Introduction to Machine Learning

Guillaume Wisniewski
guillaume.wisniewski@limsi.fr
January 2018
Université Paris Sud — LIMSI

Why should I study Machine Learning?

A first question...

- Given a large corpus (= set) of text files, you want to count the number of different URL
- How would you do that?

It's easy

Principle

1. Find a way to extract URL
   - regexp (from `http` until the next space)
   - library?
2. collect all URL and unify them
3. count

An important detail

- easy when you have Gb of texts...
- what happens when you have Tb?

It's easy

Principle

1. Find a way to extract URL
   - regexp (from `http` until the next space)
   - library?
2. collect all URL and unify them
3. count

An important detail

- easy when you have Gb of texts...
- what happens when you have Tb?
  ⇒ Specific architecture (Hadoop)
  ⊕ impossible to know if answer is correct

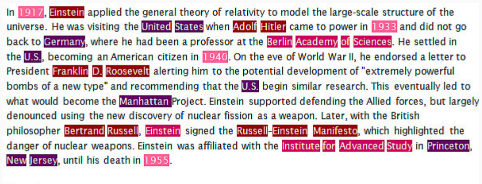
For fun: "Correct" regexp to detect an URL

https://gist.github.com/gruber/249502

```
(?xi)
\b
(                                                   # Capture 1: entire matched URL
  (?:
    [a-z][\w-]+:                                # URL protocol and colon
    (?:
      /{1,3}                                            # 1-3 slashes
      |                                                     #   or
      [a-z0-9%]                                             # Single letter or digit or '%'
                                                            # (Trying not to match e.g. "URI::Es
    )
    |                                                   #   or
    www\d{0,3}[.]                               # "www.", "www1.", "www2." ... "www999."
    |                                                   #   or
    [a-z0-9.\-]+[.][a-z]{2,4}/          # looks like domain name followed by a slash
  )
  (?:                                                   # One or more:
    [^\s()<>]+                                          # Run of non-space, non-()<>
    |                                                       #   or
    \(([^\s()<>]+|(\([^\s()<>]+\)))*\)        # balanced parens, up to 2 levels
  )+
  (?:                                                   # End with:
    \(([^\s()<>]+|(\([^\s()<>]+\)))*\)        # balanced parens, up to 2 levels
    |                                                           #   or
    [^\s`!()\[\]{};:'".,<>?«»“”‘’]                  # not a space or one of these punct char
  )
```

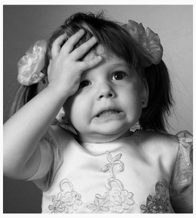
A second question...

New question: identify Named Entity (NER) in a text



In 1917, Einstein applied the general theory of relativity to model the large-scale structure of the universe. He was visiting the United States when Adolf Hitler came to power in 1933 and did not go back to Germany, where he had been a professor at the Berlin Academy of Sciences. He settled in the U.S., becoming an American citizen in 1940. On the eve of World War II, he endorsed a letter to President Franklin D. Roosevelt alerting him to the potential development of "extremely powerful bombs of a new type" and recommending that the U.S. begin similar research. This eventually led to what would become the Manhattan Project. Einstein supported defending the Allied forces, but largely denounced using the new discovery of nuclear fission as a weapon. Later, with the British philosopher Bertrand Russell, Einstein signed the Russell–Einstein Manifesto, which highlighted the danger of nuclear weapons. Einstein was affiliated with the Institute for Advanced Study in Princeton, New Jersey, until his death in 1955.

This time...



- gazetteer: list of first name, geographical places, ...
- not enough
  - language is ambiguous (e.g. Tim Cook)
  - language is evolving (new TV star, new places, new language of interest, ...)
- answer is obvious (most of the time), but hard to explain
  - fuzzy decisions, lots of 'weak', contradictory signals, general knowledge, ...
  - e.g., 'The spokesperson, Μιχάλης Χατζόπουλος, has declared that...'

Another example...

Which of these persons is a woman?

- obvious!!!
- how do you know? how to build an algorithm out of these intuitions?

⇒ fuzzy decision, several criteria, ... again

The Machine Learning paradigm

How to make a 'computer' do a specific task?

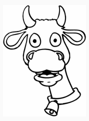
'Traditional approach'
A program is:

- hand-coded
- specific set of instructions to accomplish the task
- can be explained and proved ⇒ always gives the correct answer

Machine Learning
A program is trained:

- from large amount of annotated data
- algorithm + inductive bias
- works on average

Principle of a 'learning algorithm'

x:  What is this animal? y = cow

$x \longrightarrow$ [parametrized algorithm] $\longrightarrow \tilde{y}$

- parameters = chosen on a set of annotated examples
- learning = chose parameters so that $y_i \simeq \tilde{y}_i$ = induction 
- generalization for unknown x: $y \simeq \tilde{y}$

Three learning tasks

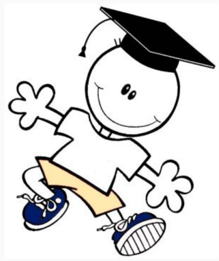
binary classification predict a yes/no response
e.g.: is this mail a spam? is this review positive or negative?

multi-class classification put an example into one of a number of classes
e.g.: is this picture an animal, a person or a car?

regression predict a real value
e.g.: price of a house, size of a tumor, ...

First learning algorithm: decision tree (with Hal Daumé III)

Recommender System



The task

- Predict if a student (e.g. Maryam) is going to like a lectures (e.g. Algorithms) or not
- given a set of lectures and a set of students
- each student has taken and evaluated a subset of the lectures

Do our system learn something?

The setting

- we are given: examples (i.e. pairs of student/course) and their label (like/dislike)
- these examples are called training data

Generalization

- predicting if Maryam will like a course she has already taken is easy
  - memorization, no learning
- the system must be evaluated on unseen examples = test set

Example of training data

	Algorithms	OOP	Machine Learning	Graphs
Maryam	yes	no	no	yes
Nadi	no	yes	yes	no
Diaa	yes	no	no	yes
Oana	yes	no	yes	no
		...		

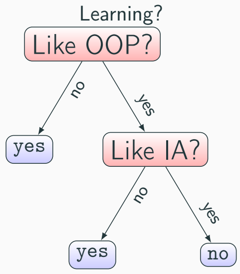
How to make a prediction?

By I asking a series of binary questions:
You: Is the course under consideration in Systems?
Me: Yes
You: Has this student taken any other Systems courses? Me: Yes
You: Has this student like most previous Systems courses?
Me: No
You: *I predict this student will not like this course.*

Goal of learning: which questions to ask? in what order? which answer?

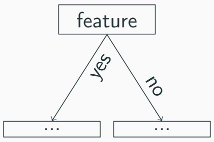
Decision tree

Will a student like Machine Learning?



- the questions can be organized in a tree
- questions = internal nodes
- predictions = leaves
- question = features, answer = feature value

Modeling a decision tree



- tree = set of nodes
- a node is either:
  - a leaf associated to a label
  - a triplet $\langle \text{feature} | \text{left\_node} | \text{right\_node} \rangle$

Recommending a new lecture

```
def predict(tree, example):
  if is_leave(tree):
     return get_label(tree)

  if get_label(tree) in example:
     return predict(get_left(tree), example)
  else:
     return predict(get_right(tree), example)
```


Training a decision tree

Principle

- if there are n features? how many trees can be built?

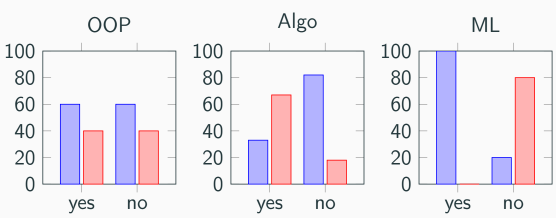
Training a decision tree

Principle

- if there are n features? how many trees can be built? $n!$
- we can not enumerate all trees to select the best ⇒ greedy search
- divide and conquer: if I could only ask one question, what question would I ask?

How to select the best feature?

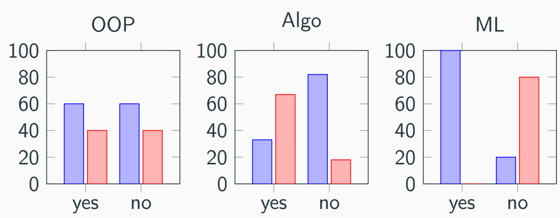
Look at the histogram of labels for each feature!
= frequency of like/dislike for each possible feature value



Which question will you ask?

How to select the best feature?

Look at the histogram of labels for each feature!
= frequency of like/dislike for each possible feature value



Which question will you ask?

The most discriminative feature is 'ML': we will know if the student like the course or not with a high confidence

At the end...

- pick the best feature, i.e. the one that will separate the data the best
- partition the examples into 2 parts according to their feature value
- ask the same question

Final Algorithm i

```
def train(examples, remaining_features):
    guess = most_frequent_label(examples)
    if all examples have the same label:
        return create_leaf(guess)
    if remaining_features is empty:
        return create_leaf(guess)

    for f in remaining_features:
        no = all examples for which f is no
        yes = all examples for which f is yes
        scores[f] = n. majority votes in yes + \
                    n. majority vote in no
```


Final Algorithm ii

```
    best_feature = argmax(scores)

    no = all examples for which best_feature is no
    yes = all examples for which best_feature is yes
    features = all remaning features but best_features
    left = train(no, features)
    right = train(yes, features)

    return create_node(best_features, left, right)
```


In real life

- this is a 'simplified' training algorithm
- goal = illustrating some important notions rather than achieving high prediction performance
- in real life: ID3, C4.5, ...
  - better scoring function
  - pruning

Evaluating a learning algorithm

Principle

- performance measured on unseen 'test' data
- train data 'similar' to test data
- must be automatic and repeatable

Loss function

- main idea: compare the expected result to the prediction
- loss function: measure how 'bad' a system is
- 0/1 loss:

$$\ell^{0/1}(y, \tilde{y}) = \begin{cases} 0 & \text{if } y = \tilde{y} \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

(other loss functions can be considered)

Probabilistic model of learning

Data generation distribution

- formalize the idea that example in the train and test sets look the same
- we assume that there is fix bug unknown probability distribution $\mathcal{D}$ that generates the examples (x, y)

Quality of a learning algorithm

Given:

- a loss function ℓ
- the generating distribution $\mathcal{D}$ (i.e. a definition of the data we expect to see)
- a classifier f

The quality of f can be measured by:

$$\epsilon = \mathbb{E}_{(x,y)\sim\mathcal{D}}[l(y, f(x))] = \sum_{(x,y)} \mathcal{D}(x, y) \cdot \ell(y, f(x)) \qquad (2)$$

- expected loss
- must be minimized... but can not be computed

Expected loss

The problem with expected loss

- computing ϵ requires to know all possible observations x and their label
- can only be estimated from a sample of the data = train set

Training error

- given a training set $(x_i, y_i)_{i=1}^N$

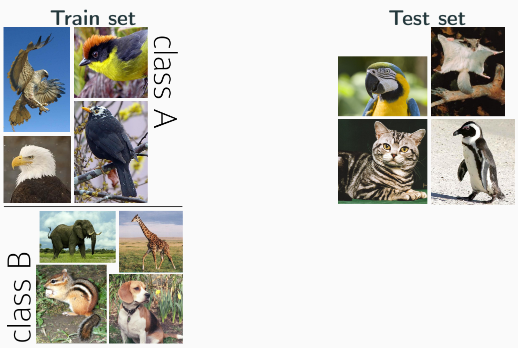
$$\epsilon_{\text{train}} = \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, f(x_i)) \qquad (3)$$

- number of errors on train set
- easy to achieve $\epsilon_{\text{train}} = 0$, but not the goal

Formalizing the Learning Problem

Given (i) a loss function ℓ and (ii) a sample D from some unknown distribution $\mathcal{D}$, compute a function f that has low expected error ϵ over $\mathcal{D}$ with respect to ℓ.

Inductive Bias (1)



What are the labels of the test set?

Inductive bias (2)

Solution

- AABB (60-70% of the answers) : bird/no-bird
- ABBA (30-40% of the answers) : fly/no-fly

impossible to decide based only on train data

Inductive bias

- what we know before the data arrive
- personal 'rule' to be able to generalize
- no generalization without inductive bias
- main difference of learning algorithm

Inductive bias of decision trees

Shallow decision tree

- tree can not query more than d features
- d is a predefined parameter = hyper-parameter

Inductive bias of shallow decision tree

- decision can be made by looking at a small number of features
- not able learn something like: a student likes ML only if has liked an even number of lectures

Under/over-fitting

'Extreme' decision trees

- empty decision tree (no question asked) — arbitrary training error
- full decision trees (query all features, arbitrary decision when no example in node) — training error is null

Generalization capacity of extreme trees

- empty tree: error will not change much
- full tree:
  - will be correct for examples that have been seen in the train
  - 50% of error for the other examples

  $\Rightarrow$ almost 40% of errors

Definition

underfitting do not 'learn' / 'extract' all information available in the data

overfitting pay too much attention to idiosyn[illegible]s of data

In practice: algorithm is not able to generalize

Conclusions

What you are supposed to know?

- difOAference between memorization and generalization
- inductive bias and its role in learning
- underfitting versus overfitting
- decide whether a machine learning algorithm is cheating or not?