Machine Learning

ML for NLP
Lecturer: Kevin Koidl Assist. Lecturer Alfredo Maldonado

https://www.cs.tcd.ie/kevin.koidl/cs4062/
kevin.koidl@scss.tcd.ie, maldonaa@tcd.ie

2017

Outline

- Does TC (and NLP) need Machine Learning?
- What can Machine Learning do for us ("what has machine learning ever done for us"?)
- What is machine learning?
- How do we design machine learning systems?
- What is a well-defined learning problem?
- An example

Why Machine Learning

- Progress in algorithms and theory
- Growing flood of online data
- Computational power is available
- Rich application potential
- The *knowledge acquisition bottleneck*

Well-Defined Learning Problems

- Learning = Improving with experience at some task
  - Improve over Task T,
  - with respect to performance measure P,
  - based on experience E.
- Example Checkers
  - T: Play Checkers,
  - P: Percentage of Games won in a tournament,
  - P: Games played against self.

Machine Learning Definitions

> Machine learning is the subfield of computer science that gives computers the ability to *learn* without being explicitly programmed (Arthur Samuel, 1959).

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E (Tom M. Mitchell, 1998).

Machine Learning Challenges

- How can a computer program make an experience?
- How can this experience be codified?
- Examples of a codified experience?

User interface agents?

- An (artificial) agent may help users cope with increasing information:

  > An *agent* is a computer system that is *situated* in some *environment* and that is capable of *autonomous action* in its environment in order to meet its design objectives. (Wooldridge, 2002)

- Definition of a Rational Agent:

  > A rational agent should select an action that is expected to maximize its performance measure, given the evidence provided. [Peter Norvig, 2003]

Do Agents need machine learning?

- Practical concerns:
  - large amounts of language data have become available (on the web and elsewhere), and one needs to be able to make sense of them all,
  - knowledge engineering methods don't seem to be able to cope with the growing flood of data
  - Machine learning can be used to automate knowledge acquisition and inference
- Theoretical contributions:
  - reasonably solid foundations (theory and algorithms)

Machine Learning Categories

- Main Machine Learning Categories:
  - Supervised Learning: Computer receives input and output data aka 'labelled' data and creates a 'mapping' between both.
  - Unsupervised Learning: Input data has no labels are given. Learning algorithm has to identify structure in the input data.

Supervised Learning

- Supervised Machine Learning Problem Categories:
  - Regression Problem: Continous Output. For example predict percentage grade (e.g. 76%) based on hours studied.
  - Classification Problem: Discrete Output. For example predict grade (e.g. A) based on hours studied.

Supervised Learning Models

- Tyical Machine Learning Models:
  - Supervised Vector Machines (SVM)
  - Gaussian Process
  - Artifical Neural Networks (ANN)
  - Classification, Regression, Decission Trees, Random Forrest, Back Propagation....
- What is the goal of a model and how do I select the right one?

Application niches for machine learning

- ML for text classification for use in, for instance, self customizing programs:
  - Newsreader that learns user interests
- Data mining: using historical data to improve decisions
  - medical records → medical knowledge
  - analysis of customer behaviour
- Software applications we can't program by hand
  - autonomous driving
  - speech recognition

Examples: data mining problem

Patient103 time=1	*Patient103* time=2	...	*Patient103* time=n
Age: 23	Age: 23		Age: 23
FirstPregnancy: no	FirstPregnancy: no		FirstPregnancy: no
Anemia: no	Anemia: no		Anemia: no
Diabetes: no	Diabetes: YES		Diabetes: no
PreviousPrematureBirth: no	PreviousPrematureBirth: no		PreviousPrematureBirth: no
Ultrasound: ?	Ultrasound: abnormal		Ultrasound: ?
Elective C–Section: ?	Elective C–Section: no		Elective C–Section: no
Emergency C–Section: ?	Emergency C–Section: ?		**Emergency C–Section: Yes**
...	...		...

Given:

- 9714 patient records, each describing a pregnancy and birth
- Each patient record contains 215 features

Learn to predict:

- Classes of future patients at high risk for Emergency Cesarean Section

Examples: data mining results

Patient103 time=1	*Patient103* time=2	...	*Patient103* time=n
Age: 23	Age: 23		Age: 23
FirstPregnancy: no	FirstPregnancy: no		FirstPregnancy: no
Anemia: no	Anemia: no		Anemia: no
Diabetes: no	Diabetes: YES		Diabetes: no
PreviousPrematureBirth: no	PreviousPrematureBirth: no		PreviousPrematureBirth: no
Ultrasound: ?	Ultrasound: abnormal		Ultrasound: ?
Elective C–Section: ?	Elective C–Section: no		Elective C–Section: no
Emergency C–Section: ?	Emergency C–Section: ?		**Emergency C–Section: Yes**
...	...		...

```
If   No previous vaginal delivery, and
     Abnormal 2nd Trimester Ultrasound, and
     Malpresentation at admission
Then Probability of Emergency C-Section is 0.6

 Over training data: 26/41 = .63,
 Over test data: 12/20 = .60
```

Other prediction problems

- Customer purchase behavior:

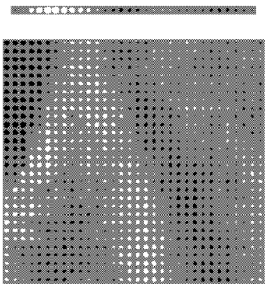
Customer103: (time=t0)	*Customer103:* (time=t1)	...	*Customer103:* (time=tn)
Sex: M	Sex: M		Sex: M
Age: 53	Age: 53		Age: 53
Income: $50k	Income: $50k		Income: $50k
Own House: Yes	Own House: Yes		Own House: Yes
MS Products: Word	MS Products: Word		MS Products: Word
Computer: 386 PC	Computer: Pentium		Computer: Pentium
Purchase Excel?: ?	Purchase Excel?: ?		**Purchase Excel?: Yes**
...	...		...

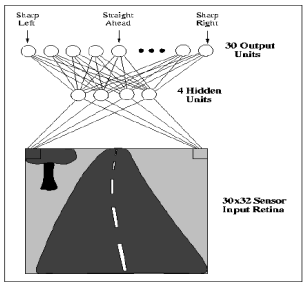
- Process optimization:

Product72: (time=t0)	*Product72:* (time=t1)	...	*Product72:* (time=tn)
Stage: mix	Stage: cook		Stage: cool
Mixing−speed: 60rpm	Temperature: 325		Fan−speed: medium
Viscosity: 1.3	Viscosity: 3.2		Viscosity: 1.3
Fat content: 15%	Fat content: 12%		Fat content: 12%
Density: 2.8	Density: 1.1		Density: 1.2
Spectral peak: 2800	Spectral peak: 3200		Spectral peak: 3100
Product underweight?: ??	Product underweight?: ??		**Product underweight?: Yes**
...	...		...

Problems Too Difficult to Program by Hand

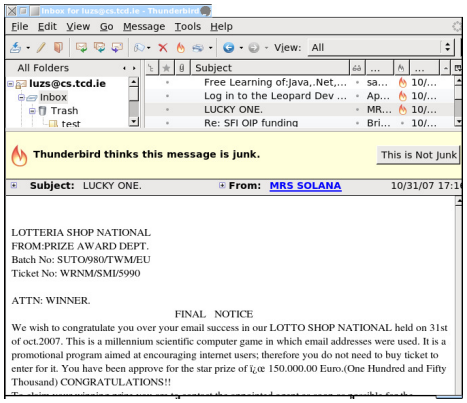
- ALVINN (Pomerleau, 1994): drives 70 mph





Software that adapts to its user

- Recommendation services,
- Bayes spam filtering
- etc



Perspectives

- Common applications
  - First-generation algorithms: neural nets, decision trees, regression ...

  - Applied to well-formated databases
- Advanced applications; areas of active reasearch:
  - Learn across full mixed-media data
  - Learn across multiple internal databases, plus the web and newsfeeds
  - Learn by active experimentation
  - Learn decisions rather than predictions
  - Cumulative, lifelong learning
  - Deep learning

Defining "learning"

- ML has been studied from various perspectives (AI, control theory, statistics, information theory, ...)
- From an AI perspective, the general definition is formulated in terms of agents and tasks. E.g.:

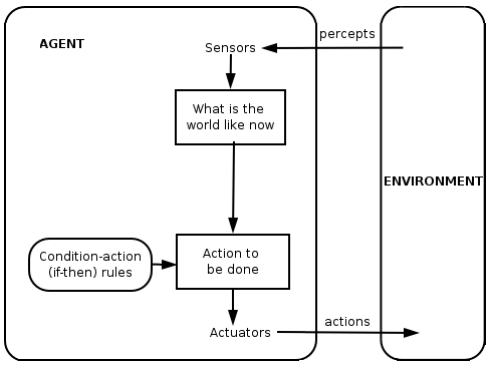
  > [An agent] is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with E. (Mitchell, 1997, p. 2)

- Statistics, model-fitting, ...

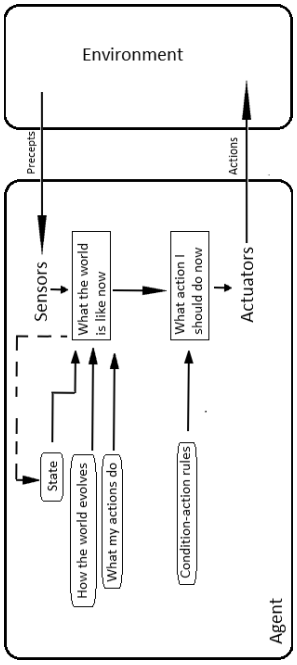
Agent Programs

- *Agent = architecture + program*
- Architecture = Sensors and Actuators
- Program = Decision Process
- Examples are: Simple Reflex Agents, Model-based reflex Agents, Goal-based Agents, Utility-based Agents and Learning (Intelligent) Agents.

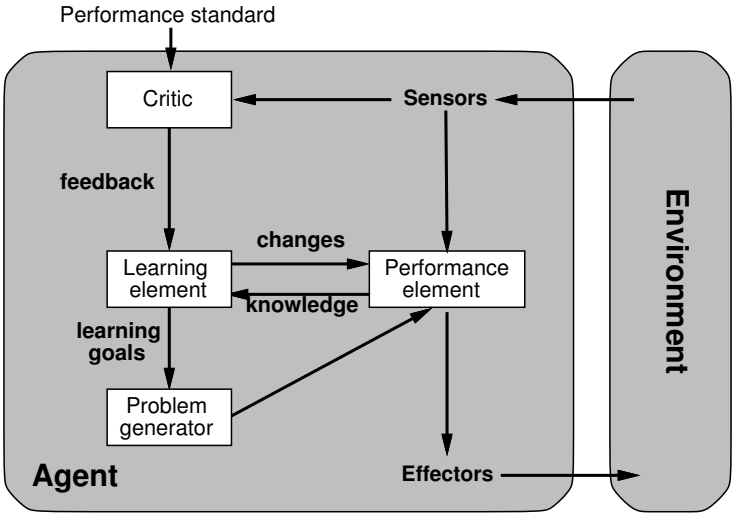
Simple Reflex Agent



Model-based Reflex Agent



Learning agents



The archiecture in some detail

- *Performance element*: responsible for selecting appropriate actions
- *Learning element*: responsible for making improvements

- *Critic*: evaluates action selection against a performance standard
- *Problem generator*: suggests actions that might lead to new and instructive experiences

Designing a machine learning system

- Main design decisions:
  - Training experience: How will the system access an use data?
  - Target function: What exactly should be learnt?
  - Hypothesis representation: How will we represent the concepts to be learnt?
  - Inductive inference: What specific algorithm should be used to learn the target concepts?

Accessing and using data

- How will the system be exposed to its training experience? Some distinctions:
  - Direct or indirect access:
    - indirect access: record of past experiences, corpora
    - direct access: situated agents → reinforcement learning
  - Source of feedback ("teacher"):
    - supervised learning
    - unsupervised learning
    - mixed: semi-supervised ("transductive"), active learning

Determining the target function

- The target function specifies the concept to be learnt.
- In supervised learning, the target function is assumed to be specified through annotation of training data or some form of feedback:
  - a corpus of words annotated for word senses, e.g. $f : W \times S \rightarrow \{0, 1\}$
  - a database of medical data
  - user feedback in spam filtering
  - assessment of outcomes of actions by a situated agent

Representing hypotheses and data

- The goal of the learning algorithm is to "induce" an approximation $\hat{f}$ of a target function f
- The data used in the induction process needs to be represented uniformly
  - E.g. representation of text as a "bag of words", Boolean vectors, etc

- The choice of representation often constrains the space of available hypotheses, hence the possible $\hat{f}$'s. E.g.:
  - the approximation to be learnt could, for instance, map conjunctions of Boolean literals to categories
  - or it could assume that co-occurence of words do not matter for categorisation
  - etc

Deduction and Induction

- Deduction (conclusion guaranteed): From general premises to a conclusion. E.g. If x = 4 And if y = 1, Then 2x + y = 9.
- Induction (conclusion likely): from instances to generalisations. E.g. All of the swans we have seen are white. Therefore, (we expect) all swans to be white.
- Machine learning algorithms produce models that generalise from *instances* presented to the algorithm
- But all (useful) learners have some form of *inductive bias*:
  - In terms of *representation*, as mentioned above,
  - But also in terms of their preferences in *generalisation procedures*. E.g:
    - prefer simpler hypotheses, or
    - prefer shorter hypotheses, or
    - incorporate domain (expert) knowledge, etc etc

Given an function $\hat{f} : X \to C$ trained on a set of instances D_c describing a concept c, we say that the inductive bias of $\hat{f}$ is a minimal set of assertions B, such that for any set of instanced X:

$$\forall x \in X(B \wedge D_c \wedge x \vdash \hat{f}(x))$$

Choosing an algorithm

- Induction task as *search* for a hypothesis (or model) that fits the data and sample of the target function available to the learner, in a large space of hypotheses
- The choice of learning algorithm is conditioned to the choice of representation
- Since the target function is not completely accessible to the learner, the algorithm needs to operate under the *inductive learning assumption* that:

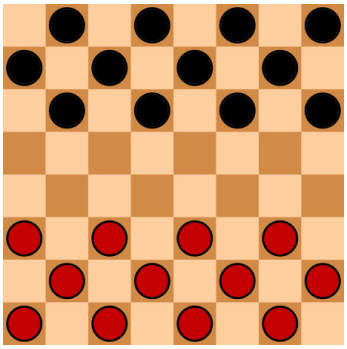
  > an approximation that *performs well* over a *sufficiently large set of instances* will perform well on unseen data.

- Note: *Computational Learning* Theory

Computational learning theory deals in a precise manner with the concepts highlighted above, namely, what it means for an approximation (learnt function) to perform well, and what counts as a sufficiently large set of instances. An influential framework is the probably approximately correct (PAC) learning framework, proposed by Valiant (1984). For an accessible introduction to several aspects of machine learning, see (Domingos, 2012). For some interesting implications see the "no-free lunch" theorems and the Extended Bayesian Framework (Wolpert, 1996).

An Example: learning to play (Mitchell, 1997)

- Learning to play draughts (checkers):



- Task? (target function, data representation) Training experience? Performance measure?

A target function

- A target function for a draughts (checkers) player:
  - $f : Board \rightarrow \mathbb{R}$
  - if b is a final board state that is *won*, then $f(b) = 100$
  - if b is a final board state that is *lost*, then $f(b) = -100$
  - if b is a final board state that is *drawn*, then $f(b) = 0$
  - if b is a *not a final state* in the game, then $f(b) = f(b')$, where b' is the *best final board state* that can be achieved starting from b and *playing optimally* until the end of the game.
- How feasible would it be to implement it?
- *Not very feasible...*
- ... and how can we find intermediate game states?

Representation

- collection of rules? neural network ? polynomial function of board features? ...
- Approximation as a linear combination of *features*:

$$\hat{f}(b) = w_0 + w_1 \cdot bp(b) + w_2 \cdot rp(b) + w_3 \cdot bk(b) + w_4 \cdot rk(b) + w_5 \cdot bt(b) + w_6 \cdot rt(b)$$

- where:
  - $bp(b)$: number of black pieces on board b
  - $rp(b)$: number of red pieces on b
  - $bk(b)$: number of black kings on b
  - $rk(b)$: number of red kings on b
  - $bt(b)$: number of red pieces threatened by black (i.e., which can be taken on black's next turn)
  - $rt(b)$: number of black pieces threatened by red

Training Experience

- Distinctions:
  - $f(b)$: the true target function
  - $\hat{f}(b)$: the learnt function
  - $f_{train}(b)$: the training value
  - A training set containing instances and its corresponding training values
- Problem: How do we estimate training values?
- A simple rule for estimating training values:
  - $f_{train}(b) \leftarrow \hat{f}(Successor(b))$
  - Successor(b) denotes the next board state following the programs move and the opponent's response.
  - Note: (Successor(b) is an 'estimation' of the value of board state b.
  - Does the $\hat{f}(b)$ tend to become more or less accurate for board states closer to the end of the game?

Example: Choosing a Function Approximation

- Learning the target function by approximation $\hat{f}(b)$
- Based on a set of training examples describing a board state b
- And the corresponding training value $f_{train}(b)$
- Each training example results in an ordered pair of the form $< b, f_{train}(b) >$
- Example: $<< bp = 3, rp = 0, bk = 1, rk = 0, bt = 0, rt = 0 >, +100 >$
- $f_{train}(b)$ is therefore +100 and black has won.

How do we learn the weights?

Algorithm 1: Least Mean Square

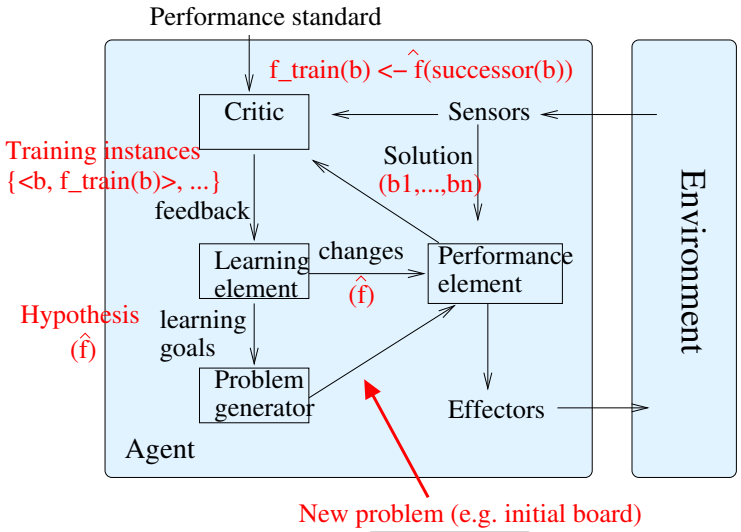
```
1  LMS(c: learning rate)
2    for each training instance < b, f_train(b) >
3      do
4        compute error(b) for current approximation
5        (using current weights):
6              error(b) = f_train(b) − f̂(b)
7        for each board feature t_i ∈ {bp(b), rp(b), ...},
8          do
9            update weight w_i:
10             w_i ← w_i + c × t_i × error(b)
11         done
12     done
```

LMS minimises the squared error between training data and current approx.: $E \equiv \sum_{\langle b, f_{train}(b)\rangle \in \mathcal{D}} (f_{train}(b) - \hat{f}(b))^2$ Notice that if $error(b) = 0$ (i.e. target and approximation match) no weights change. Similarly, if or $t_i = 0$ (i.e. feature t_i doesn't occcur) the corresponding weight doesn't get updated. This weight update rule can be shown to perform a gradient descent search for the minimal squared error (i.e. weight updates are proportional to $-\nabla E$ where $\nabla E = [\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots])$.

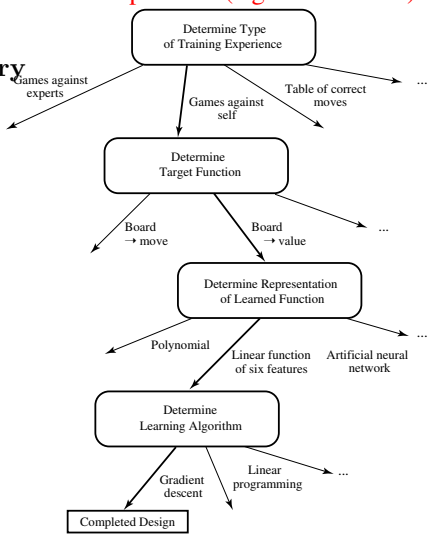
That the LMS weight update rule implements gradient descent can be seen by differentiating ∇E:

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial \sum [f(b) - \hat{f}(b)]^2}{\partial w_i} \\
&= \frac{\sum \partial [f(b) - \hat{f}(b)]^2}{\partial w_i} \\
&= \sum 2 \times [f(b) - \hat{f}(b)] \times \frac{\partial}{\partial w_i} [f(b) - \hat{f}(b)] \\
&= \sum 2 \times [f(b) - \hat{f}(b)] \times \frac{\partial}{\partial w_i} [f(b) - \sum_{i}^{|\mathcal{D}|} w_i t_i] \\
&= -\sum 2 \times error(b) \times t_i
\end{aligned}$$

Learning agent architecture



Design choices: summary



Mapping and structure

- Some target functions (specially in NLP) fit more naturally into a *transducer* pattern, and naturally have a signature

$$f\text{: sequence over vocab } \Sigma \Rightarrow \text{sequence over } (\Sigma \times \text{labels } \mathcal{C})$$

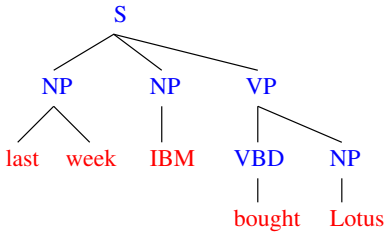
- eg. POS-tagging (Part-of Speech Tagging)

last week IBM bought Lotus ⇒ last/JJ week/NN IBM/NNP bought/VBD Lotus/NNP

Targeting Sequences and Trees

- other functions do not fit this pattern either, but instead have a signature

$$f\text{: sequence over vocab } \Sigma \Rightarrow \text{tree over } (\Sigma \cup \text{labels } \mathcal{C})$$



- eg. parsing: last week IBM bought Lotus⇒

Issues in machine learning

- What algorithms can approximate functions well (and when)?
- How does number of training examples influence accuracy?
- How does complexity of hypothesis representation impact it?
- How does noisy data influence accuracy?
- What are the theoretical limits of learnability?
- How can prior knowledge of learner help?
- What clues can we get from biological learning systems?
- How can systems alter their own representations?

Some application examples we will see in some detail

- Applications of Supervised learning in NLP:
  - Text categorisation
  - POS tagging (briefly)
  - Word-sense disambiguation (briefly)
- Unsupervised learning:
  - Keyword selection, feature set reduction
  - Word-sense disambiguation (revisited)

References

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Pomerleau, D. A. (1994). *Neural Network Perception for Mobile Robot Guidance*. Kluwer, Dordrecht, Netherlands.

Valiant, L. (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142.

Wolpert, D. H. (1996). The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390.

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. John Wiley & Sons.