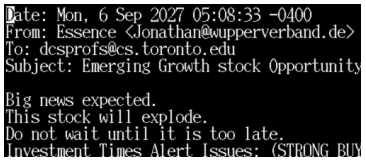
Lecture 1:

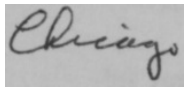
Basic Machine Learning Concepts

September 12, 2006

Intelligent Computers

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





Applications

Machine 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. ML is also useful in adaptive or dyanmic situations when the task (or its parameters) are constantly changing.

- Spam filtering, fraud detection (e.g. credit cards, phone calls)
- Search and recommendation (e.g. `google`, `amazon`)
- Automatic speech recognition & speaker verification
- Printed and handwritten text parsing
- Locating/tracking/identifying objects in images & video (e.g. faces)
- Financial prediction, pricing, volatility analysis
- Medical diagnosis/image analysis (e.g. pneumonia, pap smears)
- Driving computer players in games
- Scientific analysis/data visualization (e.g. galaxy classification)

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
- Because ML is concerned with learning from data examples, it often uses a probabilistic approach.
- Probabilistic AI: emphasis on noisy measurements, approximation in hard cases, learning, algorithmic issues.
  logical assertions $\Rightarrow$ probability distributions
  logical inference $\Rightarrow$ conditional probability distributions
  logical operators $\Rightarrow$ probabilistic generative models

Other Names for ML

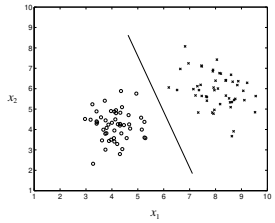
- Data mining, applied statistics, adaptive/stochastic signal processing, probabilistic planning/reasoning are all closely related to machine learning.
- Some differences:
  - Data mining almost always uses large data sets, statistics almost always small ones.
  - Data mining, planning, decision theory often have no internal parameters to be learned.
  - Statistics often has no algorithm to run!
  - ML algorithms are rarely online and rarely scale to huge data (changing now).

Canonical Tasks

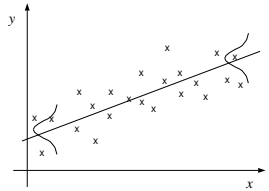
- *Supervised Learning*: given examples of inputs and corresponding desired outputs, predict outputs on future inputs.
  Ex: classification, regression, time series prediction
- *Unsupervised Learning*: given only inputs, automatically discover representations, features, structure, etc.
  Ex: clustering, 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.
  [That's the last you will hear of this in this course.]

Supervised Learning

- *Classification*: Outputs are categorical, inputs are anything. Goal is to select correct class for new inputs.



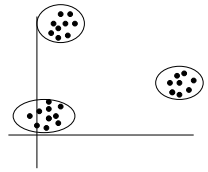
- *Regression*: outputs are continuous, inputs are anything (but usually continuous). Goal is to predict outputs accurately for new inputs.



- *Prediction*: data are time series. Goal is to perform classification/regression on new input sequences values at future time points given input values and coresponding class labels/outputs at some 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.
  ...almost the same as:



- *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.
- *Outlier detection*: inputs are anything. Goal is to select highly unusual cases from new *and* given data.
- *Rule Learning*: inputs are mixed categorical and vector. Goal is to find rules of the form:
  $(x_1 < a) \wedge (x_2 > b) \wedge \ldots \wedge (x_k = c) \Rightarrow (x_0 = d)$
  which are compact (few terms), highly accurate, and very common.

Using Variables to Represent the World

- We will use mathematical variables to encode everything we know about the task: inputs, outputs and internal states.
- 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. `1.6632`, `[3.3,-1.8,120.4]`
- Generally have repeated measurements of same quantities.
  Convention: $i, j, \ldots$ indexes components/variables/dimensions;
  $n, m, \ldots$ indexes cases/records.
  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

Hypothesis Space and Parameters

- Now that we know how to represent our inputs and outputs, how should we represent our learning machines?
  You guessed it: as *functions*.
- Q: How do we construct these functions?
  (Considering there are an uncountably infinite number of them!)
  A: 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:

$$\mathcal{H} : \{f(\mathbf{x}, \mathbf{y}|\theta)\}$$

- The hardest part of machine learning is deciding how to represent inputs and outputs and how to select the hypothesis space.
- After that, we have to decide how to set the parameters...

Loss Functions

- We need to quantify what it means to do well or poorly on a task.
- We can do this by defining a loss (error) function $L(\mathbf{X}, \mathbf{Y}, \theta)$ (or just $L(\mathbf{X}, \theta)$ in unsupervised case).
- Examples:

  Classification: $\hat{y}_n(\mathbf{x}_n, \theta)$ is predicted class. $L = \sum_n [\hat{y}_n \neq y_n]$

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

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

  Rules: R_k is the kth rule.
  $L = \sum_k [\text{support}(R_k) > a][\text{confidence}(R_k) > c]$

Training vs. Testing

- *Training data*: the $\mathbf{X}, \mathbf{Y}$ we are given now.
  Testing data: the $\mathbf{X}, \mathbf{Y}$ we will see in future.
- *Training error*: the value of loss on the training data.
  Test error: the value of loss on the test data.
- What is our real goal?
  To do well on the training data? On the testing data? On both?
  How do we decide which one is more important?
- We often want to perform well on *future unseen data*.
  So ideally we would like to minimize the *test* error.
- Q: How can we optimize for this if we don't have the test data yet?
  A: Probabilistic framework to the rescue!

Sampling Assumption

- Imagine the there exists a joint probability distribution $p(\mathbf{x}, \mathbf{y})$, which we don't know, over inputs&outputs (or just inputs).
- 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 select a parameter θ (possibly by trying to minimize the loss). The resulting hypothesis has some training error (loss) $\sum_n L(\mathbf{x}_n, \mathbf{y}_n, \theta)$.
- 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?

* (Careful: with or without replacement ?)

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*.

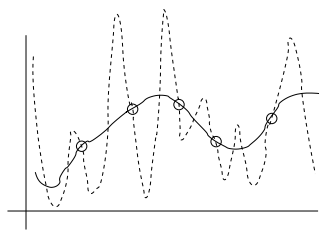
  A hypothesis which achieves low training error will not necessarily achieve low test error unless it came from a very simple class compared to the amount of data used to select it.
- Example:
  – memorize the training data
  – give random outputs on all future data
- The ability to achieve small loss on test data is *generalization*.
- Generalization is often (but not always these days) the goal of machine learning.

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 acutally 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*.
- Basically two main camps: VC-dimension and PAC.
  There are also links to MDL and Bayesian theory, Occam's Razor.
  (see CSC2614, Prof. Toni Pitassi)

Inductive Bias is Needed

- 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
(non-smooth distributions, ...)

Input $\mathbf{X}$	Output $\mathbf{Y}$
000	1
001	1
010	?
011	0
100	0
101	?
110	1
111	0

unstructured truth tables

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 $p(\mathbf{x})$
- Clustering/Rule Learning: finding small regions with high $p(\mathbf{x})$. (bump hunting)

The central object of interest is the joint distribution over inputs&outputs and the main difficulty is compactly representing it and robustly learning its shape given noisy samples.

Our inductive bias expresses our prior assumptions about these joint distributions and without it we cannot learn anything.

Formal Setup

- Cast machine learning tasks as *numerical optimization problems*.
- Quantify how good a machine is by a scalar *objective function* Φ 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 additional 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!)

General Objective Functions

- How should we design the objective function?
  Clearly, it should have something to do with the loss function!
- The general structure of the objective function is:

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

- θ are the parameters we chose
  (which index a member of the hypothesis class)
  L is the loss function
  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, in order to avoid overfitting.

Questions, 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)?
- How much training input do we need (data cost)?
- Can we prove convergence (termination) of our algorithms?
- 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?

- Conferences: NIPS, UAI, ICML, AI-STATS
- Journals: JMLR, Neural Comp., JAIR, MLJ, IEEE PAMI
- Vision/Graphics: CVPR, ICCV, ECCV, SIGGRAPH
- Speech: EuroSpeech, ICSLP, ICASSP
- Online: citeseer, google
- Books:
  - *Elements of Statistical Learning*, Hastie, Tibshirani, Friedman
  - *Pattern Recognition and Neural Networks*, Ripley
  - *Introduction to Graphical Models*, Jordan et. al (unpublished)
  - *Neural Networks for Pattern Recognition*, Bishop [dated]
  - *Machine Learning*, Mitchell [very dated]