Lecture 1:

Probability & Uncertainty in Artificial Intelligence

January 9, 2006

Intelligent Computers

- We want intelligent, adaptive, robust behaviour.

⇒ Sam Roweis

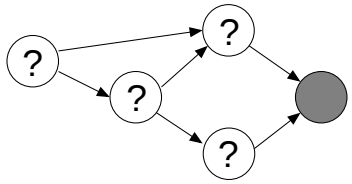
- Often hand programming not possible.
- Solution? Get the computer to program itself, by showing it examples of the behaviour we want!
  This is the *learning* approach to AI.
- Really, we write the structure of the program and the computer tunes many internal parameters.

Core vs. Probabilistic AI

- KR: work with facts/assertions; develop rules of logical inference
- Planning: work with applicability/effects of actions; develop searches for actions which achieve goals/avert disasters.
- Expert Systems: develop by hand a set of rules for examining inputs, updating internal states and generating outputs
- Learning approach: use probabilistic models to tune performance based on many data examples.
- Probabilistic AI: emphasis on noisy measurements, approximation in hard cases, learning, algorithmic issues.
  logical assertions ⇒ probability distributions
  logical inference ⇒ conditional probability distributions
  logical operators ⇒ probabilistic generative models

The Power of Learning

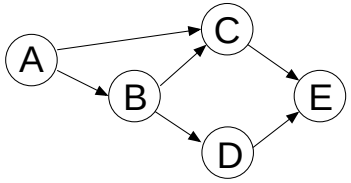
- Probabilistic Databases
  - traditional DB technology cannot answer queries about items that were never loaded into the dataset
  - UAI models are like probabilistic databases



- Automatic System Building
  - old expert systems needed hand coding of knowledge and of output semantics
  - learning automatically constructs rules and supports all types of queries

Uncertainty and Artificial Intelligence (UAI) 4

- Probabilistic methods can be used to:
  - make decisions given partial information about the world
  - account for noisy sensors or actuators
  - explain phenomena not part of our models
  - describe inherently stochastic behaviour in the world



- Example: you live in California with your spouse and two kids. You listen to the radio on your dirve home, and when you arrive you find your burglar alarm ringing.
  Do you think your house was broken into?

Other Names for UAI 5

- Machine learning, data mining, applied statistics, adaptive (stochastic) signal processing, probabilistic planning/reasoning...
- Some differences:
  - Data mining almost always uses large data sets, statistics almost always small ones.
  - Data mining, planning, decision theory often have no internal parameters to be learned.
  - Statistics often has no algorithm to run!
  - ML/UAI algorithms are rarely online and rarely scale to huge data (changing now).
- Learning is most useful when the structure of the task is not well understood but can be characterized by a dataset with strong statistical regularity. Also useful in adaptive or dynamic situations when the task (or its parameters) are constantly changing.

Applications of Probabilistic Learning 6

- Automatic speech recognition & speaker verification
- Printed and handwritten text parsing
- Face location and identification
- Tracking/separating objects in video
- Search and recommendation (e.g. `google`, `amazon`)
- Financial prediction, fraud detection (e.g. credit cards)
- Insurance premium prediction, product pricing
- Medical diagnosis/image analysis (e.g. pneumonia, pap smears)
- Game playing (e.g. backgammon)
- Scientific analysis/data visualization (e.g. galaxy classification)
- Analysis/control of complex systems (e.g. freeway traffic, industrial manufacturing plants, space shuttle)
- Troubleshooting and fault correction

Related Areas of Study 7

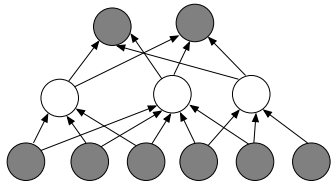
- Adaptive data compression/coding:
  – state of the art methods for image compression and error correcting codes all use learning methods
- Stochastic signal processing:
  – denoising, source separation, scene analysis, morphing
- Decision making, planning:
  – use both utility and uncertainty optimally, e.g. influence diagrams
- Adaptive software agents / auctions / preferences – action choice under limited resources and reward signals

Canonical Tasks

- *Supervised Learning*: given examples of inputs and corresponding desired outputs, predict outputs on future inputs.
  Ex: classification, regression, time series prediction
- *Unsupervised Learning*: given only inputs, automatically discover representations, features, structure, etc.
  Ex: clustering, outlier detection, compression
- *Rule Learning*: given multiple measurements, discover very common joint settings of subsets of measurements.
- *Reinforcement Learning*: given sequences of inputs, actions from a fixed set, and scalar rewards/punishments, learn to select action sequences in a way that maximizes expected reward.
  [Last two will not be covered in this course.]

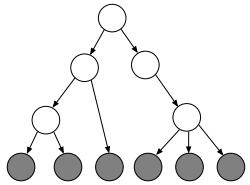
Supervised Learning

- Classification: outputs are categorical, inputs are anything.
  Goal is to select correct class for new inputs.
- Regression: outputs are continuous, inputs are anything (but usually continuous).
  Goal is to predict outputs accurately for new inputs.
- Prediction: data are time series.
  Goal is to predict on new sequences values at future time points given values at previous time points.



Unsupervised Learning

- Clustering: inputs are vector or categorical.
  Goal is to group data cases into a finite number of clusters so that within each cluster all cases have very similar inputs.
- Outlier detection: inputs are anything.
  Goal is to select highly unusual cases from new *and* given data.
- Compression/Vector Quantization: inputs are generally vector.
  Goal is to deliver an *encoder* and *decoder* such that size of encoder output is much smaller than original input but composition of encoder followed by decoder is very similar to the original input.



Representation

- Key issue: how do we represent information about the world?
  (e.g. for an image, do we just list pixel values in some order?)

$\rightarrow$ 127,254,3,18,...

- We must pick a way of numerically representing things that exploits regularities or structure in the data.
- To do this, we will rely on probability and statistics, and in particular on *random variables*.
- A *random variable* is like a variable in a computer program that represents a certain quantity, but its value changes depending on which data our program is looking at. The value a random variables is often unknown/uncertain, so we use probabilities.

Using random variables to represent the world 12

- We will use mathematical random variables to encode everything we know about the task: inputs, outputs and internal states.
- Random variables may be *discrete/categorical* or *continuous/vector*.
  Discrete quantities take on one of a fixed set of values,
  e.g. {`0,1`}, {`email,spam`}, {`sunny,overcast,raining`}.
  Continuous quantities take on real values.
  e.g. `temp=12.2, income=38231, blood-pressure=58.9`
- Generally have repeated measurements of same quantities.
  Convention: $i, j, \ldots$ indexes components/variables/dimensions;
  $n, m, \ldots$ indexes cases/records, x are inputs, y are outputs.
  – x_i^n is the value of the ith input variable on the nth case
  – y_j^m is the value of the jth output variable on the mth case
  $\mathbf{x}_n$ is a vector of all inputs for the nth case
  $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n, \ldots, \mathbf{x}_N\}$ are all the inputs

Structure of Learning Machines 13

- Given some inputs, expressed in our representation, how do we calculate something about them (e.g. this is Sam's face)?
- Our computer program uses a *mathematical function* $\hat{\mathbf{y}} = f(\mathbf{x})$
  x is the representation of our input (e.g. face)
  z is the representation of our output (e.g. Sam)
- Hypothesis Space and Parameters:
  We don't just make up functions out of thin air. We select them from a carefully specified set, known as our *hypothesis space*.
- Generally this space is indexed by a set of *parameters* θ which are knobs we can turn to create different machines:
  $\mathcal{H} : \{f(\hat{\mathbf{y}}|\mathbf{x}, \theta)\}$
- Hardest part of doing probabilistic learning is deciding how to represent inputs/outputs and how to select hypothesis spaces.

Loss Functions for Tuning Parameters 14

- Let inputs=$\mathbf{X}$, correct answers=$\mathbf{Y}$, outputs of our machine=$\hat{\mathbf{Y}}$.
- Once we select a representation and hypothesis space, how do we set our parameters θ?
- We need to quantify what it means to do well or poorly on a task.
- We can do this by defining a loss function $L(\mathbf{X}, \mathbf{Y}, \hat{\mathbf{Y}})$ (or just $L(\mathbf{X}, \hat{\mathbf{Y}})$ in unsupervised case).
- Examples:
  Classification: $\hat{y}_n(\mathbf{x}_n)$ is predicted class. $L = \sum_n [y_n \neq \hat{y}_n(\mathbf{x}_n)]$

  Regression: $\hat{\mathbf{y}}_n(\mathbf{x}_n)$ is predicted output. $L = \sum_n \|\mathbf{y}_n - \hat{\mathbf{y}}_n(\mathbf{x}_n)\|^2$

  Clustering: μ_c is mean of all cases assigned to cluster c.
  $$L = \sum_n \min_c \|\mathbf{x}_n - \mu_c\|^2$$
- Now set parameters to minimize average loss function.

A More Formal Setup for Loss 15

- Cast machine learning tasks as *numerical optimization problems*.
- Quantify how well the machine pleases us by a scalar *objective function* which we can evaluate on sets of inputs/outputs.
- Represent given inputs/outputs as arguments to this function.
- Also introduce a set of unknown parameters θ which are also arguments of the objective function.
- Goal: adjust unknown parameters to minimize objective function given inputs/outputs.
  $$\arg\min_{\theta} \Phi(\mathbf{X}, \mathbf{Y}|\theta)$$
- The *art* of designing a machine learning system is to select the numerical representation of the inputs/outputs and the mathematical formulation of the task as an objective function.
- The *mechanics* involve optimizing the objective function given the observed data to find the best parameters. (Often leads to art!)

A More General Objective Function

- The general structure of the objective function is:

$$\Phi(\mathbf{X}, \theta) = L(\mathbf{X}|\theta) + P(\theta)$$

- L is the loss function, and P is a penalty function which penalizes more complex models.
- This says that it is good to fit the data well (get low training loss) but it is also good to bias ourselves towards simpler models to avoid overfitting.

Training vs. Testing

- *Training data*: the $\mathbf{X}, \mathbf{Y}$ we are given.
  Testing data: the $\mathbf{X}, \mathbf{Y}$ we will see in future.
- *Training error*: the average value of loss on the training data.
  Test error: the average value of loss on the test data.
- What is our real goal? To do well on the data we have seen already? Usually not. We already have the answers for that data. We want to perform well on *future unseen data*. So ideally we would like to minimize the test error. How to do this if we don't have test data?
- Probabilistic framework to the rescue!

Sampling Assumption

- Imagine that our data is created randomly, from a joint probability distribution $p(\mathbf{x}, \mathbf{y})$ which we don't know.
- We are given a finite (possibly noisy) training sample: $\{\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_n, \mathbf{y}_n, \ldots, \mathbf{x}_N \mathbf{y}_N\}$ with members n generated *independently and identically distributed* (iid).
- Looking only at the training data, we construct a machine that generates outputs $\hat{\mathbf{y}}$ given inputs. (Possibly by trying to build machines with small training error.)
- Now a new sample is drawn from the same distribution as the training sample.
- We run our machine on the new sample and evaluate the loss; this is the test error.
- Central question: by looking at the machine, the training data and the training error, what if anything can be said about test error?

Generalization and Overfitting

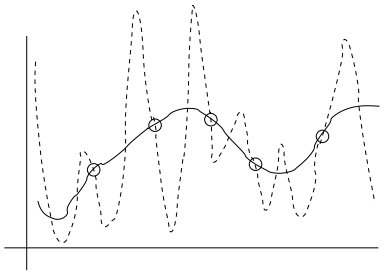
- Crucial concepts: *generalization*, *capacity*, *overfitting*.
- What's the danger in the above setup? That we will do well on training data but poorly on test data. This is called *overfitting*.
- Example: just memorize training data and give random outputs on all other data.
- Key idea: you can't learn anything about the world without making some assumptions.
  (Although you can memorize what you have seen).
- Both representation and hypothesis class (model choice) represent assumptions we make.
- The ability to achieve small loss on test data is *generalization*.

Capacity: Complexity of Hypothesis Space

- Learning == Search in Hypothesis Space
- Inductive Learning Hypothesis: Generalization is possible.
  If a machine performs well on most training data AND it is not too complex, it will probably do well on similar test data.
- Amazing fact: in many cases this can actually be *proven*. In other words, if our hypothesis space is not too complicated/flexible (has a low *capacity* in some formal sense), and if our training set is large enough then we can bound the probability of performing much worse on test data than on training data.
- The above statement is carefully formalized in 20 years of research in the area of *learning theory*.

Inductive Bias

- The converse of the Inductive Learning Hypothesis is that generalization only possible if we make some assumptions, or introduce some priors. We need an *Inductive Bias*.
- No Free Lunch Theorems: an unbiased learner can never generalize.
- Consider: arbitrarily wiggly functions or random truth tables or non-smooth distributions.



0 0 0	0
0 0 1	?
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	?
1 1 0	1
1 1 1	?

Probabilistic Approach

- Given the above setup, we can think of learning as estimation of joint probability density functions given samples from the functions.
- Classification and Regression: conditional density estimation $p(\mathbf{y}|\mathbf{x})$
- Unsupervised Learning: density estimation $p(\mathbf{x})$
- *The central object of interest is the joint distribution and the main difficulty is compactly representing it and robustly learning its shape given noisy samples.*
- *Our model of the world (inductive bias) is expresses as prior assumptions about these joint distributions.*
- *The main computations we will need to do during the operation of our algorithms are to efficiently calculate marginal and conditional distributions from our compactly represented joint model.*

In this course

- Using probabilities to represent beliefs about all aspects of a problem, including inputs, outputs and internal states.
- "Probabilistic Graphical Models" as structured representations of large probability distributions.
- Statistical parameter estimation for simple classification, regression and density models.
- Junction tree algorithm for inference of "hidden/latent variables".
- EM algorithm for general parameter learning in latent variable models.
- Function approximation with linear regression, artificial neural networks, mixtures of experts.
- Classification by nearest neighbour, logistic regression, neural nets.
- Clustering and dimensionality reduction using k-means, mixture models, factor analysis, PCA, HMMs.

Unanswered Questions

- Given a task, how do we formulate it as function approximation?
- How to choose/learn representations?
- How select/partition training/testing data?
- How much time/space do we need (computation cost)?
- Can we prove convergence of our algorithms?
- How much training input do we need (data cost)?
- Can we ever be assured (or almost assured) of success?
- How to engineer what we know about problem structure and incorporate prior/domain/expert knowledge?

General Reading

- Journals: Neural Computation, JMLR, ML, IEEE PAMI
- Conferences: NIPS, UAI, ICML, AI-STATS, IJCAI, IJCNN
- Speech: EuroSpeech, ICSLP, ICASSP
- Vision: CVPR, ECCV, SIGGRAPH
- Online: citeseer, google scholar, rexa.info
- Books:
  - *Introduction to Probabilistic Graphical Models*, Jordan
  - *Information Theory, Inference & Learning Algorithms*, Mackay
  - *Elements of Statistical Learning*, Hastie, Tibshirani, Friedman
  - *Probabilistic Reasoning in Intelligent Systems*, Pearl
  - *Neural Networks for Pattern Recognition*, Bishop
  - *Pattern Recognition and Neural Networks*, Ripley