Lecture 13
Introduction to Machine Learning: Part I

Rob Gaizauskas

Lecture Outline

- Introduction
  - What is Machine learning?
  - Some Applications of ML in General
  - Why Get Intelligent Agents to Learn?
- Forms of Learning
  - Supervised learning, Unsupervised Learning, Reinforcement Learning
- Supervised Learning
  - A Simple Supervised Learning Algorithm: Decision Tree Learning

- Reading: (Readings that begin with * are **mandatory**)
  - *Russell and Norvig (2010), Chapter 18 "Learning from Examples", sections 18.1-18.3, 18.6.1
  - T. Mitchell. *Machine Learning.* McGraw Hill, 1997.
  - I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, 2nd edition*, Morgan Kaufmann, San Francisco, 2005.

What is Machine Learning ?

- A possible definition:

  The study of how to design computer programs whose performance at some task improves through experience.

  Or, more precisely, (Mitchell, 1997):

  Definition: 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*

- Important question:
  - Are we only interested in *performance* of learning program at a specific task?
  - Or are we also interested in
    - discovering *human-comprehensible descriptions of patterns* in data? (*knowledge discovery*)
    - Reusabilty of what is learned in different task settings (important for genuinely intelligent agents)

Some Applications of Machine Learning

- Data mining: using historical data to improve decisions – increasingly important given explosion of electronic data. E.g.:
  - *Medicine:* medical records → medical knowledge
    - selecting best embyros from *in vitro* fertilisation based on 60 features of embryos and historical data on viability
  - *Business:* customer records → better business decisions
    - assessing credit-worthiness of loan applicants based on features of former borrowers and repayment outcomes
    - improving customer retention based on discovering patterns of features amongst loyal vs. defecting customers
  - *Agriculture* herd/crop records → better farming decisions
    - improving cull selection from dairy herds by data mining over database of 700 attributes of millions of cows
  - ...

Some Applications of Machine Learning

- Software applications we can't program by hand
  - autonomous driving
  - speech/character recognition
  - sonar signal classification
  - fingerprint classification
- Personalised/self-customising programs
  - Newsreader that learns user interests
  - Recommender/advertising systems tailored to learned user profile
- Computer Games
  - Heuristic evaluation functions for combinatorily explosive board games (chess, checkers, Othello, . . .)

Why Get Intelligent Agents to Learn?

- If design of an agent can be improved, why not program that improvement in in the first place?
- Three main reasons:
  1. Designers cannot anticipate all situations an agent my find itself in
     - E.g. a maze navigating robot must learn the layout of each new maze it enters
  2. Designers cannot anticipate all changes over time
     - E.g. a program designed to predict tomorrow's stock market prices must adapt to different market conditions
  3. Human designers may have no idea how to program a solution themselves
     - E.g. humans cannot program a computer to recognize faces of family members but learning algorithms can accomplish this task

Forms of Learning

- Any component of an agent can be improved by learning from data
- Improvements and techniques used depend on 4 major factors
  - Which component is to be improved
  - What prior knowledge the agent already has
  - What representation is used for the data and the component
  - What feedback is available to learn from

Components to be Learned

- Lecture 7 considered various agent designs.
- Components in these agents include:
  1. A direct mapping from conditions on the current state to actions (in reflex agents)
  2. A means to infer relevant properties of the world from the percept sequence (in model-based agents)
  3. Information about how the world evolves and about results of possible agent actions (in model-based agents)
  4. Utility information indicating desirability of world states (in utility-based agents)
  5. Action-value information indicating desirability of actions (in utility-based agents)
  6. Goals that describe classes of states whose achievement maximises the agent's utility (in utility-based agents)
- Each of these can be learned

Components to be Learned

- Taking the automated taxi example:
  1. A direct mapping from conditions on the current state to actions
     - Agent might learn condition action rule for when to brake from situations where instructor shouts break
  2. A means to infer relevant properties of the world from the percept sequence
     - Agent might learn to recognise buses from many images it is told contain buses
  3. Information about how the world evolves and about results of possible agent actions
     - Agent might learn effects of actions by trying them – e.g. braking on wet surfaces – and observing results
  4. Utility information indicating desirability of world states
     - Agent might learn component of utility function by when it fails to get a tip after driving wildly

Representation and Prior Knowledge

- Agent could use
  - Propositional logic
  - First order logic
  - Bayesian networks (a representation for probabilistic inference)

  to represent existing knowledge and knowledge to be learned
- Much of current learning focusses on learning where
  - Inputs take the form of a vector of attribute values
    - E.g. a loan applicant might be represented by a vector of values for attributes such as: gender, age, educational level, years in employment, house ownership status, current debt, etc.
  - Outputs are either
    - Continuous numerical values
    - Discrete values
    - E.g. decision about credit-worthiness might be a binary variable ("yes","no")

Feedback to Learn From

Three main types of learning:

- Supervised learning
  - Agent observes some input-output pairs and learns a function that maps from input to output
  - For example:
    - For component 1 above, inputs are percepts and outputs are provided by the teacher saying "brake!"
    - For component 2 above, inputs are images are outputs come from a teacher who says "bus" when the input images contain a bus
    - For component 3 above, inputs are pairs of environment states and actions and outputs are stopping distances in, e.g. meter. Here output is available from the agent's percepts – the environment is the teacher
- Supervised learning has received the greatest attention/ interest in machine learning over the past 10+ years

Feedback to Learn From

Three main types of learning:

- Unsupervised learning
  - Agent learns patterns in the input even though no explicit feedback is given
  - Common example is clustering – detecting potentially useful clusters in input examples
    - E.g. taxi agent might develop a concept of "good traffic days" and "bad traffic days" without being given labelled examples by a teacher

Feedback to Learn From

Three main types of learning:

- Reinforcement learning
  - Agent learns from series of reinforcements – rewards or punishments
  - E.g. receipt of a tip at the end of a journey, or winning or losing at chess, tell the agent it has done something right or wrong
  - Challenge for the agent is to decide which actions prior to the reinforcement were responsible for it (called the credit assignment problem)

Supervised Learning: Learning Input-Output Functions

- In supervised learning we are typically trying to learn a function *f* from examples
- *f* is referred to as the target function
- *f* takes a vector-valued input, an n-tuple $\mathbf{x} = (x_1, x_2 \ldots x_n)$
- *f* yields a vector-valued *k*-tuple as output, though often $k = 1$, i.e. *f* produces a single output value

Input Vectors: Example

- A loan applicant Fred might be an instance to a learner represented by the 6-tuple:

  (m, 27, 1, 3, u, 24376)

  where the attributes are *gender*, *age*, *in-work*, *years-at-current-job*, *education-code*, *salary*
  - *gender* is nominal (values *m* or *f*)
  - *age* is ordinal (discrete-valued number)
  - *in-work* is nominal – a special case of a boolean-valued attribute
  - *years-at-current-job* is ordinal (discrete-valued number)
  - *education-code* is nominal (some fixed set of code values, e.g. *u* = undergraduate degree
  - *salary* is ordinal (discrete-valued number)

Outputs: Example

- In the loans applicant example, the learned hypothesis could be
  - a function estimator if the output is a real-number approximating the probability of Fred defaulting
  - a classifier if the output is a boolean 1 or 0 indicating whether Fred should be given a loan or not

A Simple Supervised Learning Algorithm: Decision Tree Learning

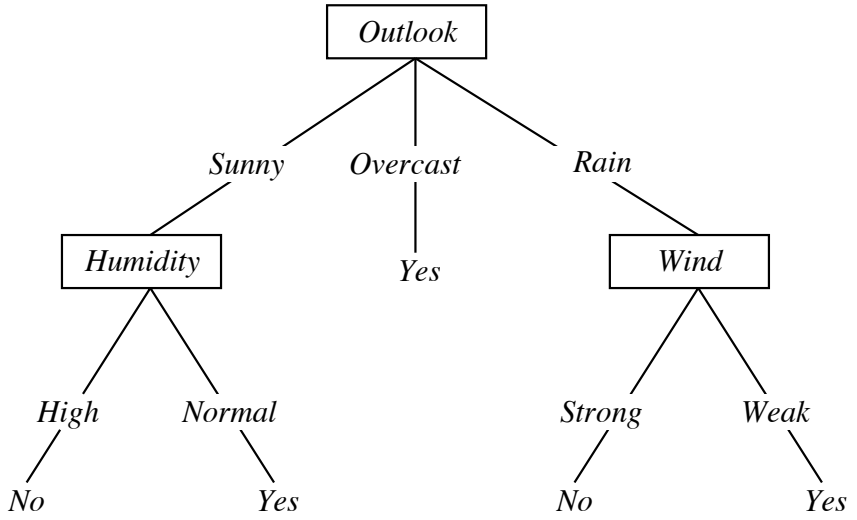
- Suppose we
  - want to learn to predict days on which Fred plays tennis
  - have a training dataset of 14 days of observations of weather + Fred's tennis playing behaviour
- One way we can do this is by decision tree learning

Attribute to be predicted

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision Tree Learning

- Decision trees are trees which classify instances by testing at each node some attribute of the instance.
- Testing starts at the root node and proceeds downwards to a leaf node, which indicates the classification of the instance.
- Each branch leading out of a node corresponds to a value of the attribute being tested at that node.
- A decision tree to classify days as appropriate for playing tennis might look like:



- The instance ⟨Outlook=Sunny,Temp=Hot,Humidity=High,Wind=Strong⟩ is classified: No

Decision Tree Learning

- Note that
  - each path through a decision tree forms a conjunction of attribute tests
  - the tree as a whole forms a disjunction of such paths; i.e. a disjunction of conjunctions of attribute tests
- E.g. preceding example could be re-expressed in logical notation as:

$$(Outlook = Sunny \wedge Humidity = Normal)$$
$$\vee\ (Outlook = Overcast)$$
$$\vee\ (Outlook = Rain \wedge Wind = Weak)$$

- Such a decision tree could be coded by hand.
- Challenge for machine learning is to propose algorithms for learning decision trees from examples.

An Algorithm for Decision Tree Learning

- Want to learn a decision tree from examples that is:
  - Consistent with all the examples
  - As small as possible
- Exhaustively searching for such a tree is intractable – number of possible trees even for *n* boolean-valued attributes > 2^{2^n}
- So need good, approximate solution – small, but perhaps not smallest consistent tree
- Standard decision tree learning algorithm proceeds by:
  - Determining "most important" attribute to test first
  - Making that attribute the root of the tree
  - Calling the algorithm recursively on the subsets of examples as determined by the root attribute values
- Note: not all datasets permit a consistent tree

Decision Tree Algorithm

function DECISION-TREE-LEARNING($examples, attributes, parent\_examples$) **returns** a tree

if $examples$ is empty **then return** PLURALITY-VALUE($parent\_examples$)
else if all $examples$ have the same classification **then return** the classification
else if $attributes$ is empty **then return** PLURALITY-VALUE($examples$)
else
  $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE($a, examples$)
  $tree \leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e : e \in examples$ **and** $e.A = v_k\}$
    $subtree \leftarrow$ DECISION-TREE-LEARNING($exs, attributes - A, examples$)
    add a branch to $tree$ with label $(A = v_k)$ and subtree $subtree$
  return $tree$

- The function plurality-value returns the most common output value in a set of examples, breaking ties randomly
- How do we choose the "most important" attribute?

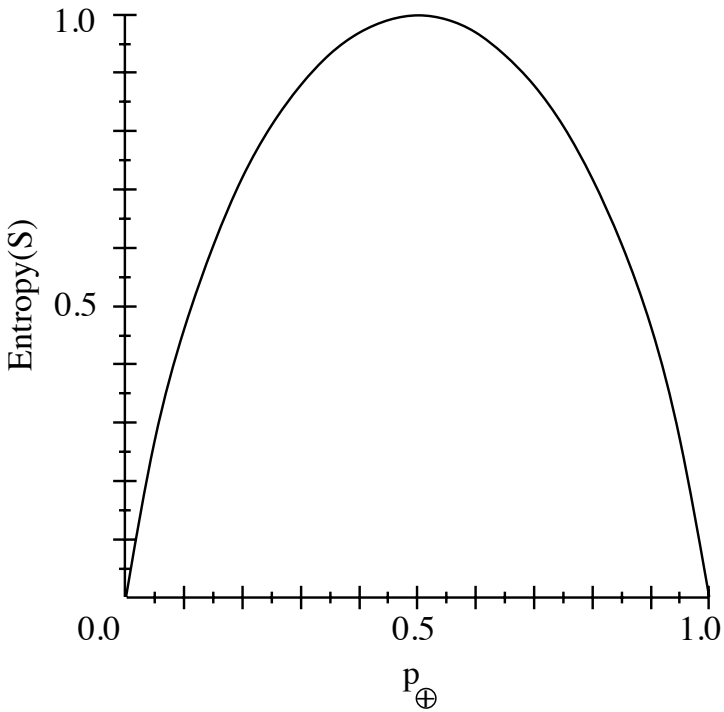
Choosing the "Most Important" Attribute

- Want to choose at each step an attribute that goes as far as possible towards providing an exact classification of the examples.
  - "perfect" attribute would divide examples into sets, each of which was "pure" – i.e. consisted entirely of either all positive or all negative examples
- A good measure is information gain, which is defined in terms of entropy, a fundamental quantity in information theory

Entropy

- Suppose we have a set of training examples *S* with a boolean valued target class
  - like *PlayTennis* in our dataset
- Let
  - $p_{\oplus}$ be the proportion of positive examples in *S*
  - $p_{\ominus}$ be the proportion of negative examples in *S*
- Then Entropy(S) is defined as:

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$



- The more "impure" (balanced mixture of positives and negatives) *S* is, the greater the entropy

Information Gain

- Information Gain is the expected reduction in entropy resulting from partitioning a set of examples on the basis of an attribute.
- Formally the information gain of a set of training examples S with respect to an attribute *A* is:

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

- In decision tree learning pick the attribute with highest information gain first