ECS 607/766 – Data Mining
Lecture 10 – Advanced Topics

Dr. Ioannis Patras
EECS, Queen Mary University of London

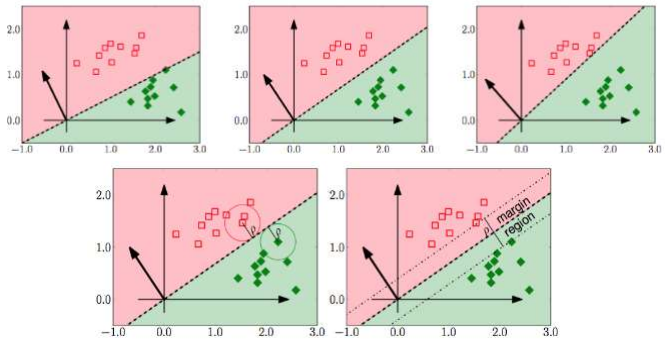
Slide thanks: Tim Hospedales

Outline

- Some Advanced Models
  - Support Vector Machines
  - Neural Networks
- Structured Modeling:
  - Time-series modeling
- Special Settings: Beyond Supervised Learning:
  - Semi-supervised learning
  - Active Learning

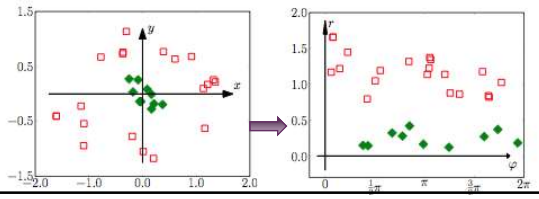
Support Vector Machines (1/2)

- Basic SVMs are linear classifier/regressors. $y = \mathbf{w}^T \mathbf{x}_i > 0$
  - Trick is they have a good objective function (hinge loss) compared to standard logistic/maxent or MSE objective.
  - This makes them find models more likely to generalise.

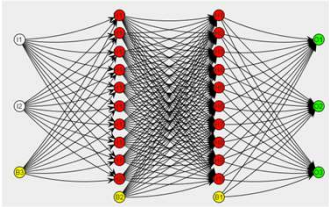


Support Vector Machines (2/2)

- SVMs are very well analysed theoretically.
- Non-linear support vector machines have interesting properties
  - Still linear, but in a new space
  - New space is a non-linear transform where data is now separable
  - Effectively an arbitrarily complex non-linear decision boundary in old space
  - Can classify/regress very complicated data with good generalisation
  - Typical cost: Train: $O(DN^3)$, Test: $O(DS)$.

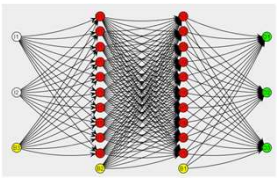


Neural Networks (1/3)



- Neural Networks
  - Loosely inspired by human brain function
  - Have a long history (since 70s and before)
  - Entire network composed of many neurons
- Each neuron *i* is simple:
  - Reads inputs (either data, or outputs of neurons in earlier layer)
  - It multiplies its inputs by some weights to get its output, $y_i=f(w_i^T x)$.
    - Cf Linear classifier/regressor.
  - But output of one neuron is input to next layer.
- Data propagates through the whole network to get output
  - Each neuron does its computation and sends output to next one.

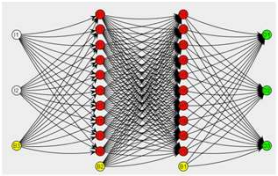
Neural Networks (2/3)



Interesting Properties:

- Linearity
  - If each neuron is simple $y=w^T x$, nothing interesting happens.
    - (Network is no different to MaxEnt/Linear Regression in the end)
  - If each neuron has a non-linearity $y=f(w^T x)$, a large network can learn an arbitrarily complex function.
- Testing: Easy
- Training:
  - Supervised training as usual: Find ws to maximize accuracy.
  - Similarly to Linear regression & linear classifiers, can train all the weights w with gradient.
  - But differentiating re: w, reveals each neuron's optimal weights dependent on the others.
  - Leads to famous backpropagation algorithm, which is large scale application of the chain rule of differentiation.

Neural Networks (3/3)



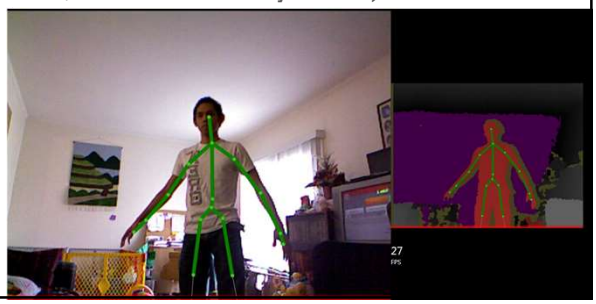
- Neural Networks were popular around 70s-90s.
  - Fell out of favor around 90s-2010 because:
    1. Opaque to understand and analyze prove/theoretical guarantees
    2. SVMs performed better in practice
    3. Large networks have very many parameters
    4. => Overfitting is easy, Backpropagation is slow, Leads to often bad local minima.
- They are back now (2012-) because:
  - A few new training tricks to reduce overfitting & get better local minima.
  - Very many parameter issues addressed by:
    - "Big Data" can still provide n>d to constrain all those parameters. Even when parameters, d=100Mil+.
    - GPU-based computation can accelerate training all those parameters.
      - E.g., Typically months to CPU train a network, days GPU-network.
  - Architecture design is still ~black magic
  - Still no theoretical understanding/guarantees, but excellent in practice.

Outline

- Some Advanced Models
  - Support Vector Machines
  - Neural Networks
- Structured Modeling
  - Time-series modeling
- Special Settings
  - Semi-supervised learning
  - Active Learning

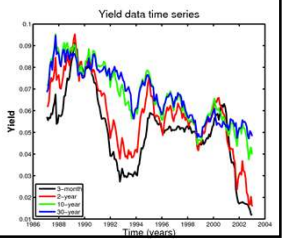
Structured Modeling

- So far we have mostly focused on simple IID (Independently and Identically Distributed) data.
  - Sometimes you want to predict based on structured inputs or outputs.
  - E.g,. Kinect: Dozens of target variables. How does it violate IID?
    - Not independent in time or space.
    - (Joints constrain each other, and move smoothly in time)



Time series Analysis

- Time-series analysis: Very well studied structured prediction task
  - Where the data has a natural ordering.
- Lots of applications: Finance, Weather, Demographics, Commercial Demand Forecasting, Control, Radar, etc
  - Signal can be either continuous or discrete.
- Normal predictive model:
  - $y_1=f(x_1)$, $y_2=f(x_2)$, etc. All y_i and y_j are predicted independently.
  - Time-series:
    - Instance indices are meaningfully ordered.
    - y_2 depends on y_1, y_n depends on $y_{(n-1)}$, etc.
- This is a deep specialist area…. But
  - We can understand+contribute given our good foundations
  - With a bit of care….



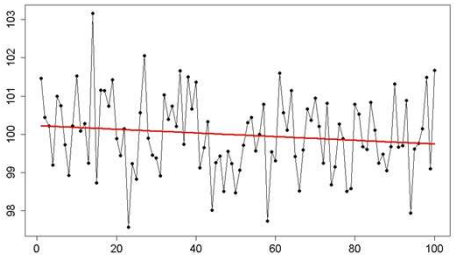
Time series Analysis: Motivation

What do we want to do with time-series?

- Trend detection:
  - Is there a trend? Is it going up or down?
- Smoothing:
  - Data is believed to be a noisy observation of a true underlying phenomenon.
  - Smooth out the noise to reveal the underlying quantity.
- Forecasting:
  - Predict the future of the time-series

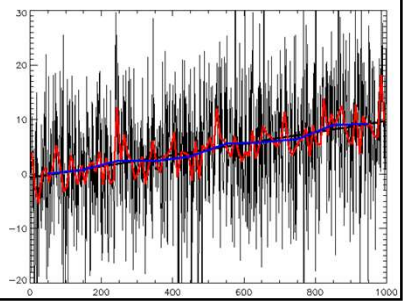
Time series Analysis: Trends

- Simplest trend detection approach:
  - Fit a linear regression:
  - y: Observation value. x: time-stamp of each observation.
  - Fit y=ax+b with standard least squares linear regression
- If no trend: Data is not systematically dependent on time
  - a=0, b=data mean
- If there is a trend:
  - Parameter a reveals its direction.
- Also:
  - Periodic trends, etc



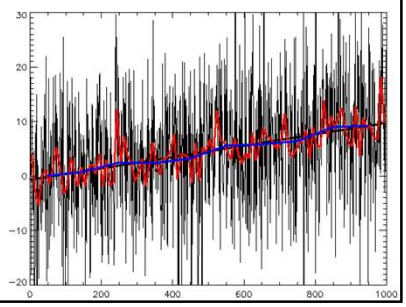
Time series Analysis: Smoothing

- Motivating Scenario for Smoothing Task:
  - Noise Removal: Softly reveal qualitative trends by smoothing out noise
  - Tracking: Underlying quantity to be revealed, observed by noisy sensors
- Noise Removal
  - E.g., Economic statistics, finance, climate change/global warming
- Tracking
  - E.g., Radar/Sonar, self-driving cars
  - E.g., Robotics, Weather



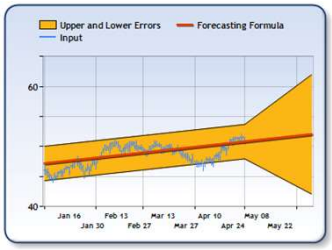
Time series Analysis: Noise Removal

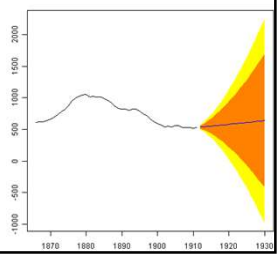
- Can you think of an algorithm?
  - Input: $y_1 \ldots y_N$. Output: Smoother $z_1 \ldots z_N$?
- Moving average:
  - E.g., $Z_i=(y_i+y_{(i-1)}+y_{(i-2)})/3$
  - This is like a low-pass filter in signal processing
- Exponential moving average:
  - E.g., $z_i=a*y_i+(1-a)*z_{(i-1)}$
  - Gives recent observations more weight
- Smoothing strength parameters:
  - Window size + 'a'



Time series Analysis: Forecasting

- Forecasting
  - (Contrast prediction!!)
  - Input: $y_1 \ldots y_n$, Predict: y_{n+1}
  - Any ideas how?
- Three simple ideas:
  1. Continue predicting y_n
  2. Get current gradient $g=(y_n-y_{(n-t)})/t$
     - Predict $y_{(n+t)}=y_n+g*t$
  3. Fit the global trend of the dataset
     - (Learn $y=f(t)$, globally)
     - Assume the trend continues: Predict $y=at+b$





Time series Analysis: Forecasting: Autoregressive Models

- Forecasting: Input: $y_1 \ldots y_{n-1}$, Predict: y_n
- View as regressing on past signal!
  - => "Autoregressive Models"
- Setup a regression problem. (Linear least squares)
  - Input: Historical Signal. Output: Current signal value.
  - Each time-instant of historical signal gives a new datapoint to train this regression
  - The value of *p* is the model order. Knows as AR(p)
    - p = 0 always predicts the overall dataset average.
    - p = 1 is like looking at the previous gradient (option #2 from prev slide)
    - p > 1 predicts from the previous time and beyond
    - Large p => complex model with many parameters, can overfit.

$$y_t = w_0 + w_1 y_{t-1} + w_2 y_{t-2} + w_3 y_{t-3} \ldots \qquad y_t = w_0 + \sum_{i=1}^{p} w_i y_{t-i}$$

Time series Analysis: Forecasting

- Autoregressive models predict based on history: $y_t = w_0 + \sum_{i=1}^{p} w_i y_{t-i}$
- As data-miners rather than signal-processors, we are more interested in finding exogenous predictive inputs.
- Suppose we want to predict stock market given the news: IID approach
  - Collect news articles x_i, i=1:N
  - Collect stock price y_i, at the time of each news article publication.
  - Each $\mathbf{x}_i$ could be encoded:
    - Scalar: Use the sentiment* of the article. (* Another predictive ML model)
    - Vector: Use bag of words
  - Using all $\{y,\mathbf{x}\}_i$ pairs, train a regressor for stock price y given news x

Time series Analysis: Forecasting

- We now know how to predict y_{t+1} based on
  1. historical signal $y_1..y_t$ (e.g., past stock market; autoregressive)
  2. external data x_{t+1} (e.g., news; conventional regression)
- Combine the two:
  - Setup a regression task to predict y_{t+1} with both inputs

$$y_t = \mu + \sum_{i=1}^{p} w_i y_{t-i} + \sum_{i=1}^{n} \beta_i x_{t-i}$$

  - Knows as an ARX(p,n) model. AutoRegressive with eXogenous inputs

Time series Analysis: Forecasting

- So far we looked at continuous time-series using regression
- Everything works analogously with discrete time-series and classification.
  - E.g., used by the predictive text in your mobile phone!
- Sometimes there is an option about how to represent target y:
  - E.g., Direct: Predict an actual market price at each time. VS:
  - Convert from value of y to per-instant change in y. (Avoids having a large bias/offset term)
  - Convert to up/down/constant (discretizing change in y)
  - Convert to change/no change (discretizing absolute val of change in y)

Time series Analysis: Forecasting

Training and testing subtleties

- For IID data, we can divide training/validation/testing instances randomly.
- For time-series data:
  - Need a training sequence of instances, so can't divide randomly.
  - Use training and testing contiguous segments.
- If the goal is forecasting, be careful not to include any future information in training/testing the model.
  - E.g., if predicting y_t at test/validation, should not have seen any $y_{s>t}$, or $x_{s>t}$ during validation/train respectively.

Outline

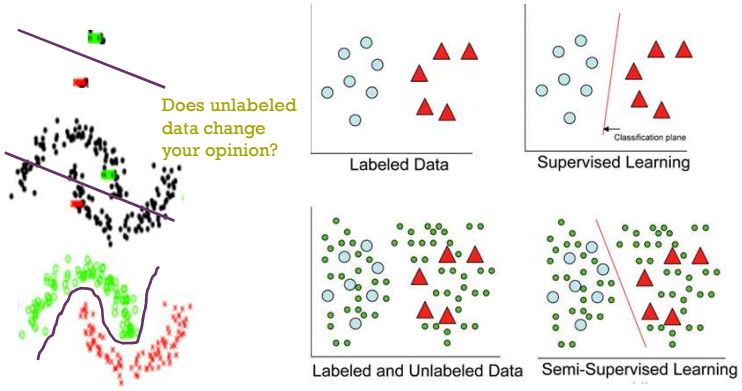
- Some Advanced Models
  - Support Vector Machines
  - Neural Networks
- Structured Modeling
  - Time-series modeling
- Special Settings
  - Semi-supervised learning
  - Active Learning

Semi-supervised learning

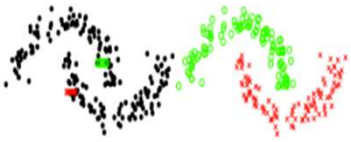
- Live in age of "Big Data"
  - But Big Data does not necessarily imply Big Annotation.
- E.g., Automated network intrusion data:
  - Can easily log all traffic on a network
  - Identifying actual examples of hacking takes lots of expert time…
  - So have millions of unclassified network traffic records x, but only dozens definitively identified as hacking & dozens-hundreds identified as legitimate, y.
- => Can we use the unlabeled examples to learn better than with the labeled ones alone??
- Conventional Learning. Learn $y=f(x)$ with:
  - Given $\{x_i,y_i\}$, $i=1..N$.
- Semi-supervised Learning. Learn $y=f(x)$ with:
  - Given $\{x_i,y_i\}$, $i=1..N$, AND $\{x_j\}$, $j=1…M$. Typically M>>N.
  - Question: Is it possible to learn better using M additional xs?
  - Question: How?

Semi-supervised learning: Illustration

- Suppose you are given few labeled example per class…
  - … with additional unlabeled data



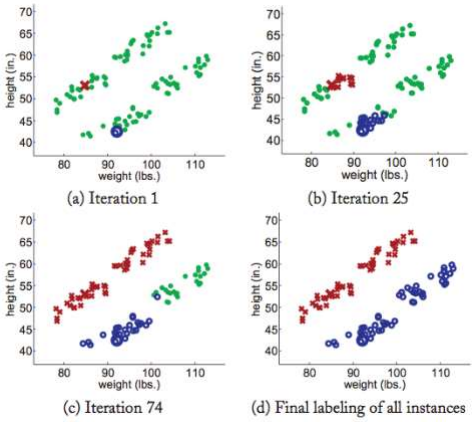
Semi-supervised learning: Algorithms



- Any idea how to achieve this?
- The intuition being used is formally called the Manifold Assumption
- An easy algorithm: "Self-training"
- Iterate:
  - Using the current labeled data X_l
  - Attempt to classify all unlabeled data X_u
  - Find the most confidently classified x^* in X_u
  - Assume its label y^* is correct.
  - Add $\{x^*, y^*\}$ to the labeled set X_l.
  - Retrain classifier

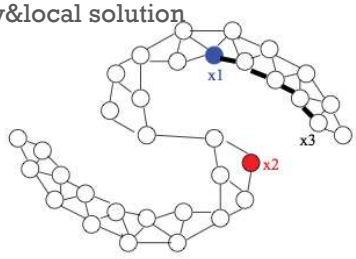
Semi-supervised learning: Algorithms

- An easy algorithm: "Self-training"



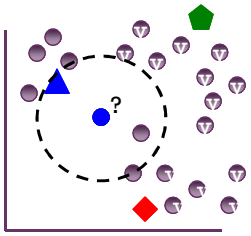
Semi-supervised learning: Algorithms

- A better algorithm: Graph-based label propagation
  - With euclidean distance x_3 would be red.
- Use a KNN graph rather than euclidean distance
  - E.g., x_3 influenced more by x_1 than x_2. Although x_2 closer in euclidean.
- Get a globally optimal rather than greedy&local solution
  - (Contrast self-train)

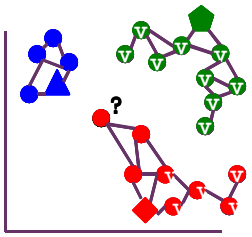


Graph-based Semi-Supervised Learning Illustration

- Contrast:
  - Regular supervised learning (1-NN)
  - Graph-based SSL.



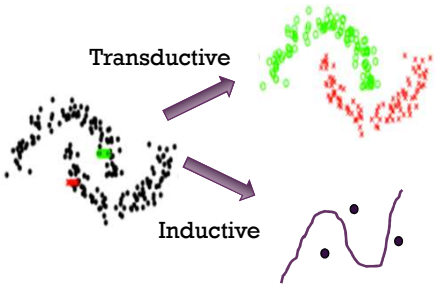
1-NN Recognition



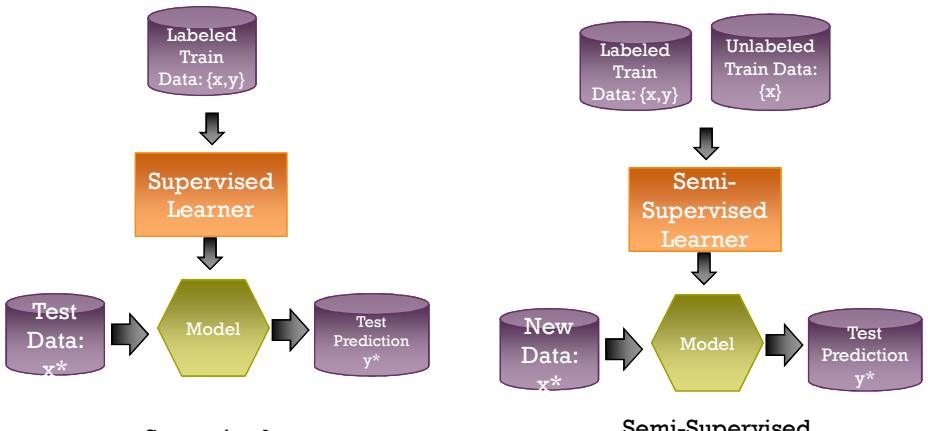
Manifold Label Propagation

SSL: Transductive vs Inductive

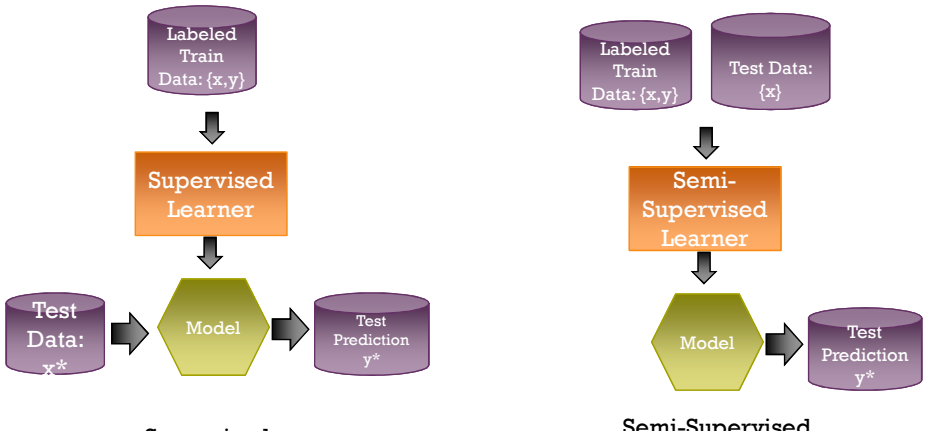
- SSL has two application settings:
  - Inductive: Train data is a mix of labeled + unlabeled.
    - Do SSL on train. Then use the result as a model for test data
  - Transductive: Train data all labeled. Treat test data as the unlabeled data.
    - => Need to process test data in batch. Not for online streaming.



SSL: Dataflow



SSL: Dataflow



Semi-supervised learning: Summary

- SSL: Exploit un-labeled data to improve learning if limited labels.
- Notes:
  - Usually useful when raw data is cheap, but labels rare/expensive
    - (If plentiful labels, model is already good, unlabeled doesn't help)
  - Can be much better than supervised when few labels, but:
    - Not usually guaranteed to improve over supervised in every case
    - Its possible to do worse than supervised in some cases. Needs care.
  - Many algorithms with different assumptions/caveats/strengths/weakness
  - Graph based methods good but can be expensive $O(N^2)$
- Transductive versus Inductive:
  - Depends on method, and problem setting.

Outline

- Some Advanced Models
  - Support Vector Machines
  - Neural Networks
- Structured Modeling
  - Time-series modeling
- Special Settings
  - Semi-supervised learning
  - Active Learning

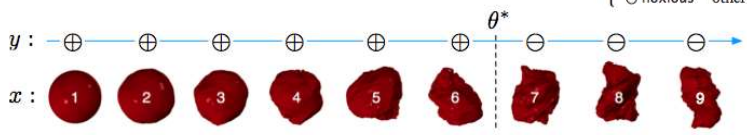
Active Learning

- Live in age of "Big Data"
  - But Big Data does not necessarily imply annotations for everything.
  - Semi-supervised learning can make better use of limited labels…
- Active Learning:
  - It's about closing the loop to find out which labels are worth spending the effort to collect.
    - (Contrast SSL: Make use of unlabeled data. Not getting new labels.)
- Active Learning: From the perspective of the learning algorithm:
  - The annotation on some datapoints are much more useful than others.
  - => Some are redundant
  - => Some tell the model something new it didn't know

Active Learning: View 1: Automating Experimental Design

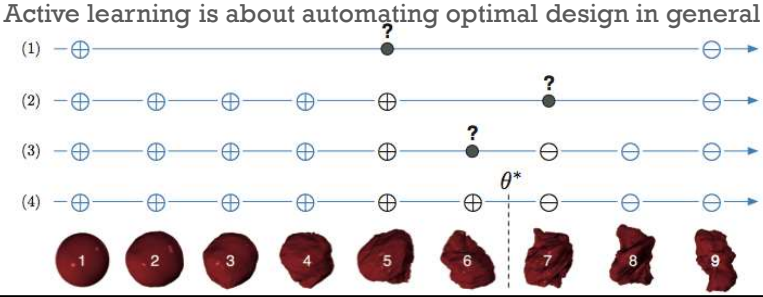
- Suppose we arrive on an alien planet: Lots of alien fruit

- Some fruit taste great, some make your colonists sick…
  - Need to work out how to classify edible & non-edible fruit
  - Fruit smoothness is the key indicator. Define $h(x;\theta) = \begin{cases} \oplus \text{ safe} & \text{if } x < \theta, \text{ and} \\ \ominus \text{ noxious} & \text{otherwise.} \end{cases}$



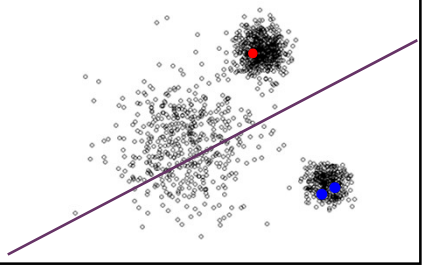
Active Learning: View 1: Automating Experimental Design

- How would you create the fruit safety classifier?
  - Option 1: Assign a group of colonists to eat each type of fruit
    - But each {x,y} annotation is costly (lots of sickness) => rebellion!
    - So want to get the classifier with minimal # of observations
  - Option 2: Binary search.
- Active learning is about automating optimal design in general case



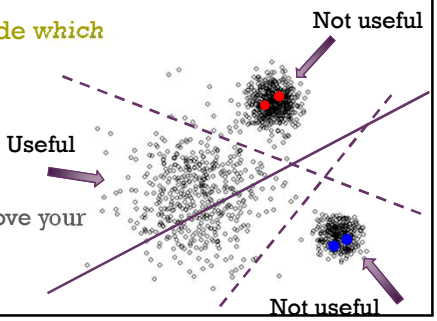
Active Learning: View 2: Introspection & Informative Data

- Suppose we are learning a binary classification problem
  - Suppose our red & blue annotation so far is…
  - Then decision boundary is…
- Clearly if we now spend resources to test more points, some are more useful than others.
  - Which? Why?



Active Learning: View 2: Introspection & Informative Data

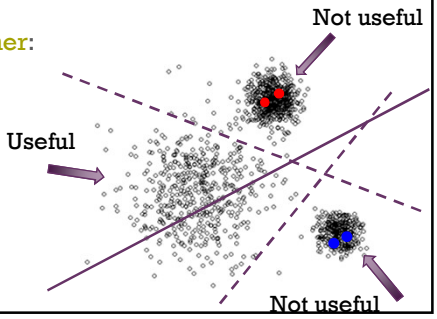
- Suppose we are learning a binary classification problem
  - Red & blue annotation and decision boundary is…
- Some points' labels have a more significant implication for the model
- The model should introspect and decide *which* ones likely to be useful
  - => Annotate those preferentially
- Studying analogy:
  - Don't ask the teacher random questions
  - Ask questions that are expected to improve your understanding



Active Learning: Framework

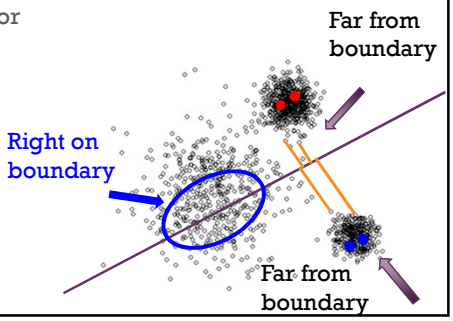
Active Learning Algorithms

- Similar to semi-supervised, we need to take as input:
  - $D_l=\{x,y\}_i, i=1..N$, $D_u=\{x\}_j, j=1..M$
- AL Output is *k* – the identity of a datapoint which, if queried, is expected to be useful
- AL works in tandem with regular learner:
- Iterate:
  - Train $y=f(x)$ on D_l
  - Find most useful point $k=g(f,D_l,D_u)$
  - Get label y_k to go with x_k.
  - Add $\{x,y\}_k$ to D_l



Active Learning: Algorithms

- AL task: Find most useful point $k=g(f,D_l,D_u)$
- Can you think of any algorithm?
- Simple algorithm: Minimum Certainty / Max Entropy
  - Return the unlabeled point that is currently:
    - Closest to the decision boundary, or
      - (Binary only)
    - Most uncertain
      - (Need probabilistic model)
      - (More general)



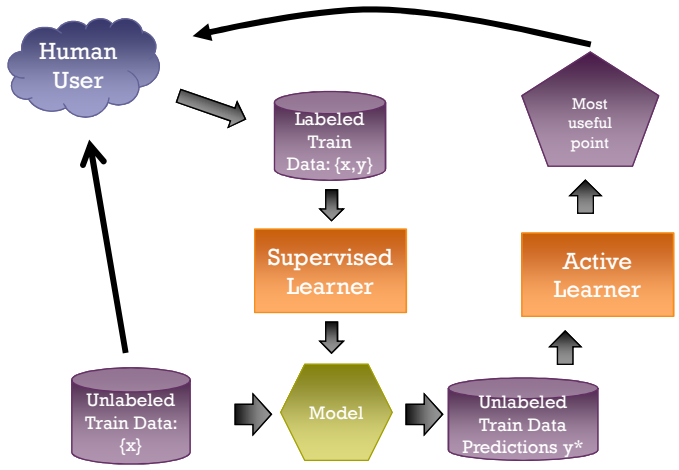
Active Learning: "Classic" lending application

- Loan Making
  - Normal banks have classifier y=f(x)
  - Predicts y (default/repay) given x (features such as job, credit history, etc)
- The problem is labeled instances {y,x} are very biased.
  - Because they preferentially lend to "good" borrowers.
  - This data bias means the model f(x) may not actually be good
- Active Learning:
  - Select the putative borrower x, who if we lent to him (i.e., got a label), would maximally increase the effectiveness of the lending model y=f(x)
  - Then lend to that guy, whether or not we expect he has a good chance to repay.
    - Contrast regular bank whose lending criteria is repayment probability.

Active Learning: EECS Research ☺ "Classic" lending application

- Loan Making
  - Normal banks have default/repay classifier y=f(x)
- Active Learning:
  - Select the putative borrower x, who if we lent to him (i.e., got a label), would maximally increase the effectiveness of the lending model y=f(x)
  - Lend to that guy, whether or not we expect he has a good chance to repay.
  - The value to improved performance on all future decisions, outweighs the cost of borrower's potential default.
  - Advanced active learning algorithms can estimate the expected increase in future accuracy of a particular annotation.
    - We have published some methods for this ☺
    - Thus can directly compare the expected future profit of an annotation, with the expected loss of a default.
- Conventional banks don't do things this clever.
  - Wonga (online lender) is famous for burning $100M on exploratory (random) lending, and now has the unique asset of a much better default model.

Active Learning: Dataflow



Active Learning: From Pool to Stream based

- Framework so far is "Pool based"
- Iterate, Given: $D_l=\{x,y\}_i, i=1..N$, $D_u=\{x\}_j, j=1..M$
  - Train $y=f(x)$ on D_l
  - Find most useful point in D_u: $k=g(f,D_l,D_u)$
  - Get label y_k to go with x_k.
  - Add $\{x,y\}_k$ to D_l
- What is the computational complexity of this?
  - Depends on base classifier f(x). But…
  - Per iteration:
    - SVM: $O(Md+N^3d)$, Linear: $O(Md+Nd)$, KNN: $O(MNd+1)$
  - Important to have incremental classifier updates
  - Doesn't scale so well with pool size M

Active Learning: Stream based

- Stream Based:
  - Data arrives in a stream (either naturally, or due to CPU/memory constraint).
  - You have to make an independent decision about if to query at each time-step.
- Iterate:
  - Receive new unlabeled data x_j
  - Boolean useful = $g(x_j,D_l,f)$
  - IF(useful) then Query y_j
  - Else continue
- Contrast
  - Pool: function g() looked at pool and returned most useful.
  - Stream: g() looks at individual instance and makes a decision
- Method for g() in stream case?
  - Check if confidence/decision boundary distance is below a threshold

Active Learning: For Interactive Mining of Interesting Patterns

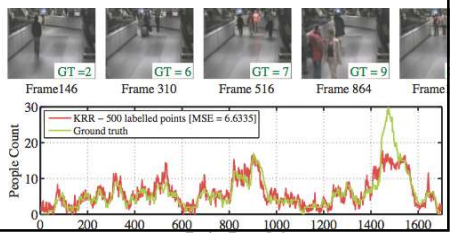
- If you have a classifier for an interesting pattern, it can return all interesting patterns, and you're done. But you usually don't.
- If you have lots of annotated patterns, you can train a classifier. But what if the category of interest is rare, un-annotated, and buried in other data?
- Active Learning can be particularly useful if categories of interest are rare.
  - => In this case randomly annotating data unlikely to annotate any interesting instances.
  - Start with your one annotated instance, and a suitably designed AL algorithm can try to find more of them.

Active Learning: Summary

- Active Learning: Introspect to decide which are useful data to get labels for
- Notes:
  - Usually useful when raw data is cheap, labels rare/expensive, but can be annotated
    - (If plentiful labels, model is already good, unlabeled doesn't help)
  - Theoretically shown that can be exponentially better than random labeling.
    - If random needs N_R instances to work well, active can work well with $\log(N_R)$ instances.
  - Many algorithms with different assumptions/caveats/strengths/weakness
- Pool is better studied, but can be expensive
- Stream useful for some applications, or big data.
  - But usually introduces a free parameter.

Case Study: Crowd Counting
EECS Research ☺

- "Crowd Counting": Regression problem from image pixels to a number indicating quantity of people in the scene.
  - Annotating a single frame requires manually counting all people => expensive
  - Challenge: Each view different, so need per camera training.
- Highly desired by:
  - Retail, Security, Airports, Urban Planners, etc
  - (Airports want to reduce queues!)



Case Study: Crowd Counting
EECS Research @ IEEE ICCV 2013 ☺

- "Crowd Counting": Regression problem from image pixels to a number indicating quantity of people in the scene.
  - Annotating a single frame requires manually counting all people => expensive
- Semi-supervised: Use unlabeled data manifold (graph-based) to improve when only few annotated frames.
- Active: Select which are the most useful few frames to have annotation for.