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

• **Classification**
  - \( y^i \) takes value in finite set, called a *label* or a *class*
  - Example: \( y^i \in \{"sunny", "cloudy", "raining"\} \)

• **Regression**
  - \( y^i \) continuous, called an *output value*
  - Example: \( y^i = \text{temperature} \in [-60,60] \)
Toy Application: fish sorting

- classifier
- fish species
- fish image
- sorting chamber

- salmon
- sea bass
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

<table>
<thead>
<tr>
<th>Length</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>bass</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>salmon</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Single Feature (length) Classifier

- Find the best length $L$ threshold

\[
\begin{align*}
\text{fish length} &< L \\
\text{fish length} &> L \\
\text{classify as salmon} & \\
\text{classify as sea bass}
\end{align*}
\]

- For example, at $L = 5$, misclassified:

  - 1 sea bass
  - 16 salmon

\[
\begin{array}{ccccccc}
 & 2 & 4 & 8 & 10 & 12 & 14 \\
\hline
\text{bass} & 0 & 1 & 3 & 8 & 10 & 5 \\
\text{salmon} & 2 & 5 & 10 & 5 & 1 & 0 \\
\end{array}
\]

- Classification error (total error) $\frac{17}{50} = 34\%$
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
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 \([\text{length, lightness}]\)

- Classification error 4%
• 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*
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.
• 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 $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^n$ into 2 sets, \textit{training} set and \textit{test} set
  - Training phase is for “teaching” our machine (finding optimal weights $\mathbf{w}$)
  - Testing phase is for estimating how well our machine works on unseen examples
Training Phase

• Find weights \( w \) s.t. \( f(x^i, w) = y^i \) “as much as possible” for training samples \( x^i \)
  • “as much as possible” needs to be defined
  • How to search for such \( w \)?
    • 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
- 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

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

- 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.
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 $A^T$

$$A = \begin{bmatrix}
  x_{11} & x_{12} & \cdots & x_{1m} \\
  x_{21} & x_{22} & \cdots & x_{2m} \\
  \vdots & \vdots & \cdots & \vdots \\
  x_{n1} & x_{n2} & \cdots & x_{nm}
\end{bmatrix} \quad A^T = \begin{bmatrix}
  x_{11} & x_{21} & \cdots & x_{n1} \\
  x_{12} & x_{22} & \cdots & x_{n2} \\
  \vdots & \vdots & \cdots & \vdots \\
  x_{1m} & x_{2m} & \cdots & x_{nm}
\end{bmatrix}$$
• 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 \( x = \begin{bmatrix} x_1 & x_2 & \ldots & x_n \end{bmatrix} \)

- 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)

\[
\langle x, y \rangle = x \cdot y = x^T y = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n = \sum_{i=1}^{n} x_i y_i
\]
More on Vectors

- **Euclidian norm or length** \( \|x\| = \sqrt{\langle x, x \rangle} = \sqrt{\sum_{i=1}^{n} x_i^2} \)

- If \( \|x\| = 1 \) we say \( x \) is *normalized* or *unit* length

- angle \( \theta \) between vectors \( x \) and \( y \): \( \cos \theta = \frac{x^T \cdot y}{\|x\| \cdot \|y\|} \)

- inner product captures direction relationship

\[
\begin{align*}
\cos \theta &= 0 \\
\cos \theta &= 1 \\
\cos \theta &= -1
\end{align*}
\]

\[
\begin{align*}
x^T y &= 0 \\
x \perp y \\
x^T y &= \|x\| \cdot \|y\| > 0 \\
x^T y &= -\|x\| \cdot \|y\| < 0
\end{align*}
\]
More on Vectors

• Vectors $x$ and $y$ are orthonormal if they are orthogonal and $\|x\| = \|y\| = 1$

• Euclidian distance between vectors $x$ and $y$:

$$\|x - y\| = \sqrt{\sum_{i=1\ldots n} (x_i - y_i)^2}$$
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 $\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n = 0 \implies \alpha_1 = \alpha_2 = \ldots = \alpha_n = 0$
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 \( \mathbf{v} \) in \( V \) can be written as

\[
\mathbf{v} = \alpha_1 u_1 + \alpha_2 u_2 + \ldots + \alpha_n u_n
\]

• \( u_1, u_2, \ldots, u_n \) 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. \( u_i \perp u_j \quad \forall i \neq j \)
• $x, y, ..., z$ form an orthonormal basis

\[
x = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T \quad x \cdot y = 0
\]

\[
y = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T \quad x \cdot z = 0
\]

\[
z = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \quad y \cdot z = 0
\]
Matrix Product

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

\[
c_{ij} = \langle a^i, b_j \rangle
\]
\[a^i \text{ is row } i \text{ of } A
\]
\[b_j \text{ is column } j \text{ of } B
\]

• # 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 equals the number of rows. If its rank is less than the number of rows, it is singular.

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

- **Identity matrix**
  $$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$
  $$AI=IA=A$$

- **Matrix** $A$ is **symmetric** if $A=A^T$
• **Inverse** of a square matrix $A$ is matrix $A^{-1}$ s.t.

$$AA^{-1} = I$$

• If $A$ is singular or not square, inverse does not exist

• **Pseudo-inverse** $A^\dagger$ is defined whenever $AA^\top$ is not singular (it is square)

  • $A^\dagger = (A^\top A)^{-1} A^\top$
  • $A^\dagger = (A^\top A)^{-1} A^\top$
  • $AA = (A^\top A)^{-1} A^\top = I$
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==1 else 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`