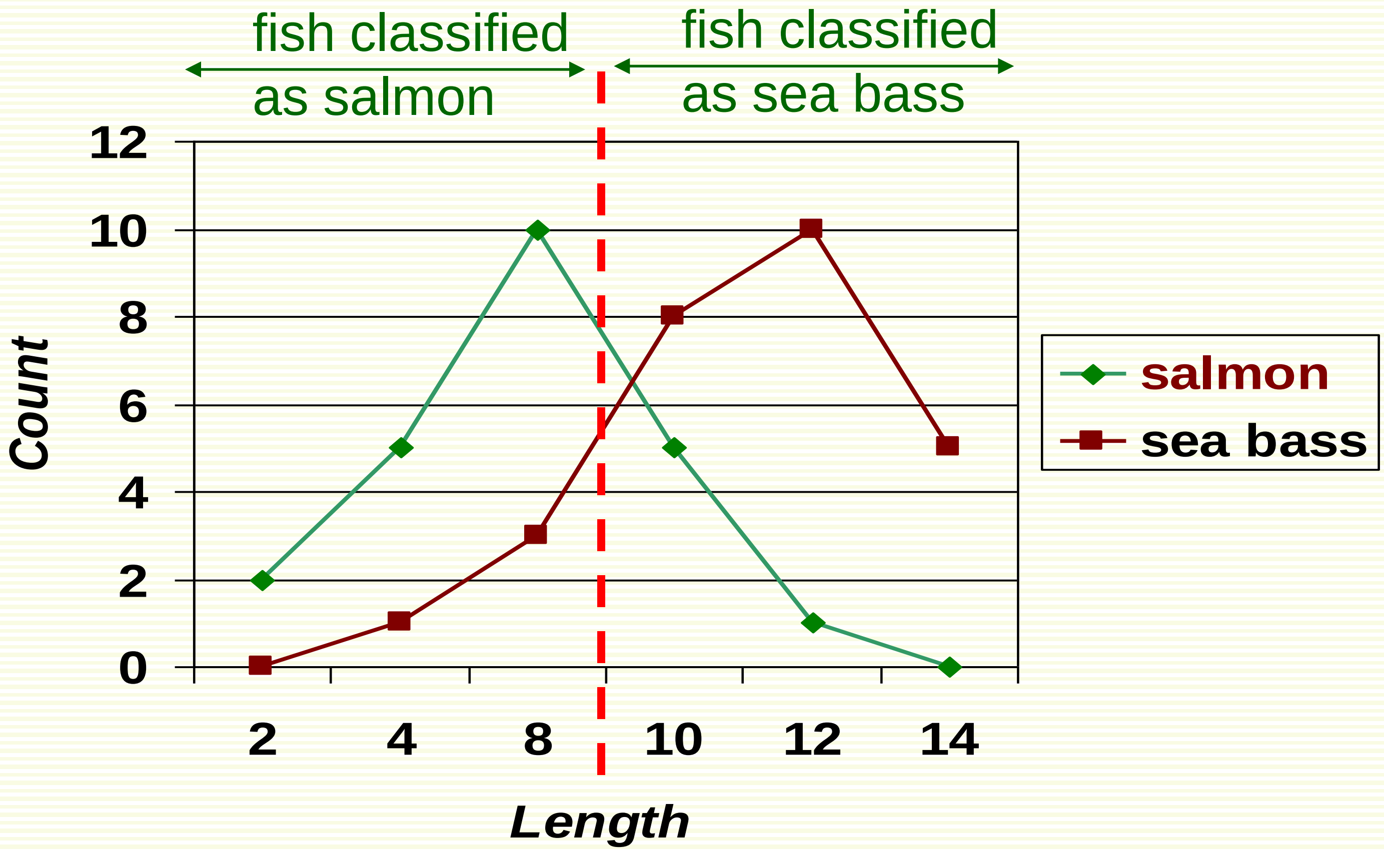
fish length > L → ***classify as sea bass***

- For example, at ***L*** = 5, misclassified:
  - 1 sea bass
  - 16 salmon

	2	4	8	10	12	14
bass	0	1	3	8	10	5
salmon	2	5	10	5	1	0

- Classification error (total error) $\frac{17}{50}$ = ***34%***

Single Feature (length) Classifier



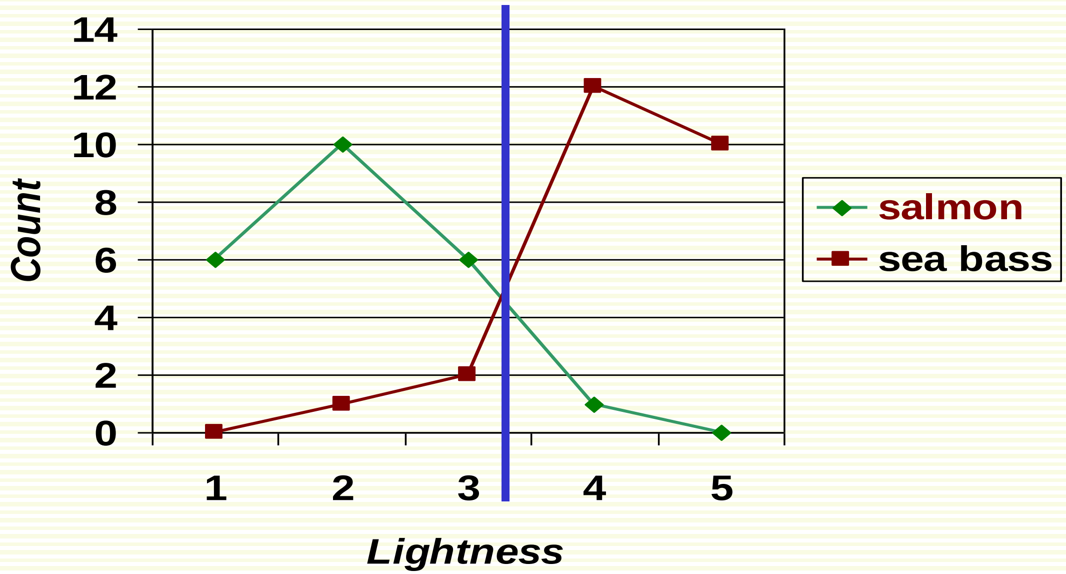
- After searching through all possible thresholds ***L***, the best ***L***= 9, and still 20% of fish is misclassified

Next Step

- Lesson learned
  - Length is a poor feature alone!
- What to do?
  - Try another feature
  - Salmon tends to be lighter
  - Try average fish lightness

Single Feature (lightness) Classifier

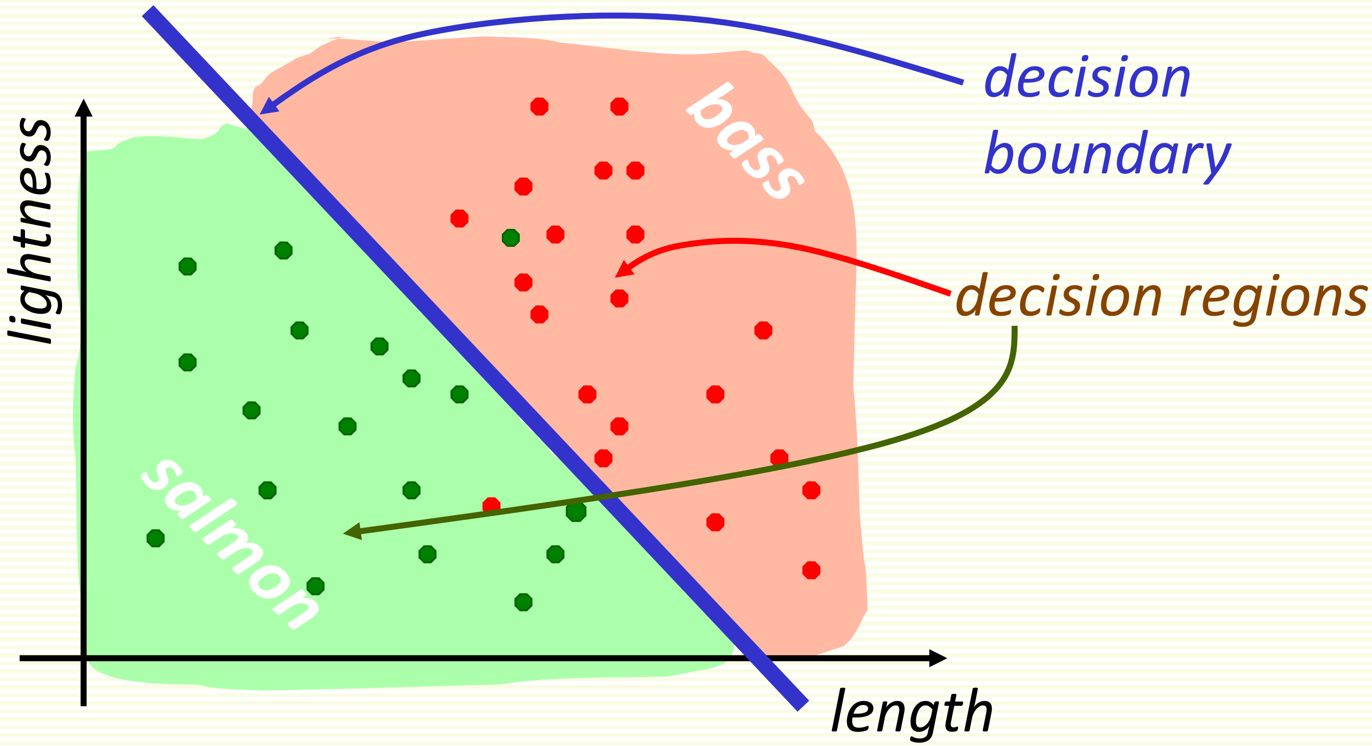
	1	2	3	4	5
bass	0	1	2	10	12
salmon	6	10	6	1	0



- Now fish are classified best at lightness threshold of 3.5 with classification error of 8%

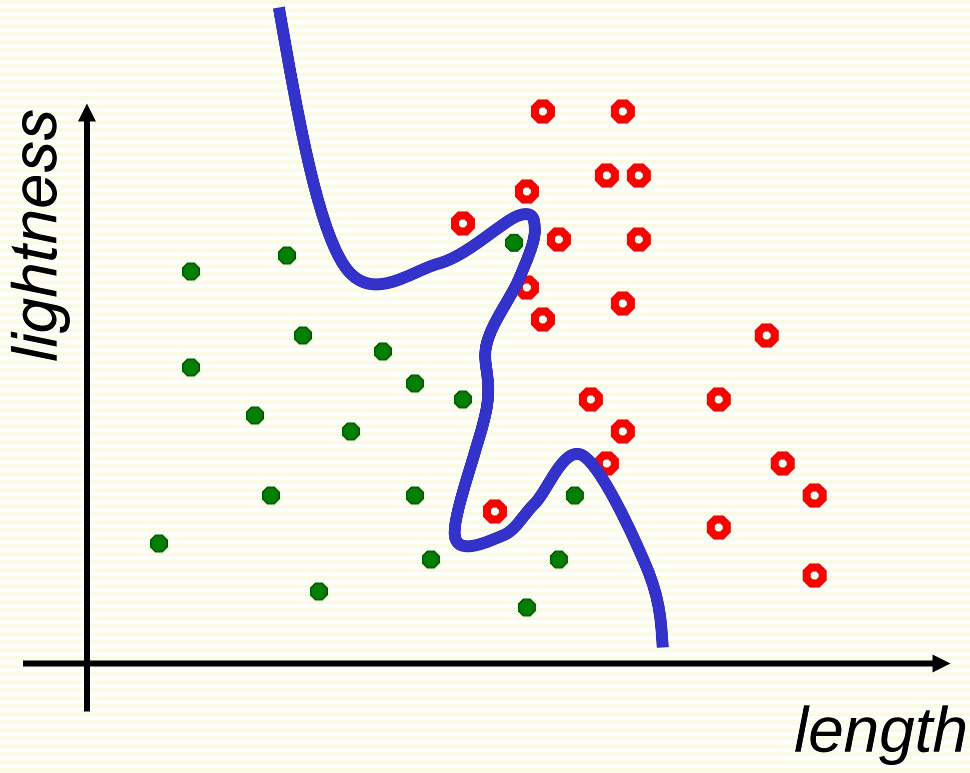
Can do better by feature combining

- Use both length and lightness features
- Feature vector [length,lightness]



- Classification error 4%

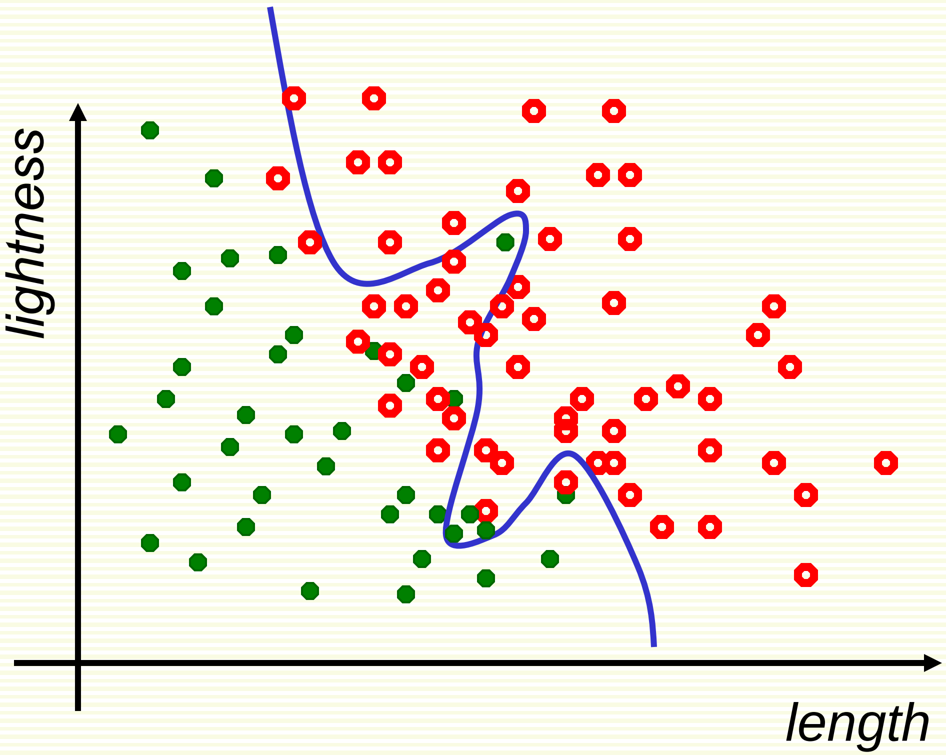
Even Better Decision Boundary



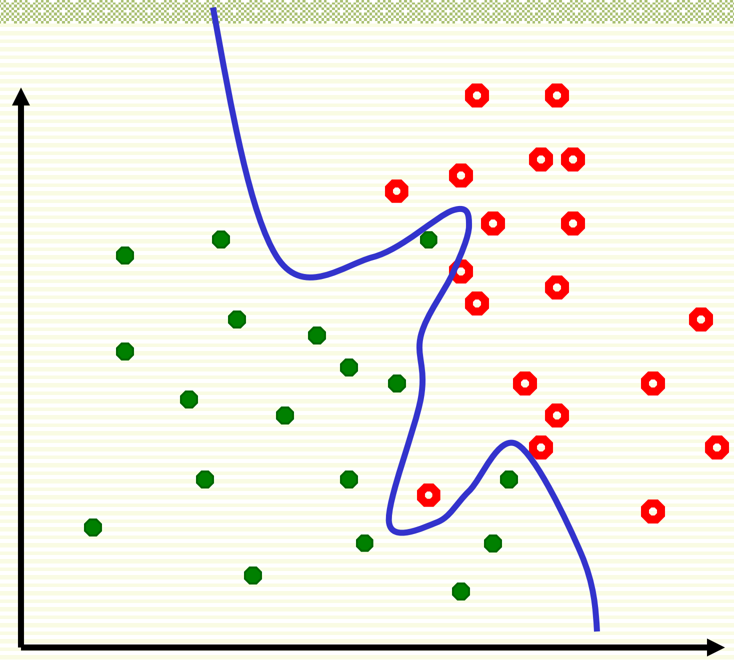
- Decision boundary (wiggly) with 0% classification error

Test Classifier on New Data

- The goal is for classifier to perform well on new data
- Test "wiggly" classifier on new data: 25% error



What Went Wrong: Overfitting



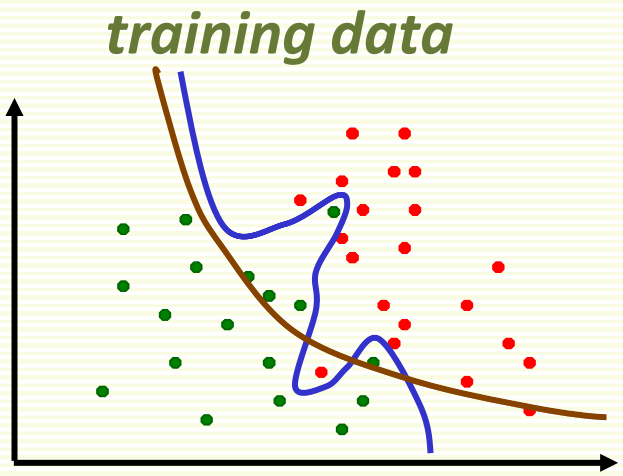
- Have only a limited amount of data for training
- Should ensure decision boundary does not adapt too closely to the particulars of training data, but grasps the “big picture”
- Complex boundaries **overfit** data, i.e. too tuned to training data

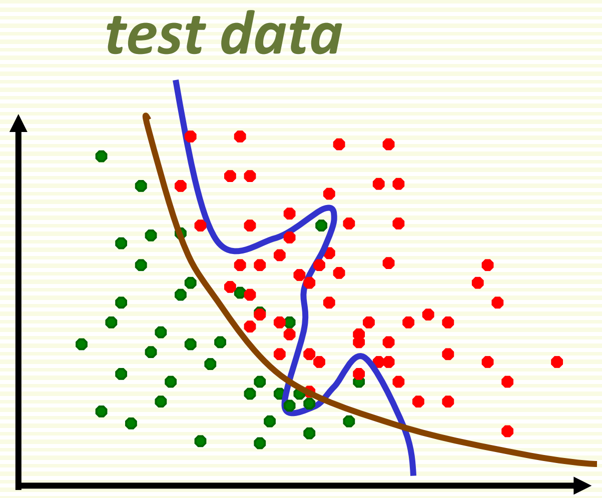
Overfitting: Extreme Example

- Say we have 2 classes: face and non-face images
- Memorize (i.e. store) all the "face" images
- For a new image, see if it is one of the stored faces
  - if yes, output "face" as the classification result
  - If no, output "non-face"
  - also called "rote learning"
- **problem**: new "face" images are different from stored "face" examples
  - zero error on stored data, 50% error on test (new) data
- Rote learning is memorization without generalization

slide is modified from Y. LeCun

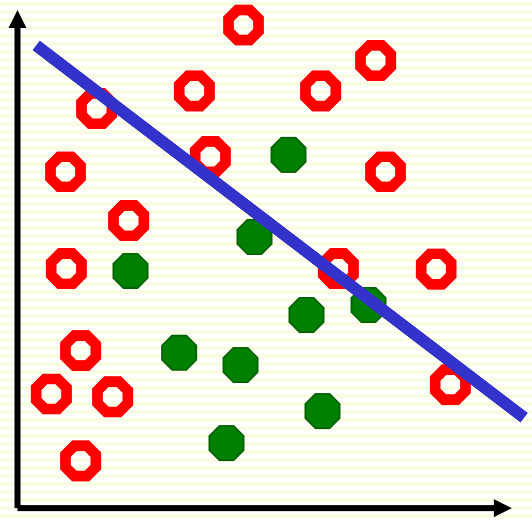
Generalization





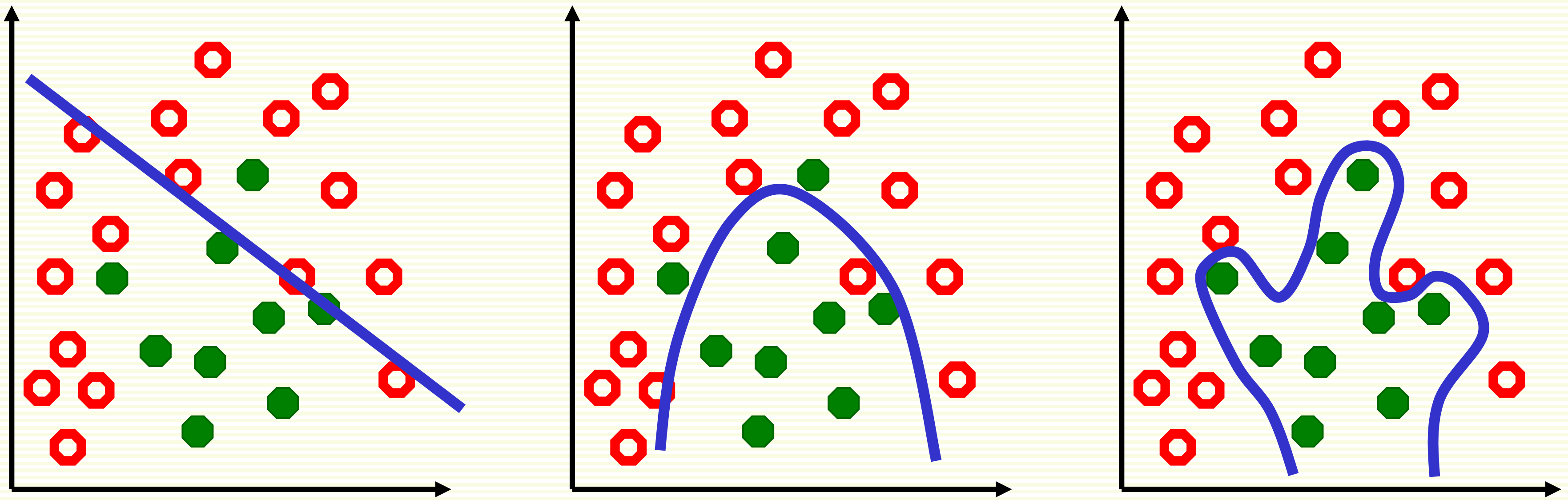
- The ability to produce correct outputs on previously unseen examples is called **generalization**
- The big question of learning theory: how to get good generalization with a limited number of examples
- Intuitive idea: favor simpler classifiers
  - William of Occam (1284-1347): “entities are not to be multiplied without necessity”
- Simpler decision boundary may not fit ideally to the training data but tends to generalize better to new data

Underfitting



- Also can underfit data, i.e. decision boundary too simple
- There is no way to fit a linear decision boundary so that the training examples are well separated

Underfitting → Overfitting



underfitting　　*"just right"*　　*overfitting*

Sketch of Supervised Machine Learning

- Chose a function **f**(**x**,**w**)
  - **w** are tunable weights
  - **x** is the input sample
  - **f**(**x**,**w**) should output the correct class of sample **x**
  - use labeled samples to tune weights **w** so that **f**(**x**,**w**) give the correct label for sample **x**
- Which function **f**(**x**,**w**) do we choose?
  - different choices will lead to decision boundaries of different complexities
  - has to be expressive enough to model our problem well, i.e. to avoid ***underfitting***
  - yet not to complicated to avoid ***overfitting***
- **f**(**x**,**w**) sometimes called *learning machine*

Training and Testing

- There are 2 phases, training and testing
  - Divide all labeled samples $\mathbf{x}^1, \mathbf{x}^2, \dots \mathbf{x}^n$ into 2 sets, *training* set and *test* set
  - Training phase is for "teaching" our machine (finding optimal weights **w**)
  - Testing phase is for estimating how well our machine works on unseen examples

Training Phase

- Find weights $\mathbf{w}$ s.t. $f(\mathbf{x}^i,\mathbf{w}) = \mathbf{y}^i$ "as much as possible" for *training* samples $\mathbf{x}^i$
  - "as much as possible" needs to be defined
  - How to search for such $\mathbf{w}$?
    - usually through optimization, can be quite time consuming

Testing Phase

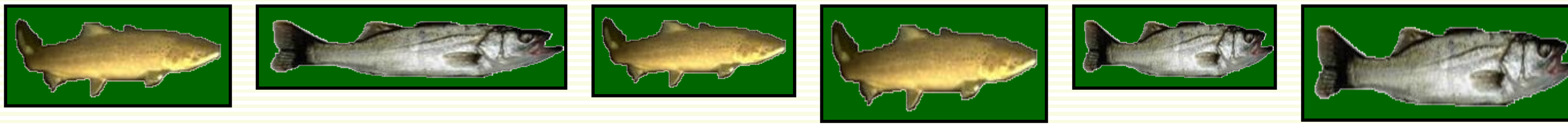
- The goal is good performance on unseen examples
- Evaluate performance of the trained classifier **f**(**x**,**w**) on the test samples (unseen labeled samples)
- Testing the machine on unseen labeled examples lets us approximate how well it will perform in practice
- If testing results are poor, may have to go back to the training phase and redesign **f**(**x**,**w**)

Generalization and Overfitting

- *Generalization* is the ability to produce correct output on previously unseen examples
  - i.e. low error on unseen examples
  - good generalization is the main goal of ML
- Low training error does not necessarily imply that we will have low test error
  - easy to produce **f**(**x**,**w**) which is perfect on training samples (rote "learning")
- *Overfitting*
  - occurs when low training error, high test error

Classification System Design Overview

- Collect and label data by hand



- Split data into training and test sets
- Preprocess by segmenting fish from background

- Extract possibly discriminating features
  - length, lightness, width, number of fins,etc.
- Classifier design
  - Choose model for classifier
  - Train classifier on training data
- Test classifier on test data

we look at these two steps in this course

Basic Linear Algebra

- Basic Concepts in Linear Algebra
  - vectors and matrices
  - products and norms
  - vector spaces and linear transformations
- Introduction to Matlab

Why Linear Algebra?

- For each example (e.g. a fish image), we extract a set of features (e.g. length, width, color)
- This set of features is represented as a *feature vector*
  - [length, width, color]
- All collected examples will be represented as collection of (feature) vectors

$[l_1, w_1, c_1]$ example 1

$[l_2, w_2, c_2]$ example 2

$[l_3, w_3, c_3]$ example 3

$$\longrightarrow \begin{bmatrix} l_1 & w_1 & c_1 \\ l_2 & w_2 & c_2 \\ l_3 & w_3 & c_3 \end{bmatrix}$$

matrix

- Also, we will use linear models since they are simple and computationally tractable

What is a Matrix?

- A matrix is a set of elements, organized into rows and columns

rows

columns

$$
\begin{array}{c|cccc|l}
 & 2 & 7 & 6 & 10 & \textbf{example 1} \\
 & 1 & 4 & 4 & 9 & \textbf{example 2} \\
 & 6 & 4 & 9 & 6 & \textbf{example 3} \\
\hline
 & \textbf{feature 1} & \textbf{feature 2} & \textbf{feature 3} & \textbf{feature 4} &
\end{array}
$$

Basic Matrix Operations

- addition, subtraction, multiplication by a scalar

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$$ add elements

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} - \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a-e & b-f \\ c-g & d-h \end{bmatrix}$$ subtract elements

$$\alpha \cdot \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} \alpha \cdot a & \alpha \cdot b \\ \alpha \cdot c & \alpha \cdot d \end{bmatrix}$$ multiply every entry

Matrix Transpose

- n by m matrix A and its m by n transpose A^T

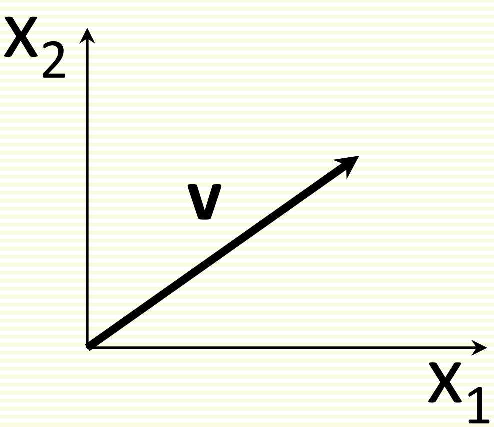
$$A = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \qquad A^T = \begin{bmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & \cdots & x_{n2} \\ \vdots & \vdots & \cdots & \vdots \\ x_{1m} & x_{2m} & \cdots & x_{nm} \end{bmatrix}$$

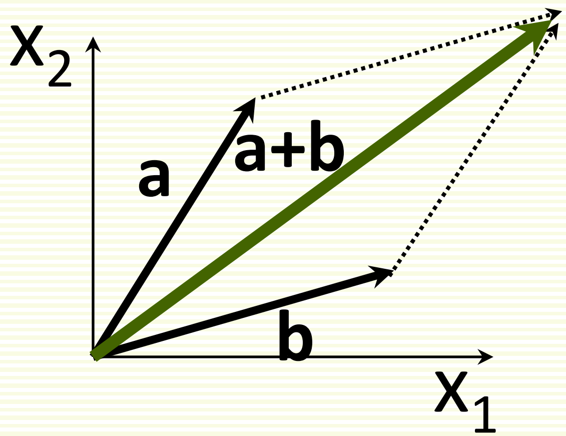
Vectors

- Vector: N x 1 matrix

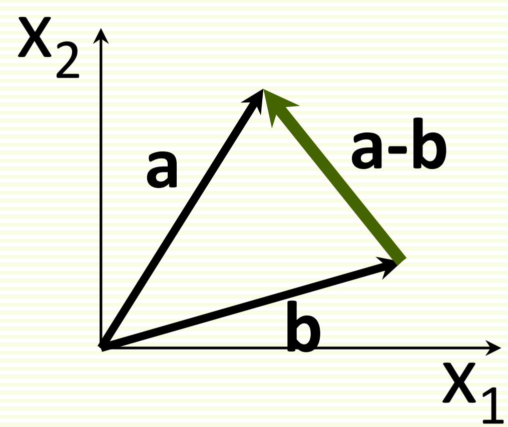
$$v = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- **dot product** and **magnitude** defined on vectors only





vector addition



vector subtraction

More on Vectors

- n-dimensional row vector $x = [x_1 \quad x_2 \quad \dots \quad x_n]$

- Transpose of row vector is column vector $x^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

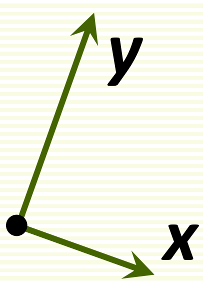
- *Vector* product (or *inner* or *dot* product)

$$\langle x, y \rangle = x \cdot y = x^T y = x_1 y_1 + x_2 y_2 + \dots + x_n y_n = \sum_{i=1\dots n} x_i y_i$$

More on Vectors

- *Euclidian norm* or *length* $\|x\| = \sqrt{\langle x, x \rangle} = \sqrt{\sum_{i=1\ldots n} x_i^2}$
- If $\|\boldsymbol{x}\| = 1$ we say $\boldsymbol{x}$ is *normalized* or *unit* length
- angle *q* between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$: $\cos\theta = \dfrac{x^T y}{\|x\|\|y\|}$
- inner product captures direction relationship

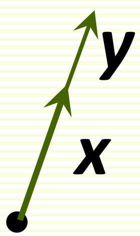
$\cos\theta = 0$



$x^T y = 0$

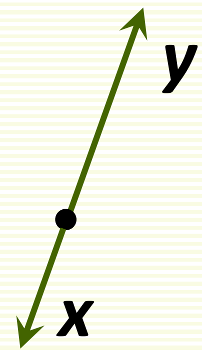
$x \perp y$

$\cos\theta = 1$



$x^T y = \|x\|\|y\| > 0$

$\cos\theta = -1$



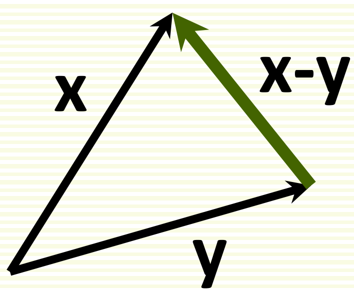
$x^T y = -\|x\|\|y\| < 0$

More on Vectors

- Vectors x and y are orthonormal if they are orthogonal and ||x|| = ||y|| =1

- Euclidian distance between vectors x and y

$$\left\| x - y \right\| = \sqrt{\sum_{i=1\ldots n} \left(x_i - y_i \right)^2}$$



Linear Dependence and Independence

- Vectors $x_1, x_2, \ldots, x_n$ are linearly dependent if there exist constants $\alpha_1, \alpha_2, \ldots, \alpha_n$ s.t.
  - $\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n = 0$
  - $\alpha_i \neq 0$ for at least one i

- Vectors $x_1, x_2, \ldots, x_n$ are linearly independent if $\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n = 0 \Rightarrow \alpha_1 = \alpha_2 = \ldots = \alpha_n = 0$

Vector Spaces and Basis

- The set of all n-dimensional vectors is called a vector space ***V***
  - A set of vectors $\{u_1, u_2, \ldots, u_n\}$ are called a basis for vector space if any ***v*** in ***V*** can be written as

$$\boldsymbol{v} = \alpha_1 u_1 + \alpha_2 u_2 + \ldots + \alpha_n u_n$$

- $u_1, u_2, \ldots, u_n$ are independent implies they form a basis, and vice versa
- $u_1, u_2, \ldots, u_n$ give an orthonormal basis if
  1. $\|u_i\| = 1 \quad \forall i$
  2. $u_i \perp u_j \quad \forall i \neq j$

Orthonormal Basis

- x, y,..., z form an orthonormal basis

$$x = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T \qquad x \cdot y = 0$$

$$y = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T \qquad x \cdot z = 0$$

$$z = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \qquad y \cdot z = 0$$

Matrix Product

$$AB = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1d} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nd} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{1m} \\ b_{21} & \cdots & b_{2m} \\ b_{31} & \cdots & b_{3m} \\ \vdots & \cdots & \vdots \\ b_{d1} & \cdots & b_{dm} \end{bmatrix} = \begin{bmatrix} & & \\ & c_{ij} & \\ & & \end{bmatrix}$$

$c_{ij} = \langle a^i, b_j \rangle$

a^i is row ***i*** of ***A***

b_j is column ***j*** of ***B***

- # of columns of A = # of rows of B
- even if defined, in general AB ≠ BA

Matrices

- **Rank** of a matrix is the number of linearly independent rows (or equivalently columns)
- A square matrix is non-singular if its rank equal to the number of rows. If its rank is less than number of rows it is singular.
- **Identity matrix**
  AI=IA=A

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

- Matrix ***A*** is **symmetric** if $A = A^T$

$$\begin{bmatrix} 1 & 2 & 9 & 5 \\ 2 & 7 & 4 & 8 \\ 9 & 4 & 3 & 6 \\ 5 & 8 & 6 & 4 \end{bmatrix}$$

Matrices

- **Inverse** of a square matrix $\boldsymbol{A}$ is matrix $\boldsymbol{A}^{-1}$ s.t. $\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I}$
- If $\boldsymbol{A}$ is singular or not square, inverse does not exist
- Pseudo-inverse $\boldsymbol{A}^{\dagger}$ is defined whenever $\boldsymbol{A}^{T}\boldsymbol{A}$ is not singular (it is square)
  - $\boldsymbol{A}^{\dagger} = (\boldsymbol{A}^{T}\boldsymbol{A})^{-1}\boldsymbol{A}^{T}$
  - $\boldsymbol{A}^{\dagger}\boldsymbol{A} = (\boldsymbol{A}^{T}\boldsymbol{A})^{-1}\boldsymbol{A}^{T}\boldsymbol{A} = \boldsymbol{I}$

MATLAB

- Starting matlab
  - xterm -fn 12X24
  - matlab
  - matlab -nodisplay
- Basic Navigation
  - quit
  - more
  - help general
- Scalars, variables, basic arithmetic
  - Clear
  - + - * / ^
  - help arith
- Relational operators
  - ==,&,|,~,xor
  - help relop
- Lists, vectors, matrices
  - A=[2 3;4 5]
  - A’
- Matrix and vector operations
  - find(A>3), colon operator
  - * / ^ .* ./ .^
  - eye(n),norm(A),det(A),eig(A)
  - max,min,std
  - help matfun
- Elementary functions
  - help elfun
- Data types
  - double
  - Char
- Programming in Matlab
  - .m files
  - scripts
  - function y=square(x)
  - help lang
- Flow control
  - if i== 1else end, if else if end
  - for i=1:0.5:2 … end
  - while i == 1 … end
  - Return
  - help lang
- Graphics
  - help graphics
  - help graph3d
- File I/O
  - load,save
  - fopen, fclose, fprintf, fscanf