Introduction to IoT Data Stream Mining

Albert Bifet (@abifet)

Paris, 22 November 2017

albert.bifet@telecom-paristech.fr

Who are We

- Jesse Read
  - Associate Professor at École Polytechnique
  - MultiLabel Learning, Data stream mining and Deep Learning
    - MEKA: Multilabel Learning
    - MOA: Massive Online Analytics
- Albert Bifet
  - Associate Professor at Télécom ParisTech
  - Data stream mining algorithms and systems
    - MOA: Massive Online Analytics
    - Apache SAMOA: Scalable Advanced Massive Online Analytics

Outline

1. Introduction
2. Stream Algorithmics
3. Classification in Multi-output Data Streams
4. Concept Drift
5. Multi-output Learning
6. Ensemble Methods
7. Regression
8. Clustering
9. Frequent Pattern Mining

IoT Data Stream Mining

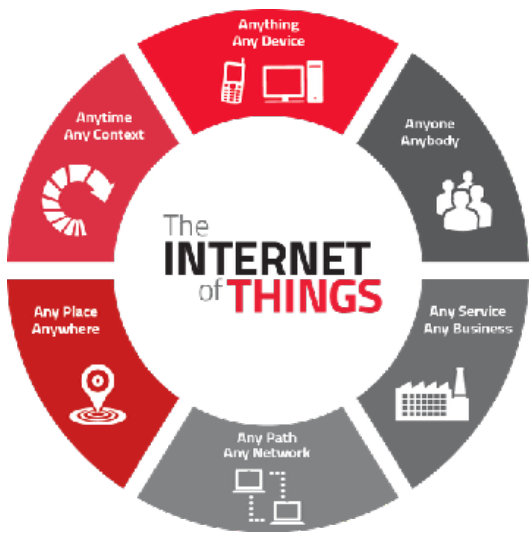
Assessment
10% Lab Assignments
30% Project
60% Test

Classes
22/11, 29/11, 13/12, 20/12 Wednesdays at 9:00

Session Labs: 6/12 and 10/01

Important Dates
Project Presentation: January 17
Test: January 31

INTERNET OF THINGS

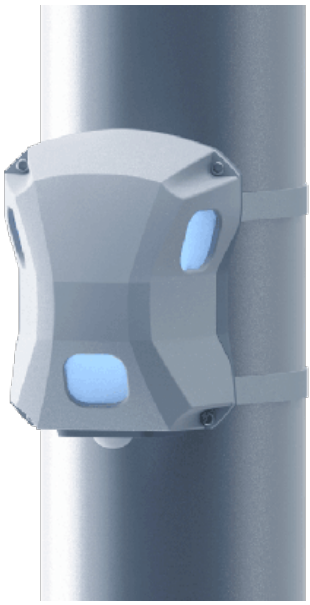


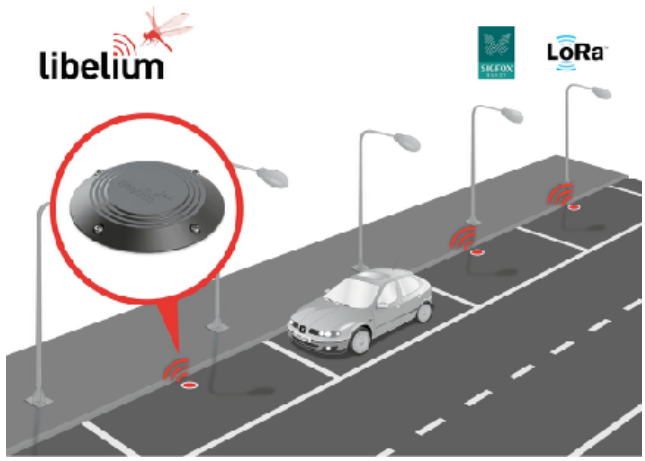
IoT: sensors and actuators connected by networks to computing systems.

- Gartner predicts 20.8 billion IoT devices by 2020.
- IDC projects 32 billion IoT devices by 2020

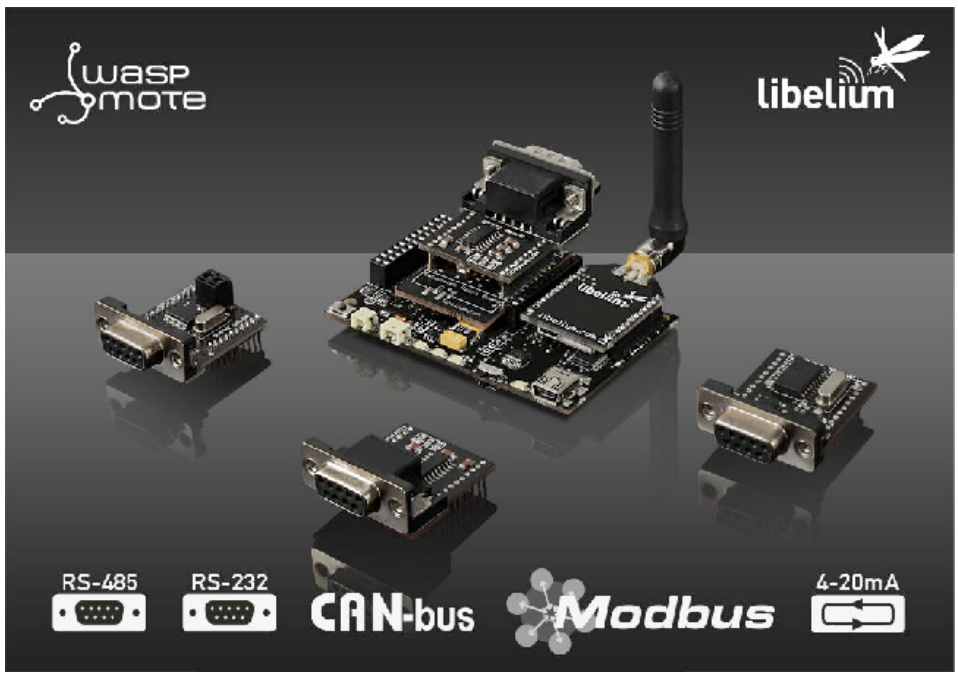
IoT Applications For Energy Management

IoT Applications For Connected/Smart Home



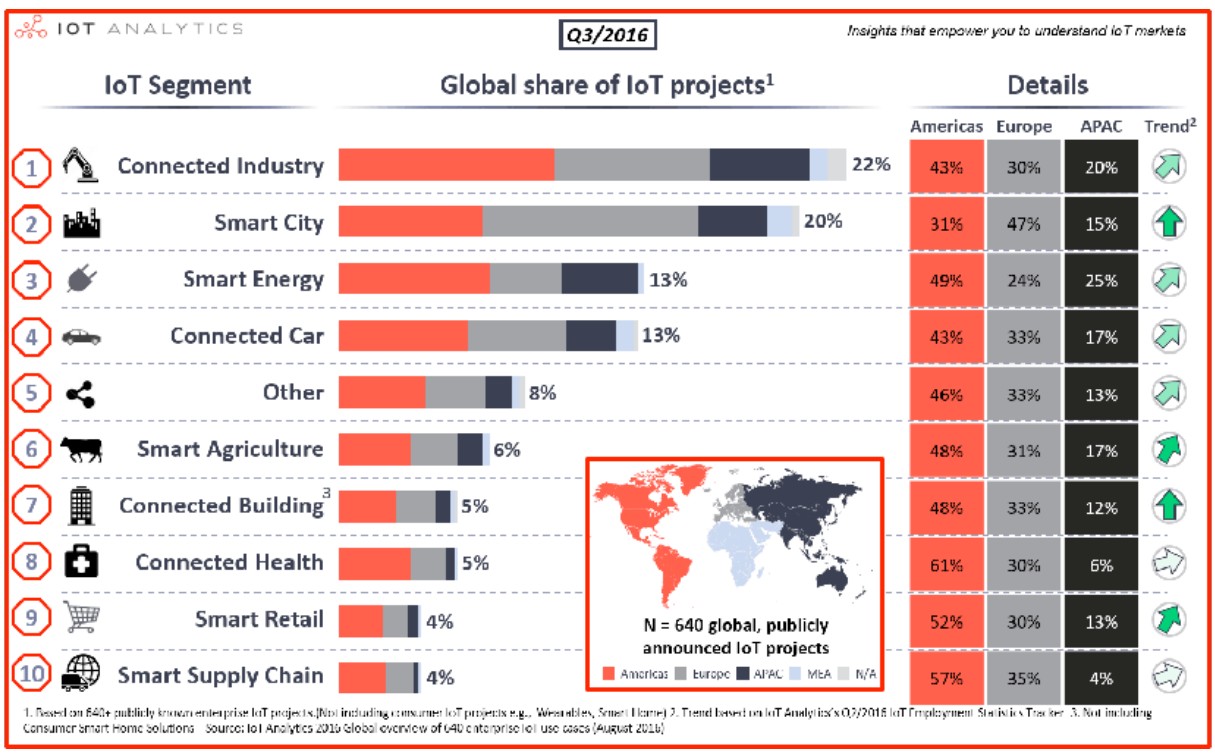


IoT Applications For Smart Cities



IoT Applications For Industrial Automation

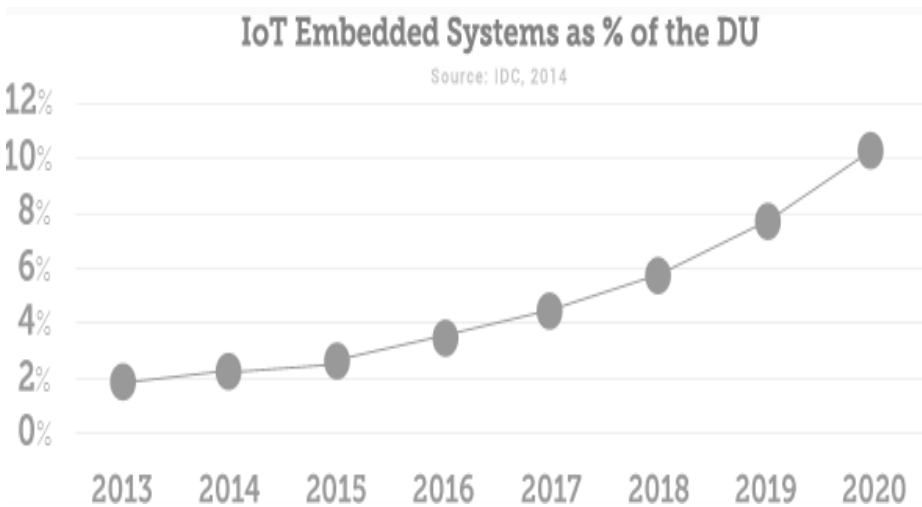
Applications IoT Analytics



IOT AND INDUSTRY 4.0

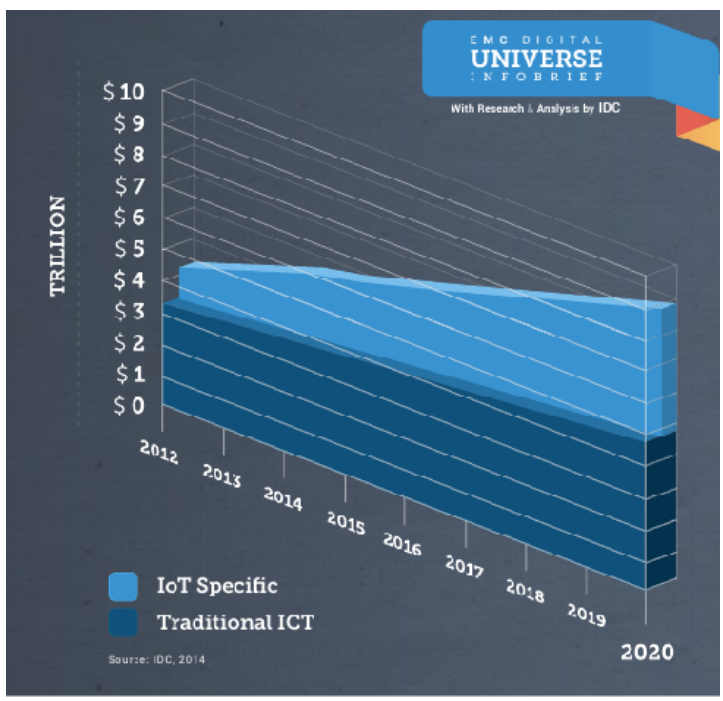
- Interoperability: IoT
- Information transparency: virtual copy of the physical world
- Technical assistance: support human decisions
- Decentralized decisions: make decisions on their own

INTERNET OF THINGS

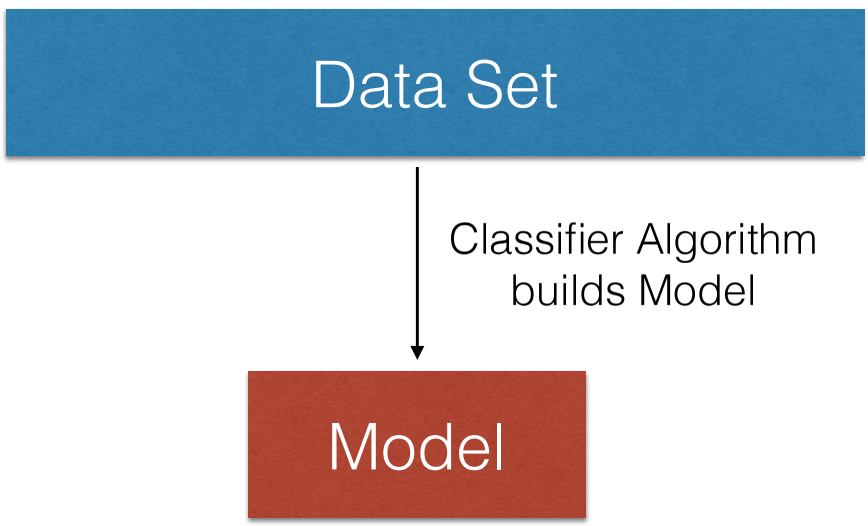


- EMC Digital Universe, 2014

INTERNET OF THINGS

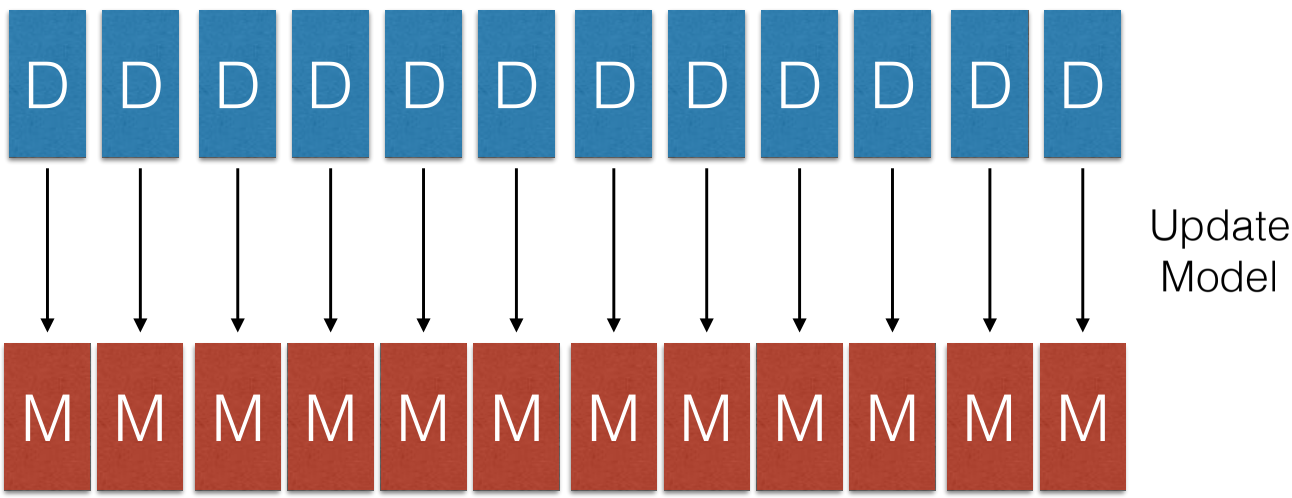


- EMC Digital Universe, 2014



Analytic Standard Approach

Finite training sets
Static models

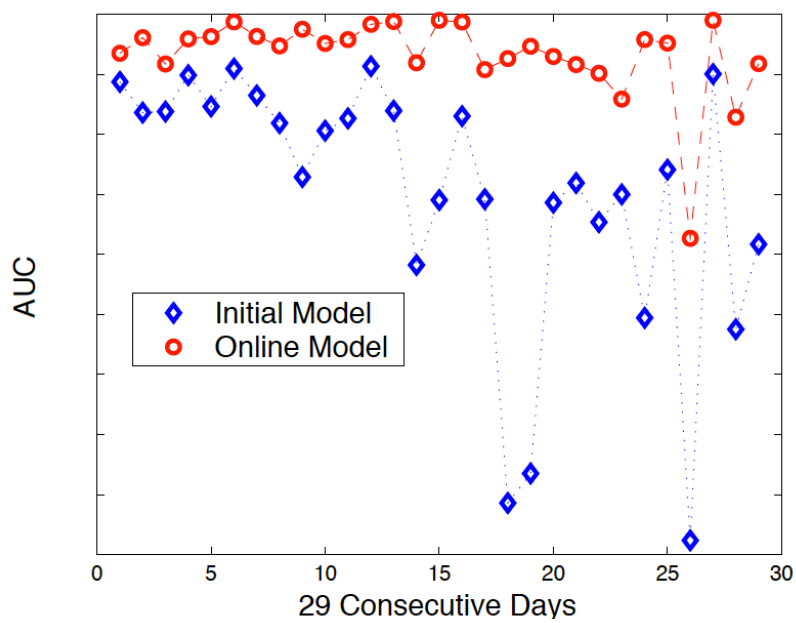


Data Stream Approach

Infinite training sets
Dynamic models

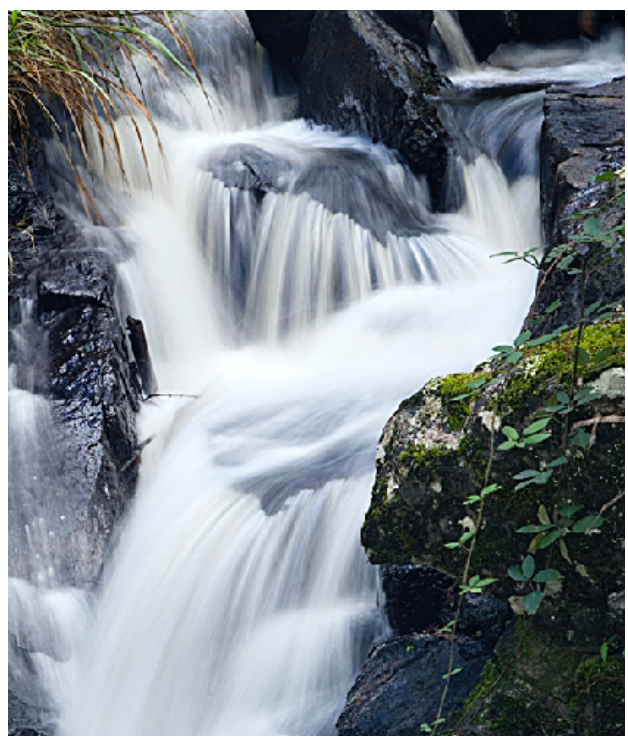
Pain Points

- Need to **retrain!**
  - Things change over time
  - How often?
- Data unused until next update!
  - Value of data wasted



IoT Stream Mining

- Maintain models online
  - Incorporate data on the fly
  - Unbounded training sets
  - Resource efficient
  - Detect changes and adapts
  - Dynamic models



Approximation Algorithms

- General idea, good for streaming algorithms
- Small error ε with high probability $1-\delta$
  - True hypothesis H, and learned hypothesis $\hat{H}$
  - $\Pr[\ |H - \hat{H}| < \varepsilon|H|\ ] > 1-\delta$

Approximation Algorithms

- What is the largest number that we can store in 8 bits?

1	0	1	0	1	0	1	0

Approximation Algorithms

- What is the largest number that we can store in 8 bits?

Programming Techniques
S.L. Graham, R.L. Rivest Editors

Counting Large Numbers of Events in Small Registers

Robert Morris
Bell Laboratories, Murray Hill, N.J.

It is possible to use a small counter to keep approximate counts of large numbers. The resulting expected error can be rather precisely controlled. An example is given in which 8-bit counters (bytes) are used to keep track of as many as 130,000 events with a relative error which is substantially independent of the number n of events. This relative error can be expected to be 24 percent or less 95 percent of the time (i.e. $\sigma = n/8$). The techniques could be used to advantage in multichannel counting hardware or software used for the monitoring of experiments or processes.

WHAT IS **MOA**?

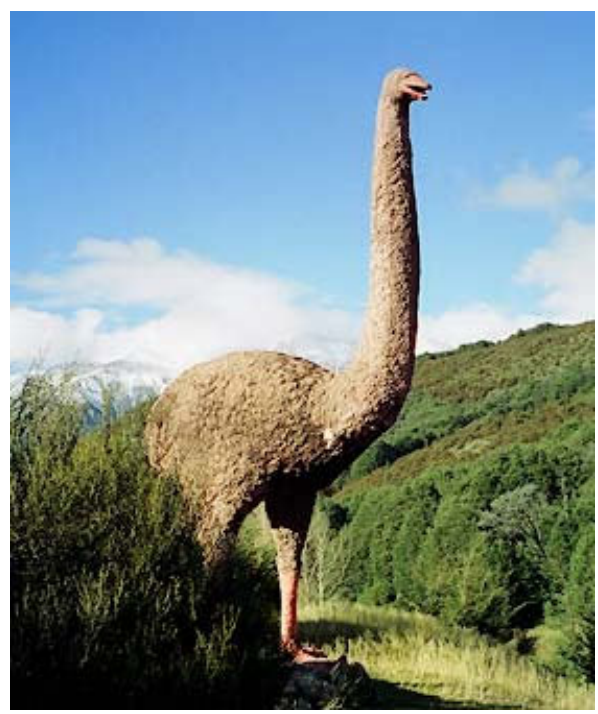
MOA

- {M}assive {O}nline {A}nalysis is a framework for online learning from data streams.
- It is closely related to WEKA
- It includes a collection of offline and online as well as tools for evaluation:
  - classification, regression
  - clustering, frequent pattern mining
- Easy to extend, design and run experiments

WEKA: the bird

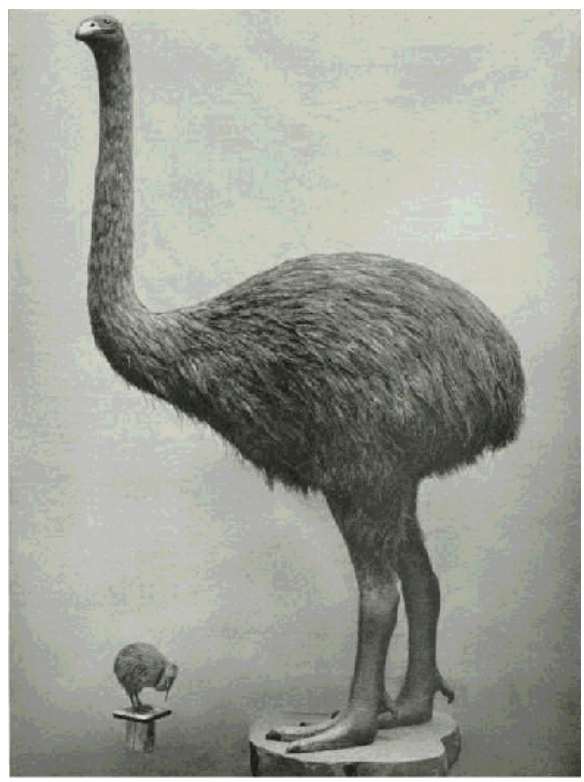
MOA: the bird

The Moa (another native NZ bird) is not only flightless, like the Weka, but also extinct.



MOA: the bird

The Moa (another native NZ bird) is not only flightless, like the Weka, but also extinct.

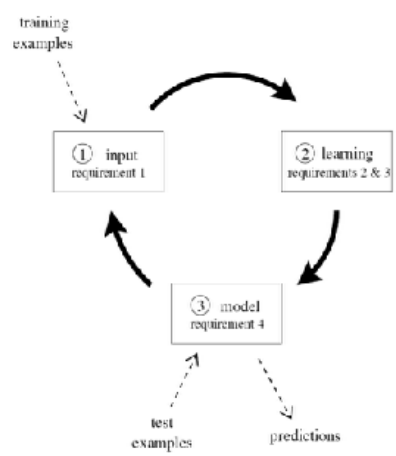


MOA: the bird

The Moa (another native NZ bird) is not only flightless, like the Weka, but also extinct.

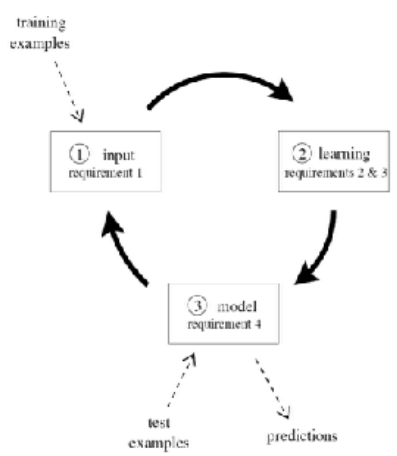
STREAM SETTING

- Process an example at a time,and inspect it only once (at most)
- Use a limited amount of memory
- Work in a limited amount of time
- Be ready to predict at any point

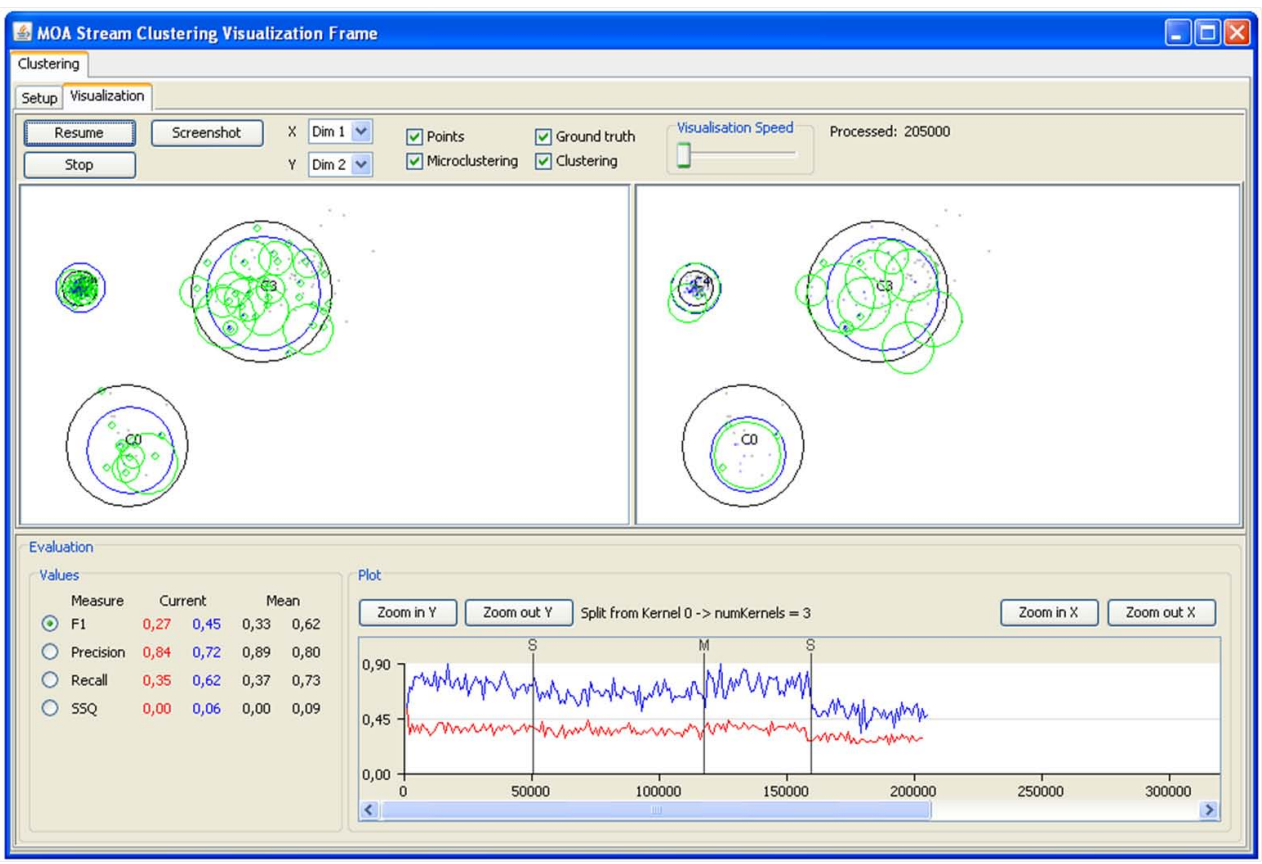


STREAM EVALUATION

- Holdout Evaluation
- Interleaved Test-Then-Train or Prequential



GUI


MOA Stream Clustering Visualization Frame
Clustering
Setup
Visualization
Resume
Stop
Screenshot
X
Dim 1
Y
Dim 2
Points
Microclustering
Ground truth
Clustering
Visualisation Speed
Processed: 205000
C3
C0
Evaluation
Values
Measure
Current
Mean
F1 0,27 0,45 0,33 0,62
Precision 0,84 0,72 0,89 0,80
Recall 0,35 0,62 0,37 0,73
SSQ 0,00 0,06 0,00 0,09
Plot
Zoom in Y
Zoom out Y
Split from Kernel 0 -> numKernels = 3
Zoom in X
Zoom out X
S
M
S
0,90
0,45
0,00
0
50000
100000
150000
200000
250000
300000

COMMAND LINE

- `java -cp .:moa.jar:weka.jar -javaagent:sizeofag.jar moa.DoTask "EvaluatePeriodicHeldOutTest -l DecisionStump -s generators.WaveformGenerator -n 100000 -i 100000000 -f 1000000" > dsresult.csv`
- This command creates a comma separated values file:
  - training the DecisionStump classifier on the WaveformGenerator data,
  - using the first 100 thousand examples for testing,
  - training on a total of 100 million examples,
  - and testing every one million examples

Data Streams

Big Data & Real Time

Stream Algorithmics

Albert Bifet (@abifet)

Paris, 22 November 2017
albert.bifet@telecom-paristech.fr

Data Streams

Big Data & Real Time

Data Streams

Data Streams

- Sequence is potentially infinite
- High amount of data: sublinear space
- High speed of arrival: sublinear time per example
- Once an element from a data stream has been processed it is discarded or archived

Big Data & Real Time

Data Stream Algorithmics

Example

Puzzle: Finding Missing Numbers

- Let π be a permutation of $\{1, \ldots, n\}$.
- Let π_{-1} be π with one element missing.
- $\pi_{-1}[i]$ arrives in increasing order

Task: Determine the missing number

Big Data & Real Time

Data Stream Algorithmics

Example

Puzzle: Finding Missing Numbers

- Let π be a permutation of $\{1, \ldots, n\}$.
- Let π_{-1} be π with one element missing.
- $\pi_{-1}[i]$ arrives in increasing order

Task: Determine the missing number

Use a n-bit vector to memorize all the numbers ($O(n)$ space)

Big Data & Real Time

Data Stream Algorithmics

Example

Puzzle: Finding Missing Numbers

- Let π be a permutation of $\{1, \ldots, n\}$.
- Let π_{-1} be π with one element missing.
- $\pi_{-1}[i]$ arrives in increasing order

Task: Determine the missing number

Data Streams:
$O(\log(n))$ space.

Big Data & Real Time

Data Stream Algorithmics

Example

Puzzle: Finding Missing Numbers

- Let π be a permutation of $\{1, \ldots, n\}$.
- Let π_{-1} be π with one element missing.
- $\pi_{-1}[i]$ arrives in increasing order

Task: Determine the missing number

Data Streams:
$O(\log(n))$ space.
Store

$$\frac{n(n+1)}{2} - \sum_{j \leq i} \pi_{-1}[j].$$

Big Data & Real Time

Data Streams

Approximation algorithms

- Small error rate with high probability
- An algorithm $(\epsilon, \delta)-$approximates F if it outputs $\tilde{F}$ for which $\Pr[|\tilde{F} - F| > \epsilon F] < \delta$.

Big Data & Real Time

Data Stream Algorithmics

Examples

1. Compute different number of pairs of IP addresses seen in a router
2. Compute top-k most used words in tweets

Two problems: find number of distinct items and find most frequent items.

8 Bits Counter

1	0	1	0	1	0	1	0

What is the largest number we can store in 8 bits?

8 Bits Counter

Programming Techniques
S.L. Graham, R.L. Rivest Editors

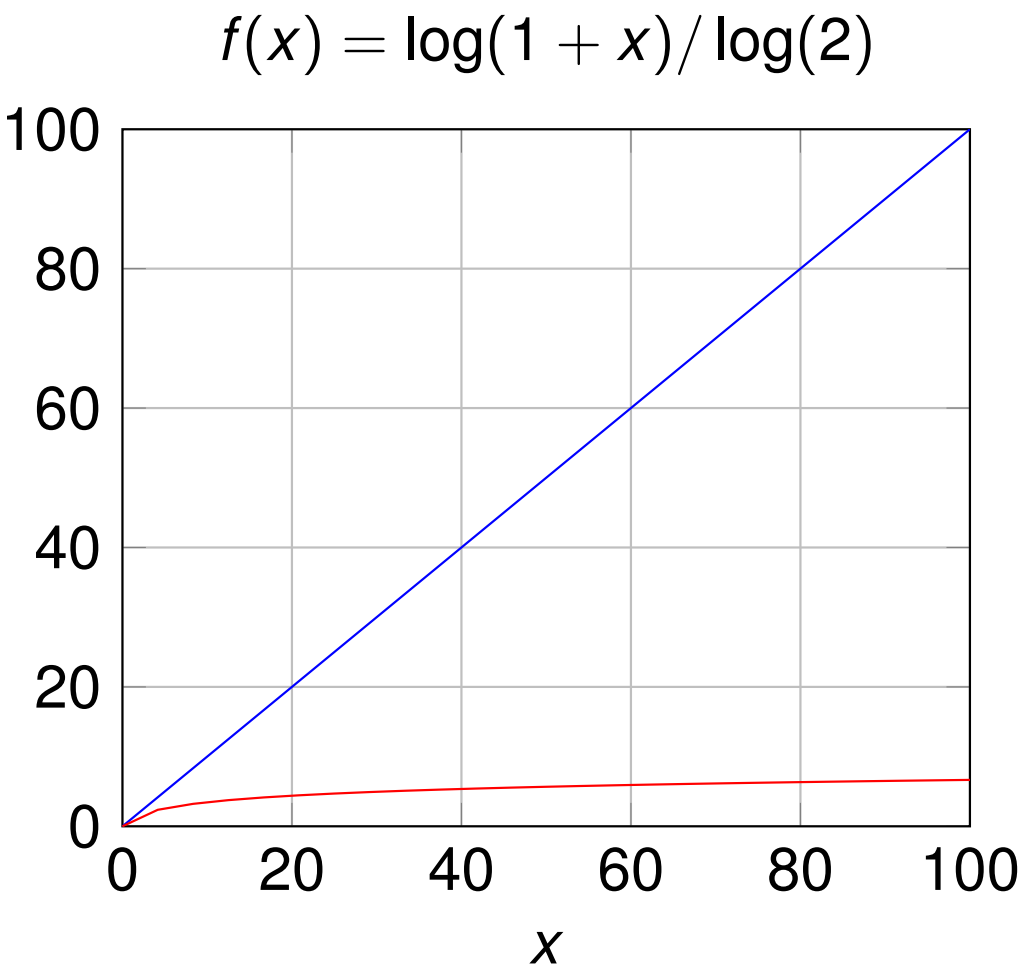
Counting Large Numbers of Events in Small Registers

Robert Morris
Bell Laboratories, Murray Hill, N.J.

It is possible to use a small counter to keep approximate counts of large numbers. The resulting expected error can be rather precisely controlled. An example is given in which 8-bit counters (bytes) are used to keep track of as many as 130,000 events with a relative error which is substantially independent of the number n of events. This relative error can be expected to be 24 percent or less 95 percent of the time (i.e. $\sigma = n/8$). The techniques could be used to advantage in multichannel counting hardware or software used for the monitoring of experiments or processes.

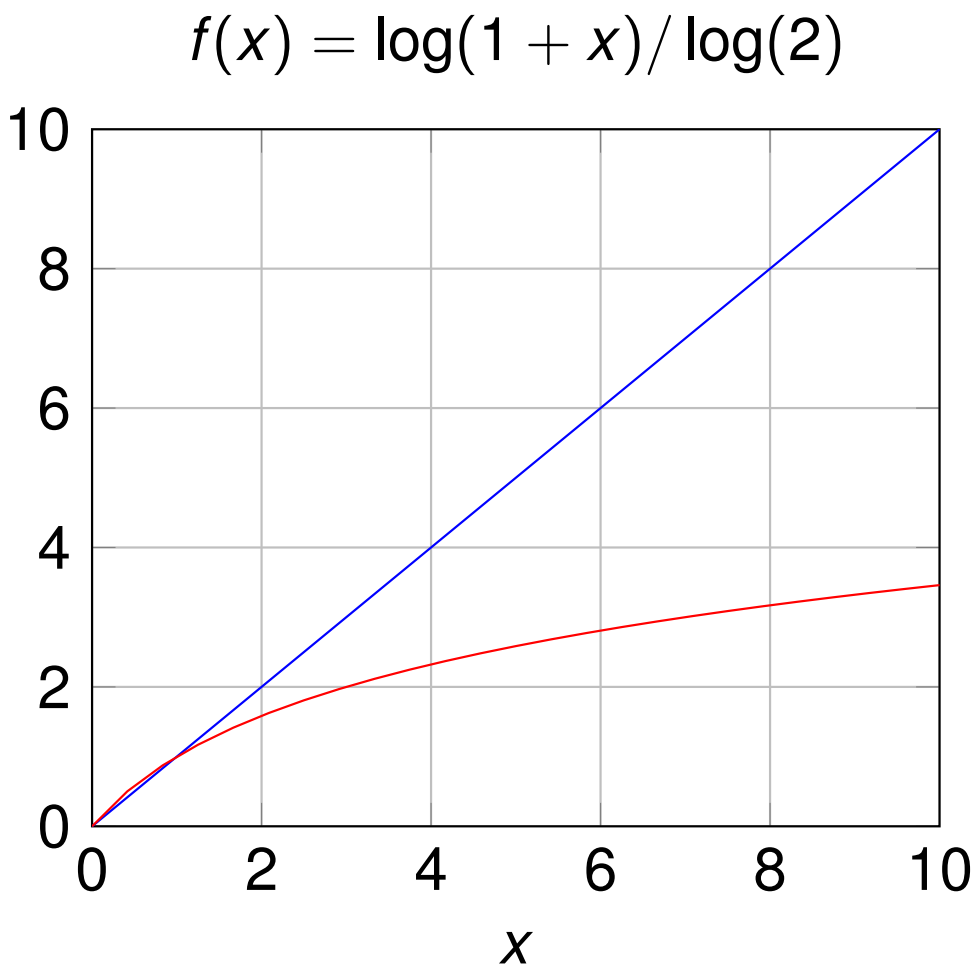
What is the largest number we can store in 8 bits?

8 Bits Counter



$f(0) = 0, f(1) = 1$

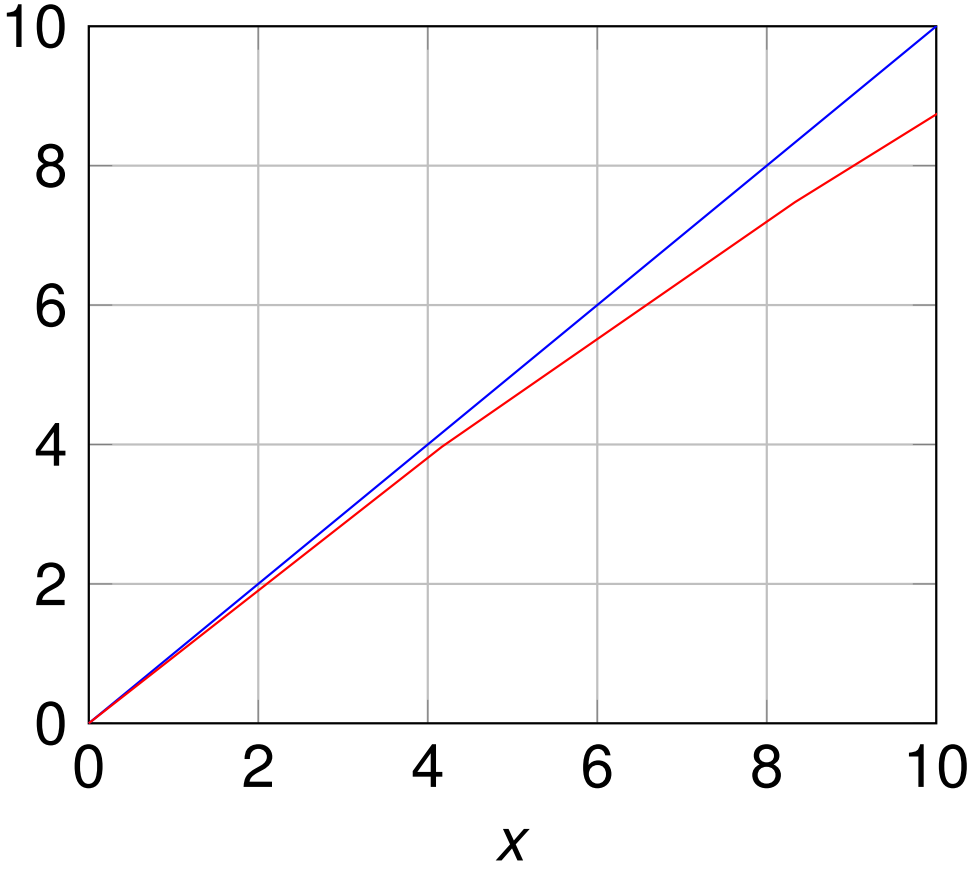
8 Bits Counter



$f(0) = 0, f(1) = 1$

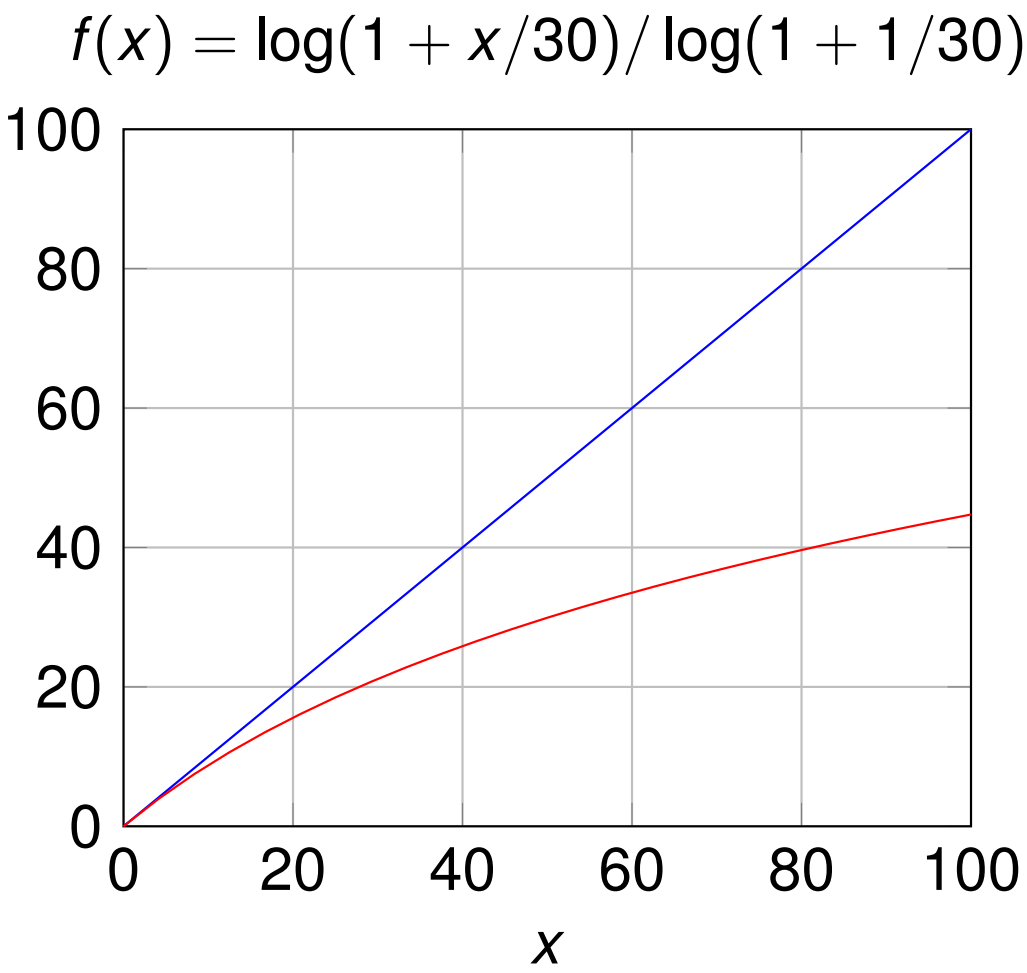
8 Bits Counter

$$f(x) = \log(1 + x/30)/\log(1 + 1/30)$$



$f(0) = 0, f(1) = 1$

8 Bits Counter



$f(0) = 0, f(1) = 1$

8 bits Counter

MORRIS APPROXIMATE COUNTING ALGORITHM

```
Init counter c ← 0
for every event in the stream
      do rand = random number between 0 and 1
         if rand < p
            then c ← c + 1
```

What is the largest number we can store in 8 bits?

8 bits Counter

MORRIS APPROXIMATE COUNTING ALGORITHM

```
Init counter c ← 0
for every event in the stream
        do rand = random number between 0 and 1
            if rand < p
                then c ← c + 1
```

With $p = 1/2$ we can store 2×256
with standard deviation $\sigma = \sqrt{n}/2$

8 bits Counter

MORRIS APPROXIMATE COUNTING ALGORITHM

```
Init counter c ← 0
for every event in the stream
      do rand = random number between 0 and 1
         if rand < p
            then c ← c + 1
```

With $p = 2^{-c}$ then $E[2^c] = n + 2$ with variance $\sigma^2 = n(n+1)/2$

8 bits Counter

MORRIS APPROXIMATE COUNTING ALGORITHM

```
Init counter c ← 0
for every event in the stream
      do rand = random number between 0 and 1
         if rand < p
            then c ← c + 1
```

$$\text{If } p = b^{-c} \text{ then } E[b^c] = n(b-1) + b,$$
$$\sigma^2 = (b-1)n(n+1)/2$$

Data Stream Algorithmics

Examples

1. **Compute different number of pairs of IP addresses seen in a router**
   IPv4: 32 bits
   IPv6: 128 bits
2. Compute top-k most used words in tweets

Find number of distinct items

Data Stream Algorithmics

Memory unit	Size	Binary size
kilobyte (kB/KB)	10^3	2^{10}
megabyte (MB)	10^6	2^{20}
gigabyte (GB)	10^9	2^{30}
terabyte (TB)	10^{12}	2^{40}
petabyte (PB)	10^{15}	2^{50}
exabyte (EB)	10^{18}	2^{60}
zettabyte (ZB)	10^{21}	2^{70}
yottabyte (YB)	10^{24}	2^{80}

Find number of distinct items

IPv4: 32 bits IPv6: 128 bits

Data Stream Algorithmics

Example

1. **Compute different number of pairs of IP addresses seen in a router**
   IPv4: 32 bits, IPv6: 128 bits
   Using 256 words of 32 bits accuracy of 5%

Find number of distinct items

Data Stream Algorithmics

Example

1. **Compute different number of pairs of IP addresses seen in a router**

   Selecting n random numbers,
   - half of these numbers have the first bit as zero,
   - a quarter have the first and second bit as zero,
   - an eigth have the first, second and third bit as zero..

   A pattern 0^i1 appears with probability $2^{-(i+1)}$, so $n \approx 2^{i+1}$

Find number of distinct items

Data Stream Algorithmics

FLAJOLET-MARTIN PROBABILISTIC COUNTING ALGORITHM

```
Init bitmap[0 ... L − 1] ← 0
for every item x in the stream
      do index = ρ(hash(x)) ▷ position of the least significant 1-bit
         if bitmap[index] = 0
            then bitmap[index] = 1
b ← position of leftmost zero in bitmap
return 2^b/0.77351
```

$$E[pos] \approx \log_2 \phi n \approx \log_2 0.77351 \cdot n$$
$$\sigma(pos) \approx 1.12$$

Data Stream Algorithmics

item x	$hash(x)$	$\rho(hash(x))$	bitmap
a	0110	1	01000
b	1001	0	11000
c	0111	1	11000
d	1100	0	11000
a			
b			
e	0101	1	11000
f	1010	0	11000
a			
b			

$$b = 2, n \approx 2^2/0.77351 = 5.17$$

Data Stream Algorithmics

FLAJOLET-MARTIN PROBABILISTIC COUNTING ALGORITHM

```
Init bitmap[0 ... L − 1] ← 0
for every item x in the stream
      do index = ρ(hash(x)) ▷ position of the least significant 1-bit
          if bitmap[index] = 0
             then bitmap[index] = 1
b ← position of leftmost zero in bitmap
return 2^b/0.77351
```

```
Init M ← −∞
for every item x in the stream
      do M = max(M, ρ(h(x))
b ← M + 1 ▷ position of leftmost zero in bitmap
return 2^b/0.77351
```

Data Stream Algorithmics

Stochastic Averaging

Perform m experiments in parallel

$$\sigma' = \sigma/\sqrt{m}$$

Relative accuracy is $0.78/\sqrt{m}$

HyperLogLog Counter

- the stream is divided in $m = 2^b$ substreams
- the estimation uses harmonic mean
- Relative accuracy is $1.04/\sqrt{m}$

Data Stream Algorithmics

HyperLogLog Counter

```
Init M[0 ... b − 1] ← −∞
for every item x in the stream
      do index = h_b(x)
            M[index] = max(M[index], ρ(h^b(x)))
return α_m m^2 / Σ_{j=0}^{m−1} 2^{−M[j]}
```

$$h(x) = 010011000111$$
$$h_3(x) = 010 \text{ and } h^3(x) = 011000111$$

Methodology

Paolo Boldi
Facebook Four degrees of separation

Big Data does not need big machines,
it needs big **intelligence**

Data Stream Algorithmics

Examples

1. Compute different number of pairs of IP addresses seen in a router
2. **Compute top-k most used words in tweets**

Find most frequent items

Data Stream Algorithmics

MAJORITY

```
Init counter c ← 0
for every item s in the stream
      do if counter is zero
            then pick up the item
         if item is the same
            then increment counter
            else decrement counter
```

Find the item that it is contained in more than half of the instances

Data Stream Algorithmics

FREQUENT

```
for every item i in the stream
    do if item i is not monitored
        do if < k items monitored
            then add a new item with count 1
            else if an item z whose count is zero exists
                then replace this item z by the new one
                else decrement all counters by one
        else ▷ item i is monitored
            increase its counter by one
```

Figure: Algorithm FREQUENT to find most frequent items

Data Stream Algorithmics

```
LossyCounting
for every item i in the stream
      do if item i is not monitored
            then add a new item with count 1 + Δ
            else ▷ item i is monitored
                  increase its counter by one
         if ⌊n/k⌋ ≠ Δ
            then Δ = ⌊n/k⌋
                  decrement all counters by one
                  remove items with zero counts
```

Figure: Algorithm LossyCounting to find most frequent items

Data Stream Algorithmics

SPACE SAVING

```
for every item i in the stream
    do if item i is not monitored
        do if < k items monitored
            then add a new item with count 1
            else replace the item with lower counter
                 increase its counter by one
        else ▷ item i is monitored
             increase its counter by one
```

Figure: Algorithm SPACE SAVING to find most frequent items

Data Stream Algorithmics

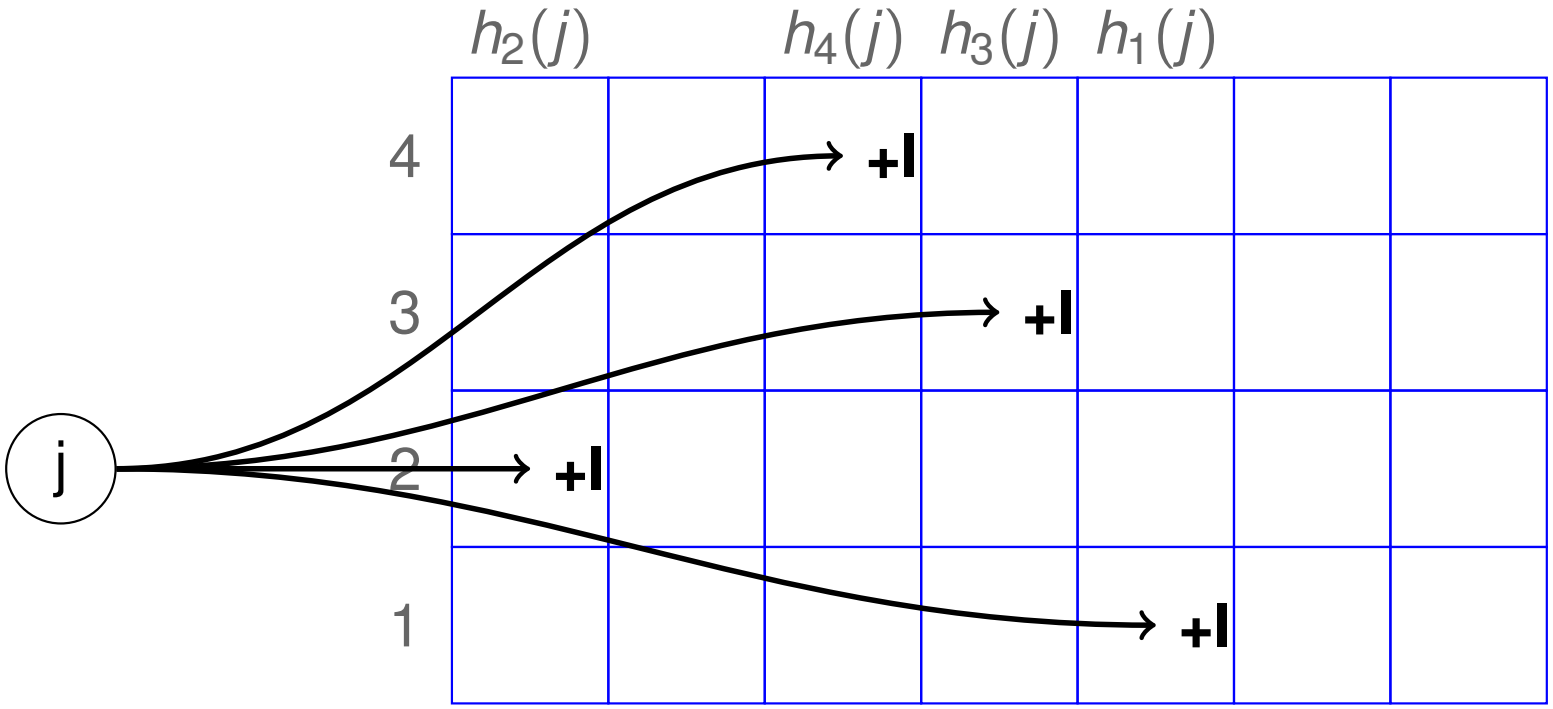


Figure: A CM sketch structure example of $\epsilon = 0.4$ and $\delta = 0.02$

Count-Min Sketch

A two dimensional array with width w and depth d

$$w = \left\lceil \frac{e}{\epsilon} \right\rceil, \qquad d = \left\lceil \ln \frac{1}{\delta} \right\rceil$$

It uses space wd with update time d

CM-Sketch computes frequency data adding and removing real values.

Count-Min Sketch

A two dimensional array with width w and depth d

$$w = \left\lceil \frac{e}{\epsilon} \right\rceil, \qquad d = \left\lceil \ln \frac{1}{\delta} \right\rceil$$

It uses space $wd = \frac{e}{\epsilon} \ln \frac{1}{\delta}$ with update time $d = \ln \frac{1}{\delta}$

CM-Sketch computes frequency data adding and removing real values.

Data Stream Algorithmics

Problem

Given a data stream, choose k items with the same probability, storing only k elements in memory.

RESERVOIR SAMPLING

Data Stream Algorithmics

RESERVOIR SAMPLING

```
for every item i in the first k items of the stream
      do store item i in the reservoir
n = k
for every item i in the stream after the first k items of the stream
      do select a random number r between 1 and n
         if r < k
            then replace item r in the reservoir with item i
         n = n + 1
```

Figure: Algorithm RESERVOIR SAMPLING

Mean and Variance

Given a stream $x_1, x_2, \ldots, x_n$

$$\bar{x}_n = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$$

$$\sigma_n^2 = \frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x}_i)^2.$$

Mean and Variance

Given a stream $x_1, x_2, \ldots, x_n$

$$s_n = \sum_{i=1}^{n} x_i, \; q_n = \sum_{i=1}^{n} x_i^2$$

$$s_n = s_{n-1} + x_n, \; q_n = q_{n-1} + x_n^2$$

$$\bar{x}_n = s_n / n$$

$$\sigma_n^2 = \frac{1}{n-1} \cdot (\sum_{i=1}^{n} x_i^2 - n\bar{x}_i^2) = \frac{1}{n-1} \cdot (q_n - s_n^2/n)$$

Data Stream Sliding Window

1011000111 $\boxed{1010101}$

Sliding Window

We can maintain simple statistics over sliding windows, using $O(\frac{1}{\epsilon}\log^2 N)$ space, where

- N is the length of the sliding window
- ϵ is the accuracy parameter

M. Datar, A. Gionis, P. Indyk, and R. Motwani.
Maintaining stream statistics over sliding windows. 2002

Data Stream Sliding Window

10110001111 $\boxed{0101011}$

Sliding Window

We can maintain simple statistics over sliding windows, using $O(\frac{1}{\epsilon}\log^2 N)$ space, where

- N is the length of the sliding window
- ϵ is the accuracy parameter

M. Datar, A. Gionis, P. Indyk, and R. Motwani.
Maintaining stream statistics over sliding windows. 2002

Data Stream Sliding Window

101100011110 $\boxed{1010111}$

Sliding Window

We can maintain simple statistics over sliding windows, using $O(\frac{1}{\epsilon}\log^2 N)$ space, where

- N is the length of the sliding window
- ϵ is the accuracy parameter

M. Datar, A. Gionis, P. Indyk, and R. Motwani.
Maintaining stream statistics over sliding windows. 2002

Data Stream Sliding Window

1011000111101 $\boxed{0101110}$

Sliding Window

We can maintain simple statistics over sliding windows, using $O(\frac{1}{\epsilon}\log^2 N)$ space, where

- N is the length of the sliding window
- ϵ is the accuracy parameter

M. Datar, A. Gionis, P. Indyk, and R. Motwani.
Maintaining stream statistics over sliding windows. 2002

Data Stream Sliding Window

$$10110001111010\ \boxed{1011101}$$

Sliding Window

We can maintain simple statistics over sliding windows, using $O(\frac{1}{\epsilon}\log^2 N)$ space, where

- N is the length of the sliding window
- ϵ is the accuracy parameter

M. Datar, A. Gionis, P. Indyk, and R. Motwani.
Maintaining stream statistics over sliding windows. 2002

Data Stream Sliding Window

101100011110101 $\boxed{0111010}$

Sliding Window

We can maintain simple statistics over sliding windows, using $O(\frac{1}{\epsilon}\log^2 N)$ space, where

- N is the length of the sliding window
- ϵ is the accuracy parameter

M. Datar, A. Gionis, P. Indyk, and R. Motwani.
Maintaining stream statistics over sliding windows. 2002

Exponential Histograms

$M = 2$

	1010101	101	11	1	1	1
Content:	4	2	2	1	1	1
Capacity:	7	3	2	1	1	1

	1010101	101	11	11	1
Content:	4	2	2	2	1
Capacity:	7	3	2	2	1

	1010101	10111	11	1
Content:	4	4	2	1
Capacity:	7	5	2	1

Exponential Histograms

	1010101	101	11	1	1
Content:	4	2	2	1	1
Capacity:	7	3	2	1	1

Error $<$ content of the last bucket W/M

$\epsilon = 1/(2M)$ and $M = 1/(2\epsilon)$

$M \cdot \log(W/M)$ buckets to maintain the data stream sliding window

Exponential Histograms

	1010101	101	11	1	1
Content:	4	2	2	1	1
Capacity:	7	3	2	1	1

To give answers in $O(1)$ time,
it maintain three counters LAST, TOTAL and VARIANCE.

$M \cdot \log(W/M)$ buckets to maintain the data stream sliding window

Concept Drift

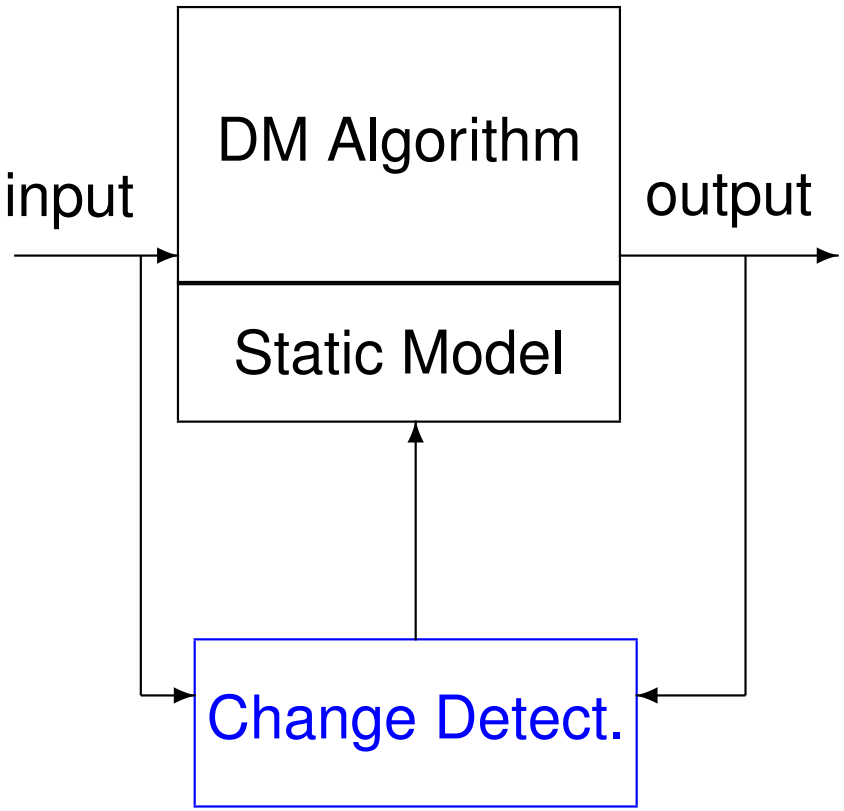
Albert Bifet (@abifet)

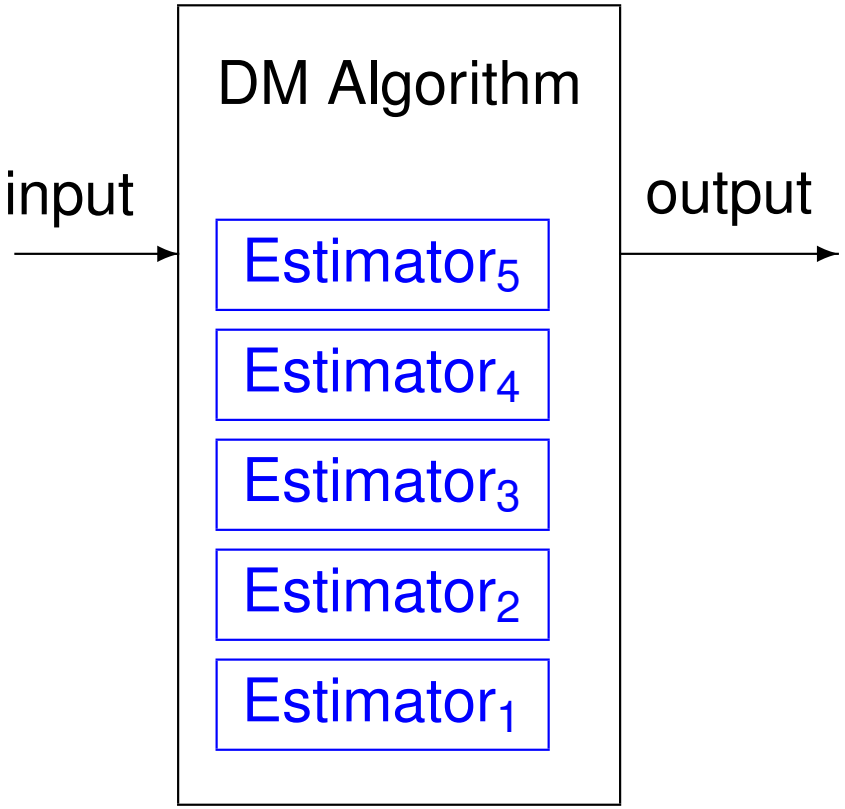
Paris, 2 December 2016
albert.bifet@telecom-paristech.fr

Data Streams

Big Data & Real Time

Data Mining Algorithms with Concept Drift.





Introduction.

Problem

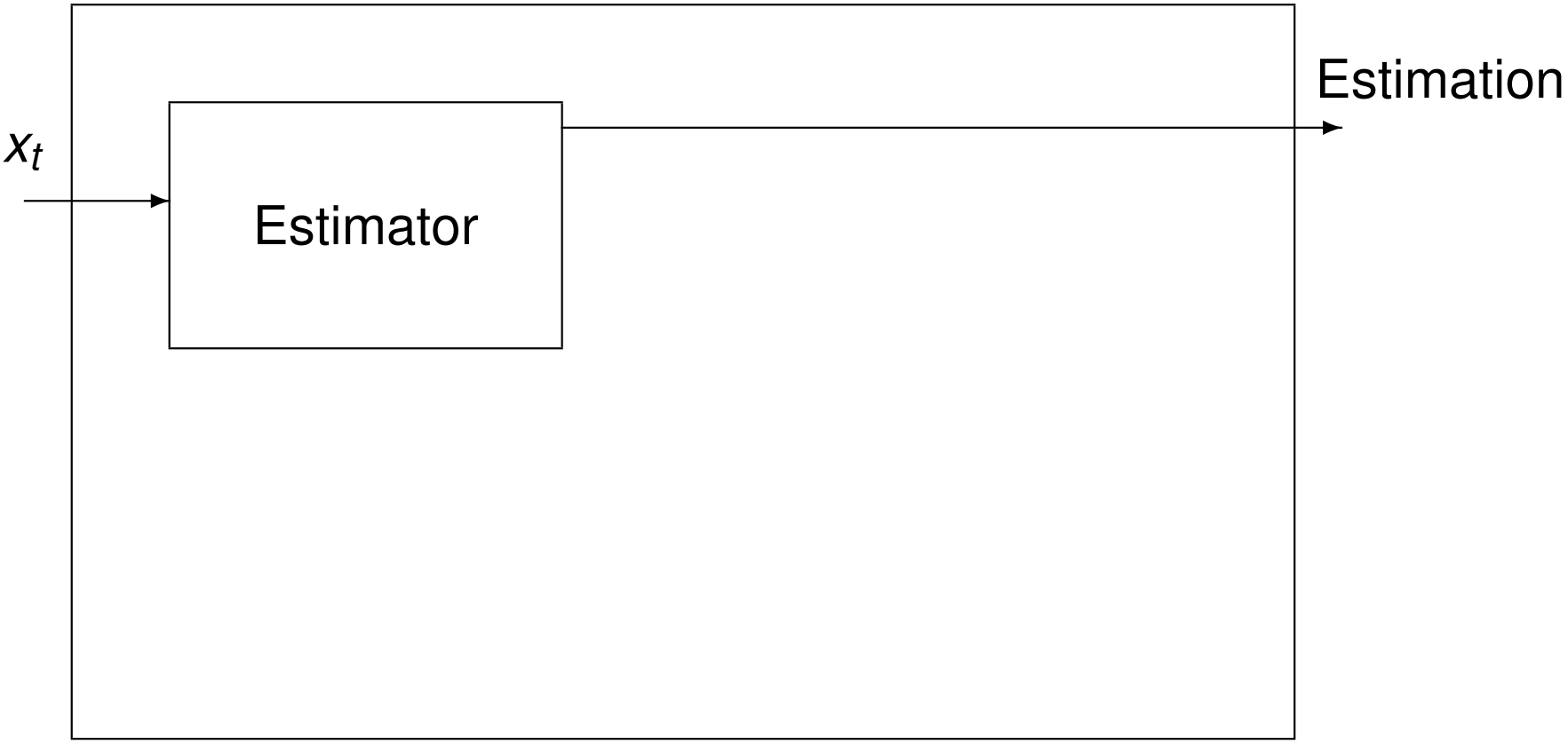
Given an input sequence $x_1, x_2, \cdots, x_t$ we want to output at instant t an alarm signal if there is a distribution change and also a prediction $\widehat{x}_{t+1}$ minimizing prediction error:

$$|\widehat{x}_{t+1} - x_{t+1}|$$

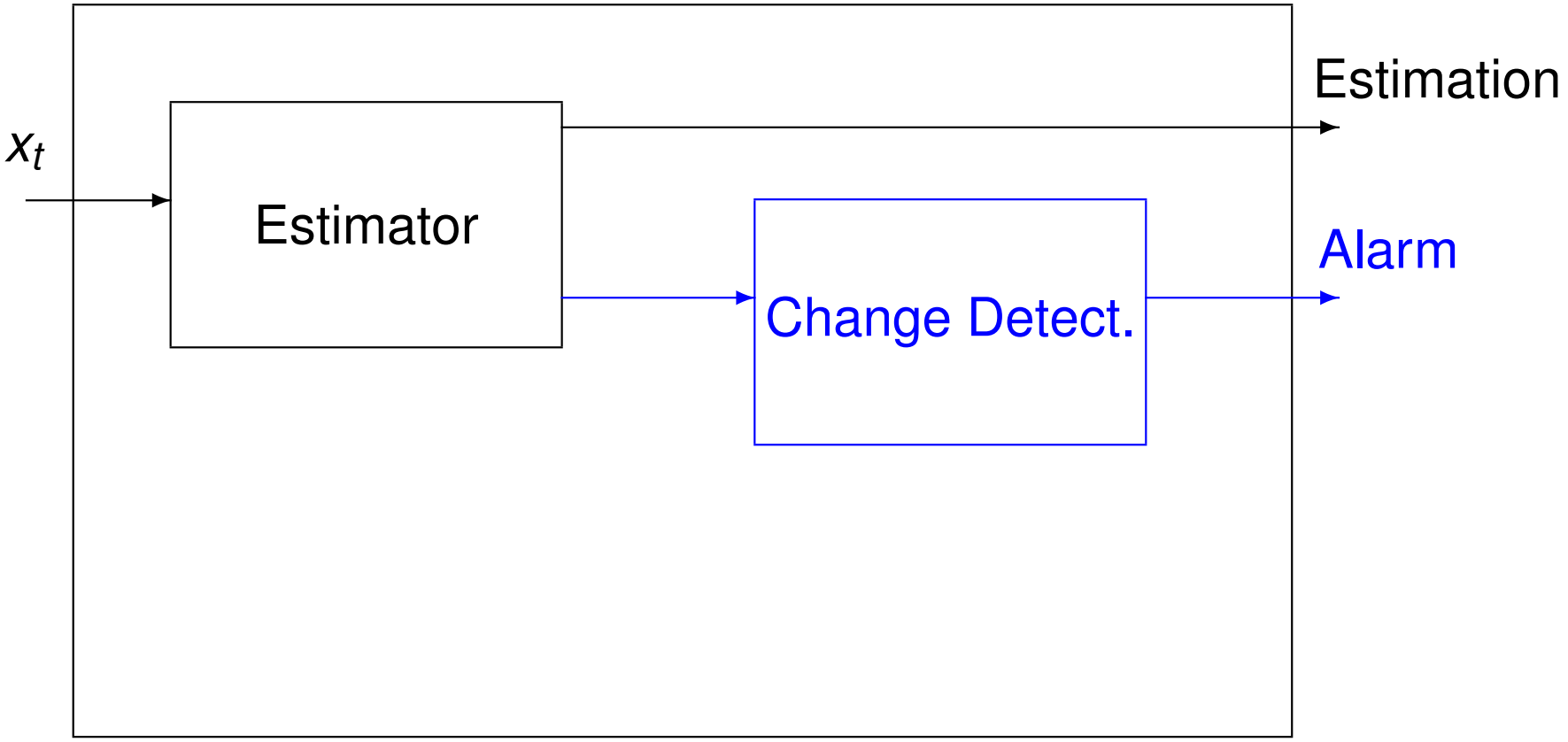
Outputs

- an estimation of some important parameters of the input distribution, and
- a signal alarm indicating that distribution change has recently occurred.

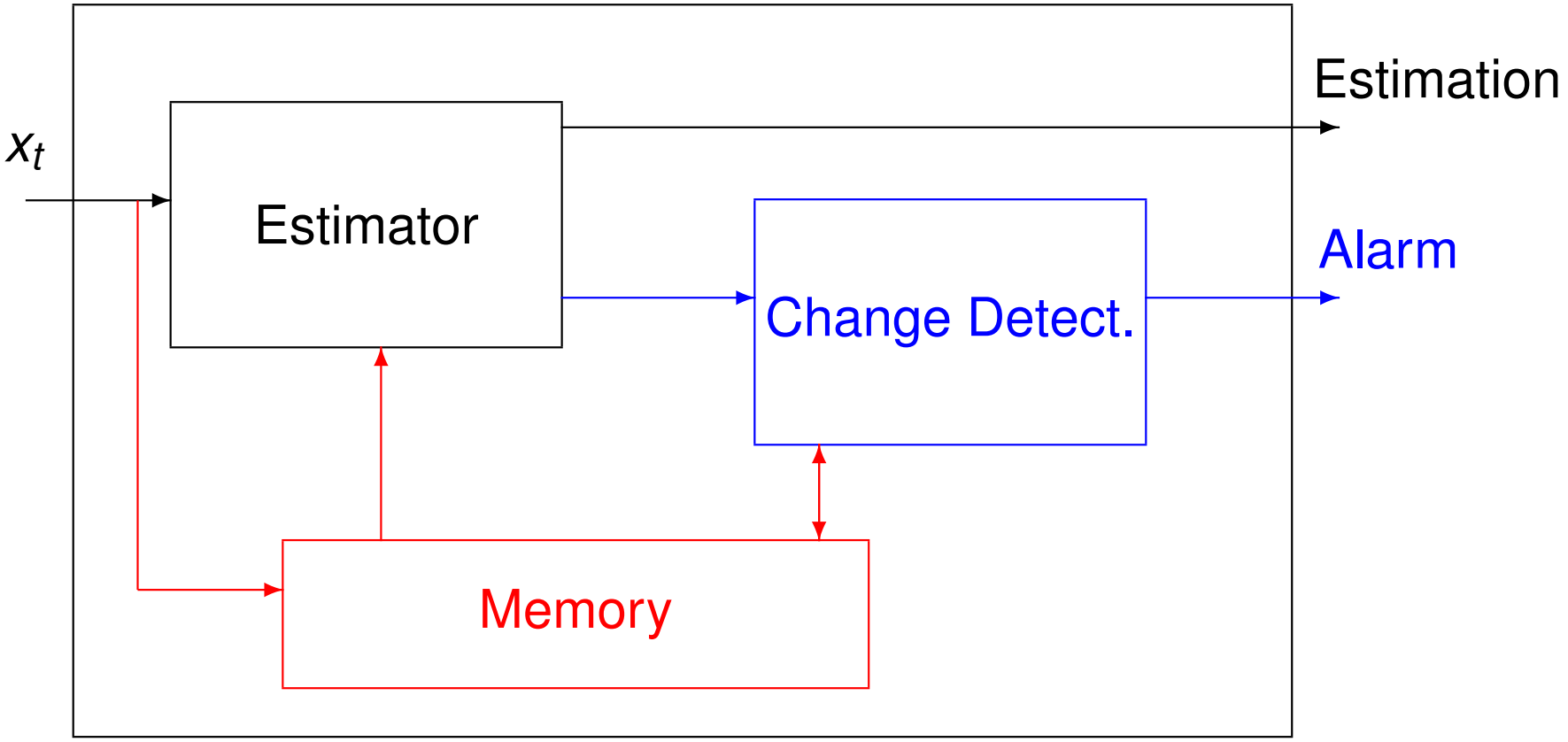
Change Detectors and Predictors



Change Detectors and Predictors



Change Detectors and Predictors



Concept Drift Evaluation

Mean Time between False Alarms (MTFA)

Mean Time to Detection (MTD)

Missed Detection Rate (MDR)

Average Run Length (ARL(θ))

The design of a change detector is a compromise between detecting true changes and avoiding false alarms.

Data Stream Algorithmics

- High accuracy in the prediction
- Low mean time to detection (MTD), false positive rate (FAR) and missed detection rate (MDR)
- Low computational cost: minimum space and time needed
- Theoretical guarantees
- No parameters needed

Main properties of an optimal change detector and predictor system.

The CUSUM Test

- The cumulative sum (CUSUM algorithm), gives an alarm when the mean of the input data is significantly different from zero.
- The CUSUM test is memoryless, and its accuracy depends on the choice of parameters υ and h.

$$g_0 = 0, \qquad g_t = \max\left(0, g_{t-1} + \epsilon_t - \upsilon\right)$$

$$\text{if } g_t > h \text{ then alarm and } g_t = 0$$

Cumulative sum algorithm (CUSUM).

Page Hinckley Test

- The CUSUM test

$$g_0 = 0, \qquad g_t = \max\left(0, g_{t-1} + \epsilon_t - \upsilon\right)$$

$$\text{if } g_t > h \text{ then alarm and } g_t = 0$$

- The Page Hinckley Test

$$g_0 = 0, \qquad g_t = g_{t-1} + (\epsilon_t - \upsilon)$$

$$G_t = \min(g_t)$$

$$\text{if } g_t - G_t > h \text{ then alarm and } g_t = 0$$

Geometric Moving Average Test

- The CUSUM test

$$g_0 = 0, \qquad g_t = \max\,(0, g_{t-1} + \epsilon_t - \upsilon)$$

$$\text{if } g_t > h \text{ then alarm and } g_t = 0$$

- The Geometric Moving Average Test

$$g_0 = 0, \qquad g_t = \lambda g_{t-1} + (1 - \lambda)\epsilon_t$$

$$\text{if } g_t > h \text{ then alarm and } g_t = 0$$

The forgetting factor λ is used to give more or less weight to the last data arrived.

Statistical test

$$\hat{\mu}_0 - \hat{\mu}_1 \in N(0, \sigma_0^2 + \sigma_1^2), \text{ under } H_0$$

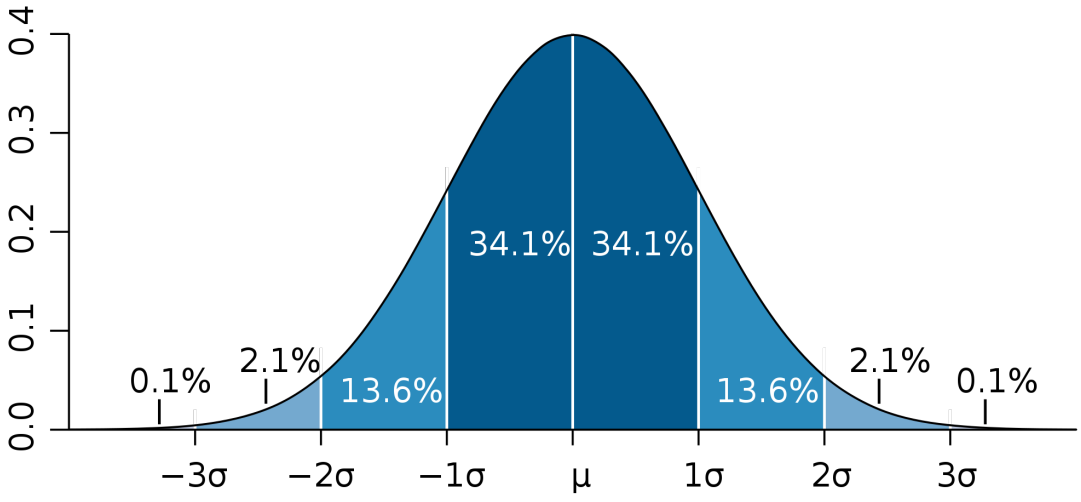
Example: Probability of false alarm of 5%

$$\Pr\left(\frac{|\hat{\mu}_0 - \hat{\mu}_1|}{\sqrt{\sigma_0^2 + \sigma_1^2}} > h\right) = 0.05$$

As $P(X < 1.96) = 0.975$ the test becomes

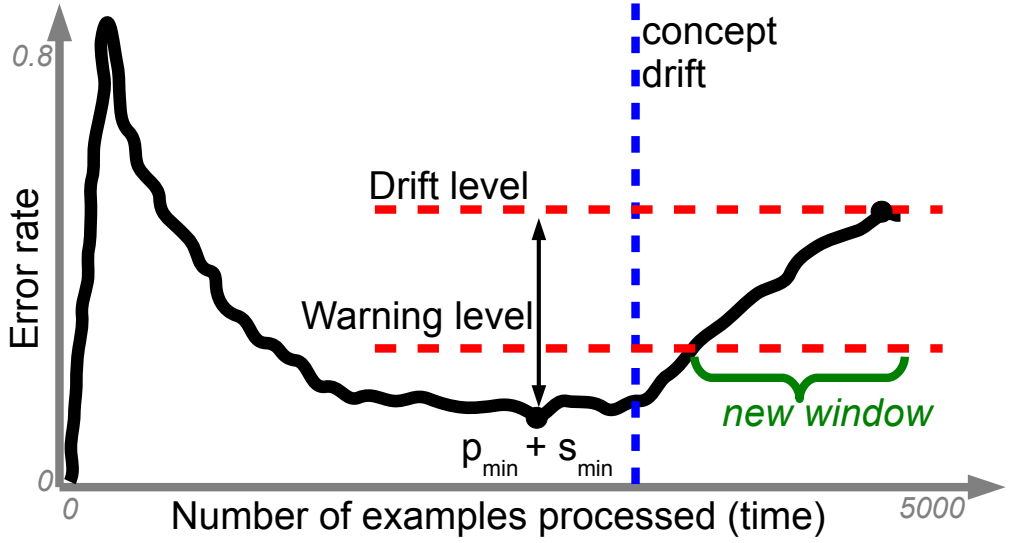
$$\frac{(\hat{\mu}_0 - \hat{\mu}_1)^2}{\sigma_0^2 + \sigma_1^2} > 1.96^2$$

Concept Drift



6 sigma

Concept Drift



Statistical Drift Detection Method
(Joao Gama et al. 2004)

ADWIN: Adaptive Data Stream Sliding Window

Let $W = \boxed{101010110111111}$

- Equal & fixed size subwindows: $\boxed{1010}1011011\boxed{1111}$
- Equal size adjacent subwindows: $1010101\boxed{1011}\boxed{1111}$
- Total window against subwindow: $\boxed{\boxed{10101011011}1111}$
- ADWIN: All adjacent subwindows:

1	01010110111111
1010	10110111111
1010101	10111111
1010101101	11111
10101011011111	1

Data Stream Sliding Window

101100011110101 $\boxed{0111010}$

Sliding Window

We can maintain simple statistics over sliding windows, using $O(\frac{1}{\epsilon}\log^2 N)$ space, where

- N is the length of the sliding window
- ϵ is the accuracy parameter

M. Datar, A. Gionis, P. Indyk, and R. Motwani.
Maintaining stream statistics over sliding windows. 2002

Exponential Histograms

$M = 2$

	1010101	101	11	1	1	1
Content:	4	2	2	1	1	1
Capacity:	7	3	2	1	1	1

	1010101	101	11	11	1
Content:	4	2	2	2	1
Capacity:	7	3	2	2	1

	1010101	10111	11	1
Content:	4	4	2	1
Capacity:	7	5	2	1

Exponential Histograms

	1010101	101	11	1	1
Content:	4	2	2	1	1
Capacity:	7	3	2	1	1

Error $<$ content of the last bucket W/M

$\epsilon = 1/(2M)$ and $M = 1/(2\epsilon)$

$M \cdot \log(W/M)$ buckets to maintain the data stream sliding window

Exponential Histograms

	1010101	101	11	1	1
Content:	4	2	2	1	1
Capacity:	7	3	2	1	1

To give answers in $O(1)$ time,
it maintain three counters LAST, TOTAL and VARIANCE.

$M \cdot \log(W/M)$ buckets to maintain the data stream sliding window

Algorithm ADaptive Sliding WINdow

ADWIN: ADAPTIVE WINDOWING ALGORITHM

```
Initialize W as an empty list of buckets
Initialize WIDTH, VARIANCE and TOTAL
for each t > 0
      do SETINPUT(x_t, W)
            output μ̂_W as TOTAL/WIDTH and ChangeAlarm
```

SETINPUT(item e, List W)

```
INSERTELEMENT(e, W)
repeat  DELETEELEMENT(W)
   until |μ̂_{W_0} − μ̂_{W_1}| < ε_cut holds
      for every split of W into W = W_0 · W_1
```

Algorithm ADaptive Sliding WINdow

INSERTELEMENT(item e, List W)

1 create a new bucket b with content e and capacity 1
2 $W \leftarrow W \cup \{b\}$ (i.e., add e to the head of W)
3 update WIDTH, VARIANCE and TOTAL
4 COMPRESSBUCKETS(W)

DELETEELEMENT(List W)

1 remove a bucket from tail of List W
2 update WIDTH, VARIANCE and TOTAL
3 ChangeAlarm $\leftarrow$ **true**

Algorithm ADaptive Sliding WINdow

```
COMPRESSBUCKETS(List W)
Traverse the list of buckets in increasing order
    do If there are more than M buckets of the same capacity
        do merge buckets
            COMPRESSBUCKETS(sublist of W not traversed)
```

Algorithm ADaptive Sliding WINdow

Theorem

At every time step we have:

1. (False positive rate bound). *If μ_t remains constant within W, the probability that* ADWIN *shrinks the window at this step is at most δ.*
2. (False negative rate bound). *Suppose that for* some *partition of W in two parts $W_0 W_1$ (where W_1 contains the most recent items) we have $|\mu_{W_0} - \mu_{W_1}| > 2\epsilon_{cut}$. Then with probability $1 - \delta$* ADWIN *shrinks W to W_1, or shorter.*

ADWIN tunes itself to the data stream at hand, with no need for the user to hardwire or precompute parameters.

Algorithm ADaptive Sliding WINdow

ADWIN using a Data Stream Sliding Window Model,

- can provide the exact counts of 1's in $O(1)$ time per point.
- tries $O(\log W)$ cutpoints
- uses $O(\frac{1}{\epsilon} \log W)$ memory words
- the processing time per example is $O(\log W)$ (amortized and worst-case).

Sliding Window Model

	1010101	101	11	1	1
Content:	4	2	2	1	1
Capacity:	7	3	2	1	1

Evaluation

Albert Bifet (@abifet)

Paris, 2 December 2016

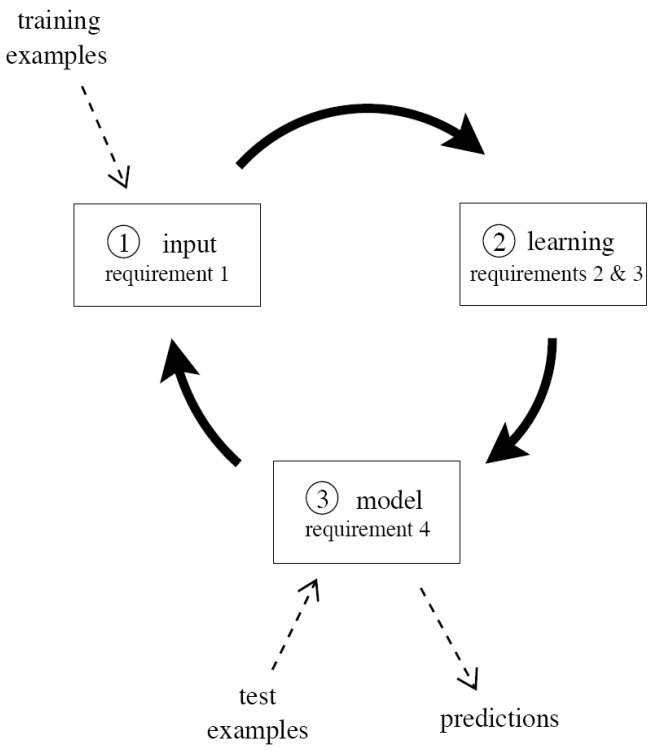
albert.bifet@telecom-paristech.fr

Data Streams

Big Data & Real Time

Data stream classification cycle

1. Process an example at a time, and inspect it only once (at most)
2. Use a limited amount of memory
3. Work in a limited amount of time
4. Be ready to predict at any point



Evaluation

1. Error estimation: *Hold-out or Prequential*
2. Evaluation performance measures: *Accuracy or κ-statistic*
3. Statistical significance validation: *MacNemar or Nemenyi test*

Evaluation Framework

Error Estimation

Data available for testing

- Holdout an independent test set
- Apply the current decision model to the test set, at regular time intervals
- The loss estimated in the holdout is an unbiased estimator

Holdout Evaluation

1. Error Estimation

No data available for testing

- The error of a model is computed from the sequence of examples.
- For each example in the stream, the actual model makes a prediction, and then uses it to update the model.

Prequential or
Interleaved-Test-Then-Train

1. Error Estimation

Hold-out or Prequential?

Hold-out is more accurate, but needs data for testing.

- Use prequential to approximate Hold-out
- Estimate accuracy using sliding windows or fading factors

Hold-out or Prequential or
Interleaved-Test-Then-Train

2. Evaluation performance measures

	Predicted Class+	Predicted Class-	Total
Correct Class+	75	8	83
Correct Class-	7	10	17
Total	82	18	100

Table: Simple confusion matrix example

- Accuracy = $\frac{75}{100} + \frac{10}{100} = \frac{75}{83}\frac{83}{100} + \frac{10}{17}\frac{17}{100} = 85\%$
- Arithmetic mean = $(\frac{75}{83} + \frac{10}{17})/2 = 74.59\%$
- Geometric mean = $\sqrt{\frac{75}{83}\frac{10}{17}} = 72.90\%$

2. Performance Measures with Unbalanced Classes

	Predicted Class+	Predicted Class-	Total
Correct Class+	75	8	83
Correct Class-	7	10	17
Total	82	18	100

Table: Simple confusion matrix example

	Predicted Class+	Predicted Class-	Total
Correct Class+	68.06	14.94	83
Correct Class-	13.94	3.06	17
Total	82	18	100

Table: Confusion matrix for chance predictor

2. Performance Measures with Unbalanced Classes

Kappa Statistic

- p_0: classifier's prequential accuracy
- p_c: probability that a chance classifier makes a correct prediction.
- κ statistic

$$\kappa = \frac{p_0 - p_c}{1 - p_c}$$

- $\kappa = 1$ if the classifier is always correct
- $\kappa = 0$ if the predictions coincide with the correct ones as often as those of the chance classifier

Forgetting mechanism for estimating prequential kappa

Sliding window of size w with the most recent observations

3. Statistical significance validation (2 Classifiers)

	Classifier A Class+	Classifier A Class-	Total
Classifier B Class+	c	a	c+a
Classifier B Class-	b	d	b+d
Total	c+b	a+d	a+b+c+d

$$M = |a - b - 1|^2/(a + b)$$

The test follows the χ^2 distribution. At 0.99 confidence it rejects the null hypothesis (the performances are equal) if $M > 6.635$.

McNemar test

3. Statistical significance validation ($>$ 2 Classifiers)

Two classifiers are performing differently if the corresponding average ranks differ by at least the critical difference

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

- k is the number of learners, N is the number of datasets,
- critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$.

Nemenyi test

3. Statistical significance validation (> 2 Classifiers)

Two classifiers are performing differently if the corresponding average ranks differ by at least the critical difference

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

- k is the number of learners, N is the number of datasets,
- critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$.

# classifiers	2	3	4	5	6	7
$q_{0.05}$	1.960	2.343	2.569	2.728	2.850	2.949
$q_{0.10}$	1.645	2.052	2.291	2.459	2.589	2.693

Table: Critical values for the Nemenyi test

Cost Evaluation Example

	Accuracy	Time	Memory
Classifier A	70%	100	20
Classifier B	80%	20	40

Which classifier is performing better?

RAM-Hours

RAM-Hour

Every GB of RAM deployed for 1 hour

Cloud Computing Rental Cost Options

Cost Evaluation Example

	Accuracy	Time	Memory	**RAM-Hours**
Classifier A	70%	100	20	**2,000**
Classifier B	80%	20	40	**800**

Which classifier is performing better?

Evaluation

1. Error estimation: *Hold-out or Prequential*
2. Evaluation performance measures: *Accuracy or κ-statistic*
3. Statistical significance validation: *MacNemar or Nemenyi test*
4. **Resources needed**: *time and memory or RAM-Hours*

Evaluation Framework

Ensemble Methods

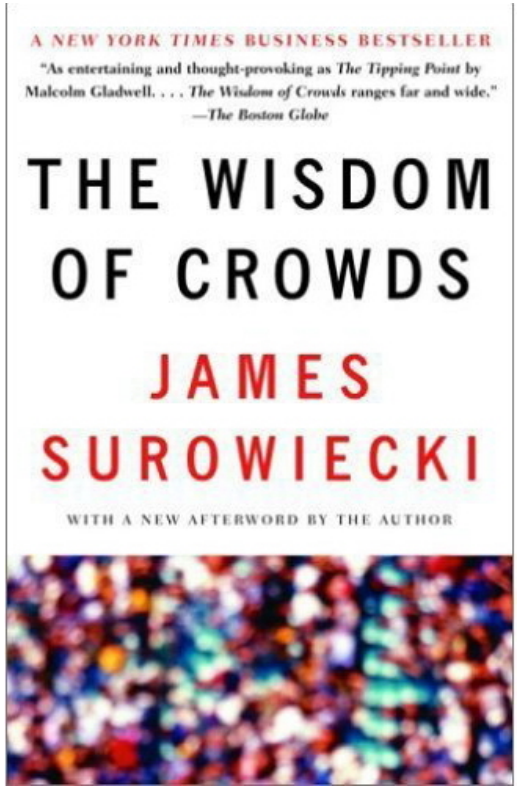
Albert Bifet (@abifet)

Paris, 6 December 2016
albert.bifet@telecom-paristech.fr

Data Streams

Big Data & Real Time

Ensemble Learning: The Wisdom of Crowds



Diversity of opinion, Independence
Decentralization, Aggregation

Bagging

Example

Dataset of 4 Instances : A, B, C, D

Classifier 1: B, A, C, B
Classifier 2: D, B, A, D
Classifier 3: B, A, C, B
Classifier 4: B, C, B, B
Classifier 5: D, C, A, C

Bagging builds a set of M base models, with a bootstrap sample created by drawing random samples with replacement.

Bagging

Example

Dataset of 4 Instances : A, B, C, D

Classifier 1: A, B, B, C
Classifier 2: A, B, D, D
Classifier 3: A, B, B, C
Classifier 4: B, B, B, C
Classifier 5: A, C, C, D

Bagging builds a set of M base models, with a bootstrap sample created by drawing random samples with replacement.

Bagging

Example

Dataset of 4 Instances : A, B, C, D

Classifier 1: A, B, B, C: A(1) B(2) C(1) D(0)
Classifier 2: A, B, D, D: A(1) B(1) C(0) D(2)
Classifier 3: A, B, B, C: A(1) B(2) C(1) D(0)
Classifier 4: B, B, B, C: A(0) B(3) C(1) D(0)
Classifier 5: A, C, C, D: A(1) B(0) C(2) D(1)

Each base model's training set contains each of the original training example K times where $P(K = k)$ follows a binomial distribution.

Bagging

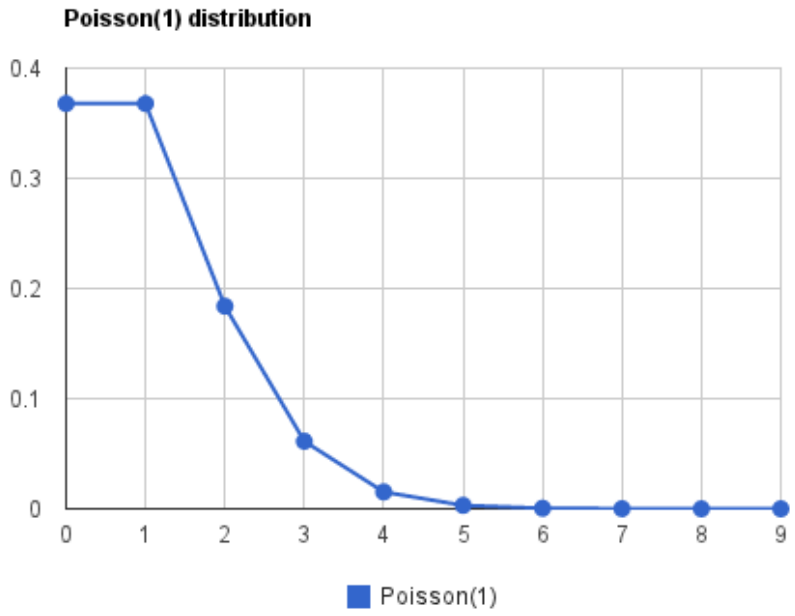


Figure: Poisson(1) Distribution.

Each base model's training set contains each of the original training example K times where $P(K = k)$ follows a binomial distribution.

Oza and Russell's *Online Bagging* for M models

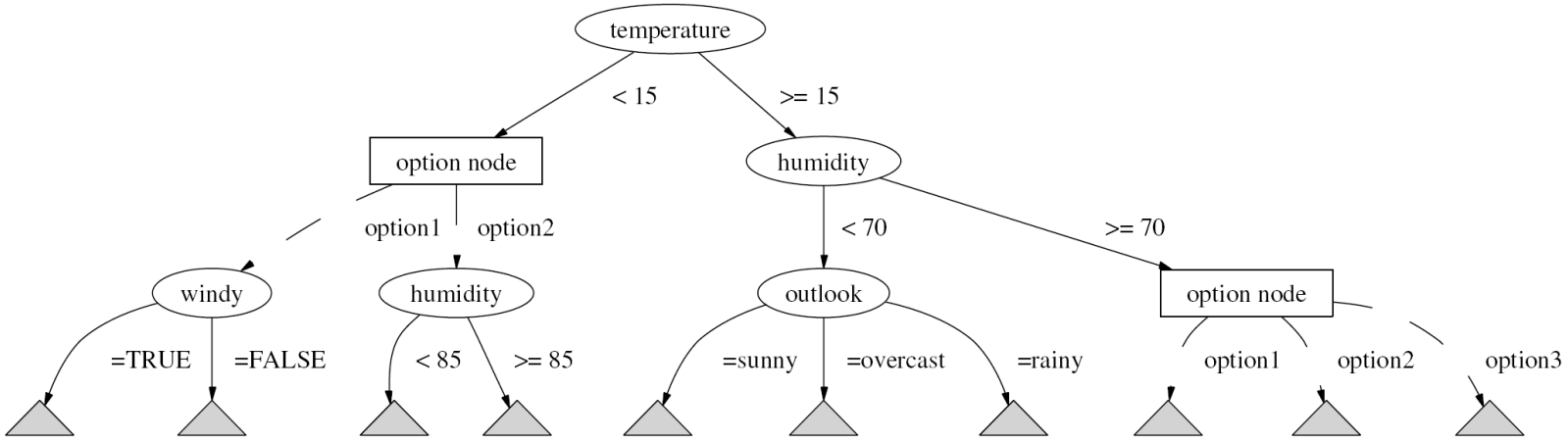
Initialize base models h_m for all $m \in \{1, 2, ..., M\}$
for all training examples **do**
  for $m = 1, 2, ..., M$ **do**
    Set $w = Poisson(1)$
    Update h_m with the current example with weight w

anytime output:
return hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} I(h_t(x) = y)$

Hoeffding Option Tree

Hoeffding Option Trees

Regular Hoeffding tree containing additional option nodes that allow several tests to be applied, leading to multiple Hoeffding trees as separate paths.



Random Forests (Breiman, 2001)

Adding randomization to decision trees

- the input training set is obtained by sampling with replacement, like Bagging
- the nodes of the tree only may use a fixed number of random attributes to split
- the trees are grown without pruning

Accuracy Weighted Ensemble

Mining concept-drifting data streams using ensemble classifiers. Wang et al. 2003

- Process chunks of instances of size W
- Builds a new classifier for each chunk
- Removes old classifier
- Weight each classifier using error

$$w_i = MSE_r - MSE_i$$

where

$$MSE_r = \sum_c p(c)(1 - p(c))^2$$

and

$$MSE_i = \frac{1}{|S_n|} \sum_{(x,c) \in S_n} (1 - f_c^i(x))^2$$

ADWIN Bagging

ADWIN

An adaptive sliding window whose size is recomputed online according to the rate of change observed.

ADWIN has rigorous guarantees (theorems)

- On ratio of false positives and negatives
- On the relation of the size of the current window and change rates

ADWIN Bagging

When a change is detected, the worst classifier is removed and a new classifier is added.

ADWIN *Bagging* for M models

Initialize base models h_m for all $m \in \{1, 2, ..., M\}$
for all training examples **do**
  for $m = 1, 2, ..., M$ **do**
    Set $w = Poisson(1)$
    Update h_m with the current example with weight w
    if ADWIN detects change in error of one of the classifiers **then**
      Replace classifier with higher error with a new one

anytime output:
return hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} I(h_t(x) = y)$

Leveraging Bagging for Evolving Data Streams

Randomization as a powerful tool to increase accuracy and diversity

There are three ways of using randomization:

- Manipulating the input data
- Manipulating the classifier algorithms
- Manipulating the output targets

Input Randomization

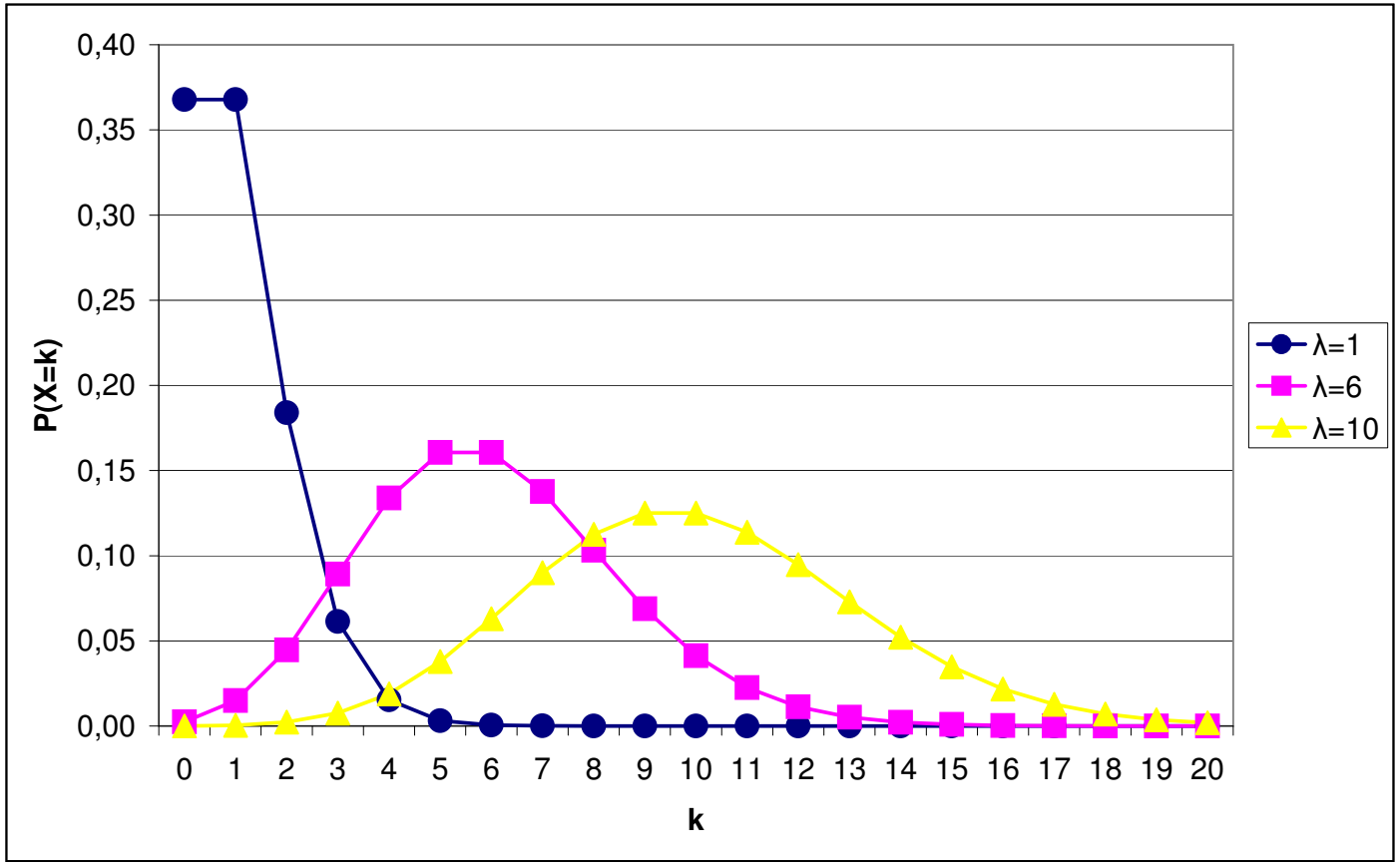


Figure: Poisson Distribution.

ECOC Output Randomization

Table: Example matrix of random output codes for 3 classes and 6 classifiers

	Class 1	Class 2	Class 3
Classifier 1	0	0	1
Classifier 2	0	1	1
Classifier 3	1	0	0
Classifier 4	1	1	0
Classifier 5	1	0	1
Classifier 6	0	1	0

Leveraging Bagging for Evolving Data Streams

Leveraging Bagging

- Using $Poisson(\lambda)$

Leveraging Bagging MC

- Using $Poisson(\lambda)$ and Random Output Codes

Fast Leveraging Bagging ME

- if an instance is misclassified: weight = 1
- if not: weight = $e_T/(1 - e_T)$,

Empirical evaluation

	Accuracy	RAM-Hours
Hoeffding Tree	74.03%	0.01
Online Bagging	77.15%	2.98
ADWIN Bagging	79.24%	1.48
Leveraging Bagging	85.54%	20.17
Leveraging Bagging MC	85.37%	22.04
Leveraging Bagging ME	80.77%	0.87

Leveraging Bagging

- Leveraging Bagging
  - Using $Poisson(\lambda)$
- Leveraging Bagging MC
  - Using $Poisson(\lambda)$ and Random Output Codes
- Leveraging Bagging ME
  - Using weight 1 if misclassified, otherwise $e_T/(1 - e_T)$

Boosting

The strength of Weak Learnability, Schapire 90

A boosting algorithm transforms a weak learner into a strong one

Boosting

A formal description of Boosting (Schapire)

- given a training set $(x_1, y_1), \ldots, (x_m, y_m)$
- $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$
- for $t = 1, \ldots, T$
  - construct distribution D_t
  - find weak classifier

$$h_t : X \implies \{-1, +1\}$$

  with small error $\epsilon_t = \Pr_{D_t}[h_t(x_i) \neq y_i]$ on D_t
- output final classifier

Boosting

Oza and Russell's *Online Boosting*

Initialize base models h_m for all $m \in \{1, 2, ..., M\}$, $\lambda_m^{sc} = 0$, $\lambda_m^{sw} = 0$
for all training examples **do**
    Set "weight" of example $\lambda_d = 1$
    for $m = 1, 2, ..., M$ **do**
        Set $k = Poisson(\lambda_d)$
        for $n = 1, 2, ..., k$ **do**
            Update h_m with the current example
        if h_m correctly classifies the example **then**
            $\lambda_m^{sc} \leftarrow \lambda_m^{sc} + \lambda_d$
            $\epsilon_m = \frac{\lambda_m^{sw}}{\lambda_m^{sw} + \lambda_m^{sc}}$
            $\lambda_d \leftarrow \lambda_d \left(\frac{1}{2(1-\epsilon_m)} \right)$ Decrease λ_d
        else
            $\lambda_m^{sw} \leftarrow \lambda_m^{sw} + \lambda_d$
            $\epsilon_m = \frac{\lambda_m^{sw}}{\lambda_m^{sw} + \lambda_m^{sc}}$
            $\lambda_d \leftarrow \lambda_d \left(\frac{1}{2\epsilon_m} \right)$ Increase λ_d
anytime output:
return hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_{m: h_m(x) = y} - \log \epsilon_m / (1 - \epsilon_m)$

Stacking

Use a classifier to combine predictions of base classifiers

- Example: use a perceptron to do stacking

Restricted Hoeffding Trees

Trees for all possible attribute subsets of size k

- $\binom{m}{k}$ subsets
- $\binom{m}{k} = \frac{m!}{k!(m-k)!} = \binom{m}{m-k}$

Example for 10 attributes

$$\binom{10}{1} = 10 \quad \binom{10}{2} = 45 \quad \binom{10}{3} = 120$$

$$\binom{10}{4} = 210 \quad \binom{10}{5} = 252$$

Classification

Jesse Read

[D&K] IoT Stream Data Mining 2017-2018

December 13, 2017

Outline

1. Introduction
2. Applications
3. Probabilistic Models
4. Evaluation Metrics
5. Label Dependence and Structure
6. Inference in Probabilistic Models
7. (Multi-output) Neural Networks
8. (Multi-output) k-Nearest Neighbours
9. (Multi-output) Decision Trees
10. Big Multi-Labelled Data
11. Summary

Classification

$\mathbf{x} =$

Binary classification

$$y \in \{\texttt{non_sunset}, \text{sunset}\}$$

Classification

$\mathbf{x} =$

Multi-class classification

$$y \in \{\text{sunset}, \texttt{people}, \texttt{foliage}, \texttt{beach}, \texttt{urban}, \texttt{field}\}$$

Classification

$\mathbf{x} =$

Multi-label classification

$$\mathbf{y} \subseteq \{\text{sunset}, \texttt{people}, \text{foliage}, \texttt{beach}, \texttt{urban}, \texttt{field}\}$$
$$\in \{0,1\}^6 \qquad = [1, 0, 1, 0, 0, 0]$$

i.e., multiple labels per instance instead of a single label.

Single-label vs. Multi-label

	$K = 2$	$K > 2$
$L = 1$ (single-label)	**binary**	**multi-class**
$L > 1$ (multi-label)	**multi-label**	**multi-output**[†]

[†] aka multi-objective, multi-target, multi-dimensional, multi-label.

Figure: For L labels (target variables), each of K values.

Single-label vs. Multi-label

Table: Single-label $Y \in \{0, 1\}$

X_1	X_2	X_3	X_4	X_5	Y
1	0.1	3	1	0	0
0	0.9	1	0	1	1
0	0.0	1	1	0	0
1	0.8	2	0	1	1
1	0.0	2	0	1	0
0	0.0	3	1	1	?

Table: Multi-label $Y \subseteq \{\lambda_1, \ldots, \lambda_L\}$

X_1	X_2	X_3	X_4	X_5	Y
1	0.1	3	1	0	$\{\lambda_2, \lambda_3\}$
0	0.9	1	0	1	$\{\lambda_1\}$
0	0.0	1	1	0	$\{\lambda_2\}$
1	0.8	2	0	1	$\{\lambda_1, \lambda_4\}$
1	0.0	2	0	1	$\{\lambda_4\}$
0	0.0	3	1	1	?

Single-label vs. Multi-label

Table: Single-label $Y \in \{0, 1\}$

X_1	X_2	X_3	X_4	X_5	Y
1	0.1	3	1	0	0
0	0.9	1	0	1	1
0	0.0	1	1	0	0
1	0.8	2	0	1	1
1	0.0	2	0	1	0
0	0.0	3	1	1	?

Table: Multi-label $[Y_1, \ldots, Y_L] \in 2^L$

X_1	X_2	X_3	X_4	X_5	Y_1	Y_2	Y_3	Y_4
1	0.1	3	1	0	0	1	1	0
0	0.9	1	0	1	1	0	0	0
0	0.0	1	1	0	0	1	0	0
1	0.8	2	0	1	1	0	0	1
1	0.0	2	0	1	0	0	0	1
0	0.0	3	1	1	?	?	?	?

Notation

Given a set of L labels, e.g.,

$$\mathcal{L} = \{\texttt{sunset}, \texttt{people}, \texttt{foliage}, \texttt{beach}, \texttt{urban}, \texttt{field}\}$$

($|\mathcal{L}| = L = 6$ in this example) we wish to obtain a model h, that can take some input,

$\mathbf{x}_i =$

and produce predictions,

$$\begin{aligned}\hat{\mathbf{y}}_i &= h(\mathbf{x}_i) \\ &= [1, 0, 1, 0, 0, 0] \Leftrightarrow \{\text{sunset}, \text{foliage}\} \\ &\in \{0, 1\}^6 \Leftrightarrow \hat{Y}_i \subseteq \mathcal{L}\end{aligned}$$

i.e., multiple labels per instance instead of a single label.

Outline

1. Introduction
2. Applications
3. Probabilistic Models
4. Evaluation Metrics
5. Label Dependence and Structure
6. Inference in Probabilistic Models
7. (Multi-output) Neural Networks
8. (Multi-output) k-Nearest Neighbours
9. (Multi-output) Decision Trees
10. Big Multi-Labelled Data
11. Summary

Text Categorization and Tag Recommendation

For example, the IMDb dataset: Textual movie plot summaries associated with genres (labels). Also: Bookmarks, Bibtex, del.icio.us datasets.

Text Categorization and Tag Recommendation

For example, the IMDb dataset: Textual movie plot summaries associated with genres (labels). Also: Bookmarks, Bibtex, del.icio.us datasets.

i	*abandoned* X_1	*accident* X_2	... ...	*violent* X_{1000}	*wedding* X_{1001}	`horror` Y_1	`romance` Y_2	... ...	`comedy` Y_{27}	`action` Y_{28}
1	1	0	...	0	1	0	1	...	0	0
2	0	1	...	1	0	1	0	...	0	0
3	0	0	...	0	1	0	1	...	0	0
4	1	1	...	0	1	1	0	...	0	1
5	1	1	...	0	1	0	1	...	0	1
⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮	⋱	⋮	⋮
120919	1	1	...	0	0	0	0	...	0	1

Labelling E-mails

Enron, e-mails messages made public from the Enron corporation.

- For example, the *Enron* e-mails multi-labelled to 53 categories by the *UC Berkeley Enron Email Analysis Project*

Company Business, Strategy, etc.
Purely Personal
Empty Message
Forwarded email(s)
...
company image – current
...
Jokes, humor (related to business)
...
Emotional tone: worry / anxiety
Emotional tone: sarcasm
...
Emotional tone: shame
Company Business, Strategy, etc.

Labelling Images

Images are labelled to indicate

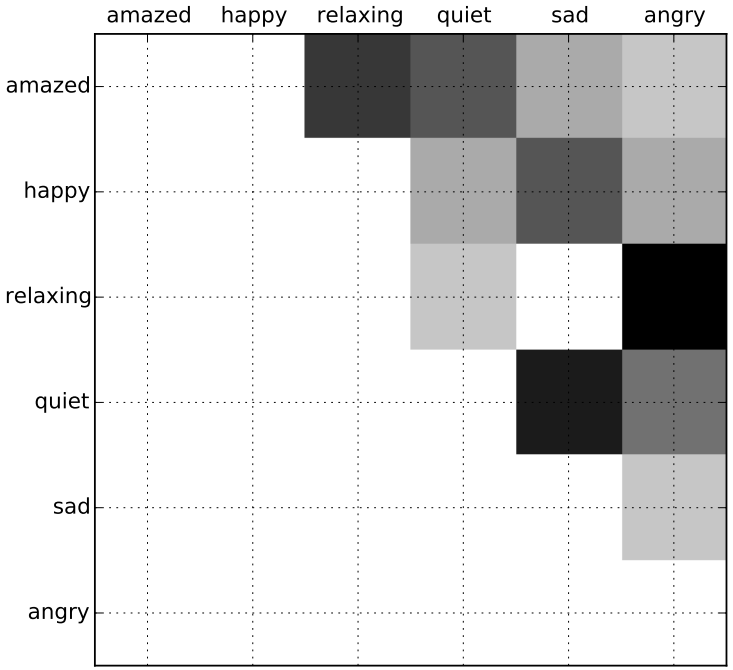
- multiple concepts
- multiple objects
- multiple people

e.g., Associating Scenes with concepts
$\subseteq \{\texttt{beach}, \texttt{sunset}, \texttt{foliage}, \texttt{field}, \texttt{mountain}, \texttt{urban}\}$

Labelling Audio

For example, labelling music with emotions, concepts, etc.

- amazed-surprised
- happy-pleased
- relaxing-calm
- quiet-still
- sad-lonely
- angry-aggressive



Medicine

Medical Diagnosis

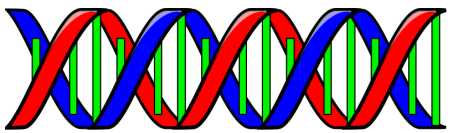
- medical history, symptoms $\rightarrow$ diseases / ailments

The 'Medical' dataset (a benchmark multi-label dataset),

- clinical free text reports by radiologists
- label assignment out of 45 ICD-9-CM codes

Bioinformatics

A gene or a protein may be associated with biological functions.



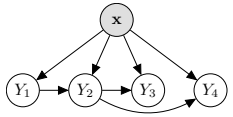
For example, the well-known Yeast dataset: $2,417$ genes, described by 103 attributes, labeled into 14 groups of the FunCat functional catalogue.
Useful applications such as anti-biotics[1]:

> *Activity prediction of AMPs from their amino acid sequences is of great therapeutic importance but imposes challenges on prediction methods due to label interactions. [...] we propose a novel multi-label learning model to address this problem.*

[1] e.g., Wang et al. *Multi-label Learning for Predicting the Activities of Antimicrobial Peptides*. Nature 2017

Multi-label Classification in Artificial Intelligence

- For knowledge representation (multi-label methods are often based on probabilistic graphical models)



- For decision making: multiple agents, agents taking multiple actions
- Techniques can be applied (almost) as-is to sequential data
- A stepping stone to structured output prediction,

$$h : \mathcal{X} \mapsto \mathcal{Y}$$

which encompasses tasks mapping (multiple) perception to (multiple) actions.

Localization and Tracking

We can consider a pixel/tile as a label, where $y_j = 1$ if the j-th tile contains an object, otherwise $y_j = 0$. The instance $\mathbf{x}$ contains some observation, e.g., sensor readings,

Figure: A room with a single light source and a number of light sensors.

Given some training data, we train a multi-label model for this problem

$$h : \mathcal{X} \to \mathcal{Y}$$

(some domain assumptions may help).

Localization and Tracking

We can consider a pixel/tile as a label, where $y_j = 1$ if the j-th tile contains an object, otherwise $y_j = 0$. The instance $\mathbf{x}$ contains some observation, e.g., sensor readings,

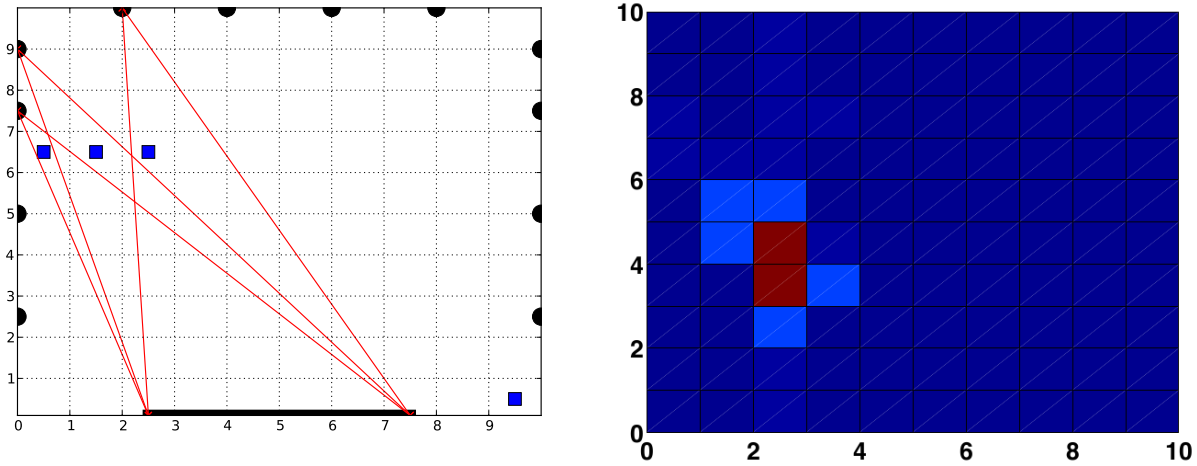


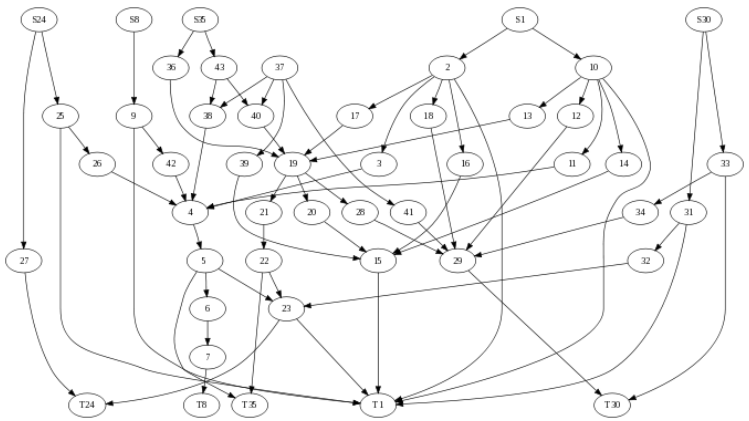
Figure: A room with a single light source and a number of light sensors.

Given some training data, we train a multi-label model for this problem

$$h : \mathcal{X} \to \mathcal{Y}$$

(some domain assumptions may help).

Internet of Things (IoT)

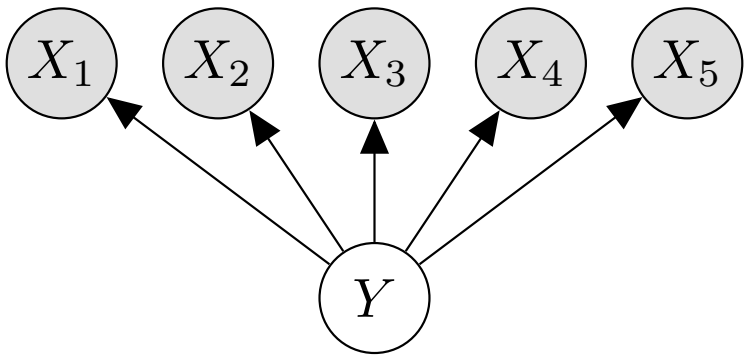


A network of interconnected *things* (software, actuators, sensors, vehicles, . . .). Many nodes require a multi-input multi-output decision-making framework.

Outline

1. Introduction
2. Applications
3. Probabilistic Models
4. Evaluation Metrics
5. Label Dependence and Structure
6. Inference in Probabilistic Models
7. (Multi-output) Neural Networks
8. (Multi-output) k-Nearest Neighbours
9. (Multi-output) Decision Trees
10. Big Multi-Labelled Data
11. Summary

(Single-label) Naive Bayes



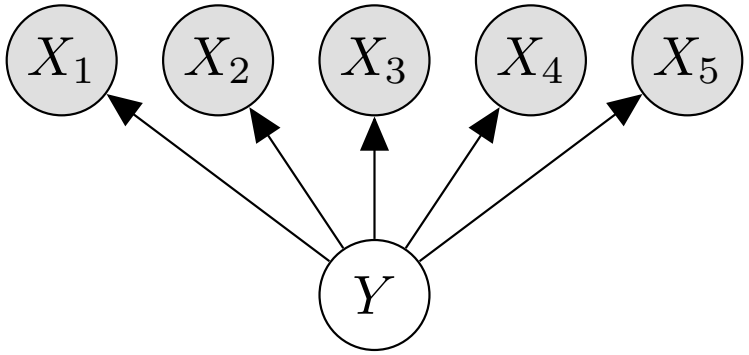
$$P(Y|\mathbf{X}) = \frac{P(Y)P(\mathbf{X}|Y)}{P(\mathbf{X})} \quad \triangleright \text{ Bayes rule}$$

$$P(\mathbf{X}, Y) = P(Y)P(\mathbf{X}|Y) \quad \triangleright \text{ factorize from PGM}$$

where likelihood (under independence assumption)

$$P(\mathbf{X}|Y) = \prod_{d=1}^{D} P(X_d|Y)$$

(Single-label) Naive Bayes

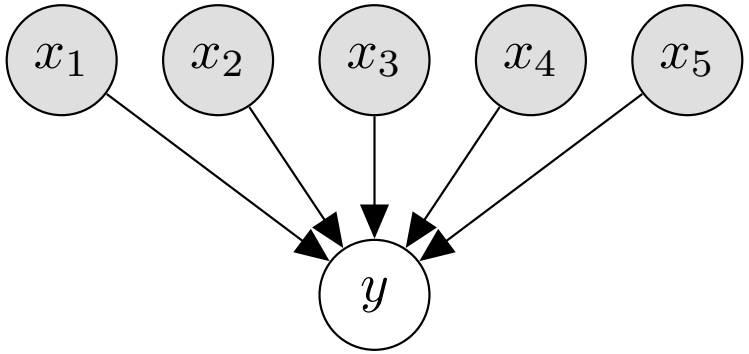


$$\mathsf{h}(\mathbf{x}) = \operatorname*{argmax}_{y \in \{0,1\}} P(y) P(\mathbf{x}|y) \quad \triangleright \text{ decision rule}$$

$$= \operatorname*{argmax}_{y \in \{0,1\}} P(y) \prod_{d=1}^{D} P(x_d|y)$$

i.e., our **classifier**, with with we may make classifications $\hat{y} = h(\mathbf{x})$.

Logistic Regression



$$p(y = 1|\mathbf{x}) \approx p(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^\top \mathbf{x})}$$

We take the gradient of the (log) error

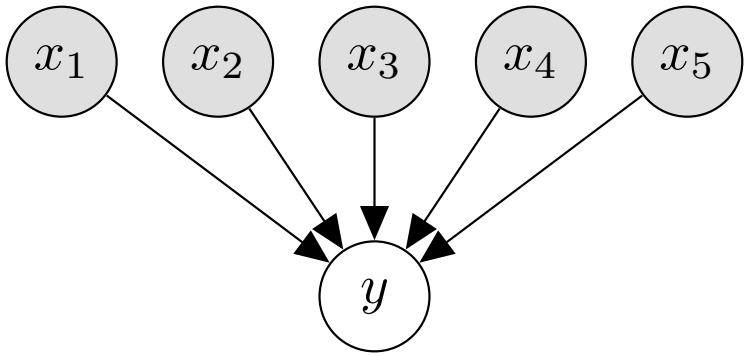
$$E(\boldsymbol{\theta}) = -\log \prod_{i=1}^{N} \left[(p_i)^{y_i} \cdot (1 - p_i)^{1-y_i} \right] = \sum_{i=1}^{N} E_i(\boldsymbol{\theta})$$

and we get a weight-update, to *descend* the error gradient

$$\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i + \lambda \nabla E_i(\boldsymbol{\theta})$$

i.e., (stochastic/*incremental*) gradient descent.

Logistic Regression



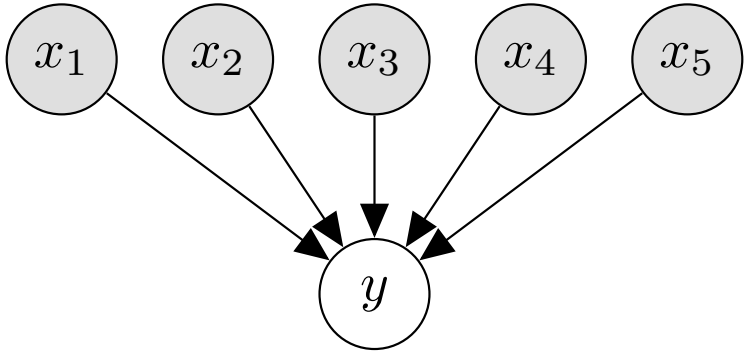
$$\hat{y} = h(\mathbf{x}) = \underset{y \in \{0,1\}}{\operatorname{argmax}}\, p(y|\mathbf{x})$$

$$p(y = 1|\mathbf{x}) = p(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^\top \mathbf{x})}$$

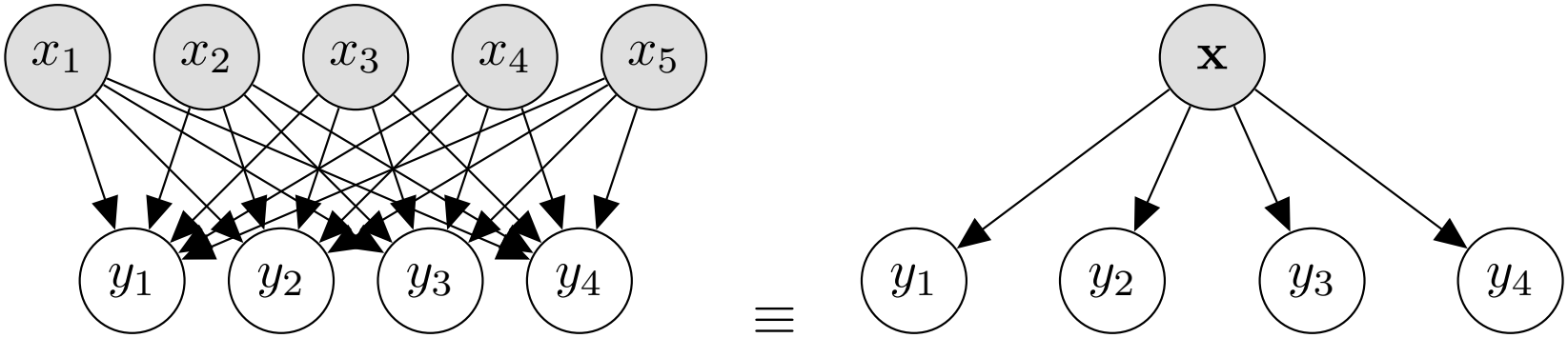
$$p(y = 0|\mathbf{x}) = 1 - p(\mathbf{x}; \boldsymbol{\theta})$$

A Note on Graphical Notation

We move from single label representation...

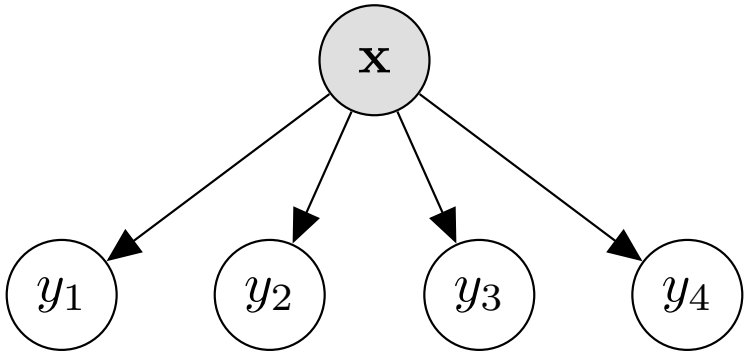


to multi-label representation...



(the shaded nodes represent the observed instance).

The Binary Relevance Approach



$$\hat{y}_j = h_j(\mathbf{x}) = \underset{y_j \in \{0,1\}}{\operatorname{argmax}} P(y_j|\mathbf{x}) \quad \triangleright \text{ for index, } j = 1, \ldots, L$$

and then,

$$\begin{aligned}
\hat{\mathbf{y}} &= \mathbf{h}(\mathbf{x}) = [\hat{y}_1, \ldots, \hat{y}_4] \\
&= \left[\underset{y_1 \in \{0,1\}}{\operatorname{argmax}} P(y_1|\mathbf{x}), \cdots, \underset{y_4 \in \{0,1\}}{\operatorname{argmax}} P(y_4|\mathbf{x}) \right] \\
&= \left[h_1(\mathbf{x}), \cdots, h_4(\mathbf{x}) \right]
\end{aligned}$$

Transformation

1. Transform dataset ...

$\mathbf{X}$	Y_1	Y_2	Y_3	Y_4
$\mathbf{x}^{(1)}$	0	1	1	0
$\mathbf{x}^{(2)}$	1	0	0	0
$\mathbf{x}^{(3)}$	0	1	0	0
$\mathbf{x}^{(4)}$	1	0	0	1
$\mathbf{x}^{(5)}$	0	0	0	1

...into L separate binary problems (one for each label)

$\mathbf{X}$	Y_1
$\mathbf{x}^{(1)}$	0
$\mathbf{x}^{(2)}$	1
$\mathbf{x}^{(3)}$	0
$\mathbf{x}^{(4)}$	1
$\mathbf{x}^{(5)}$	0

$\mathbf{X}$	Y_2
$\mathbf{x}^{(1)}$	1
$\mathbf{x}^{(2)}$	0
$\mathbf{x}^{(3)}$	1
$\mathbf{x}^{(4)}$	0
$\mathbf{x}^{(5)}$	0

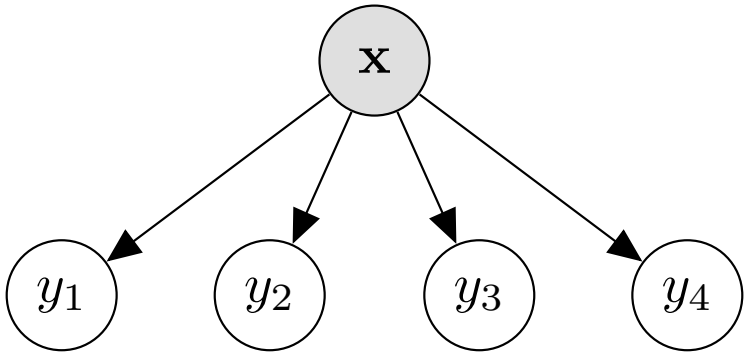
$\mathbf{X}$	Y_3
$\mathbf{x}^{(1)}$	1
$\mathbf{x}^{(2)}$	0
$\mathbf{x}^{(3)}$	0
$\mathbf{x}^{(4)}$	0
$\mathbf{x}^{(5)}$	0

$\mathbf{X}$	Y_4
$\mathbf{x}^{(1)}$	0
$\mathbf{x}^{(2)}$	0
$\mathbf{x}^{(3)}$	0
$\mathbf{x}^{(4)}$	1
$\mathbf{x}^{(5)}$	1

2. and train with any off-the-shelf binary base classifier:

$$h_j : \mathcal{X} \to \{0, 1\}$$

Why Not Binary Relevance?

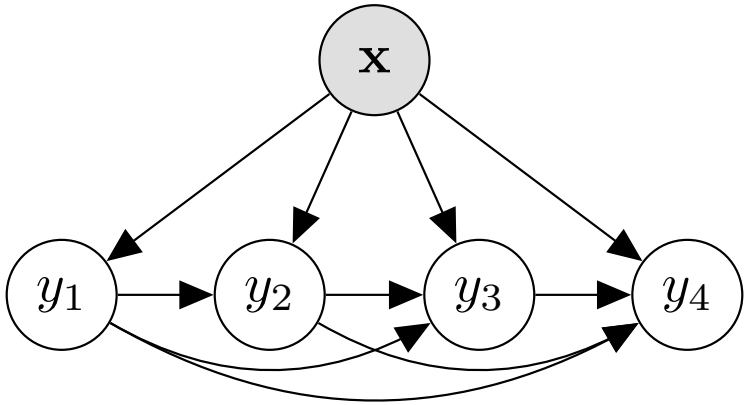


The independence assumption is not a good representation of the knowledge we have of the world;

$$P(\mathbf{y}|\mathbf{x}) \neq \prod_{j=1}^{L} P(y_j|\mathbf{x})$$

(in most cases).

A Probabilistic Cascade[2]



Given a query instance $\mathbf{x}$,

$$\hat{\mathbf{y}} = \underset{\mathbf{y}\in\{0,1\}^L}{\operatorname{argmax}} P(\mathbf{y}|\mathbf{x}) \quad = \underset{\mathbf{y}\in\{0,1\}^L}{\operatorname{argmax}} P(y_1|\mathbf{x}) \prod_{j=2}^{L} P(y_j|\mathbf{x}, y_1, \ldots, y_{j-1}) \qquad \triangleright \text{ chain}$$

We may write $P_{\mathbf{x}}(\mathbf{y}) = P_{\mathbf{x}}(y_1) \prod_{j=2}^{L} P_{\mathbf{x}}(y_j|y_1, \ldots, y_{j-1})$

[2] In the literature, known as *Probabilistic Classifier Chains*

Multi-label Learning

From the formulation

$$\hat{\mathbf{y}} = \underset{\mathbf{y}\in\{0,1\}^L}{\operatorname{argmax}}\, P(\mathbf{y}|\mathbf{x}) = \underset{\mathbf{y}\in\{0,1\}^L}{\operatorname{argmax}}\, P(y_1|\mathbf{x}) \prod_{j=2}^{L} P(y_j|\mathbf{x}, y_1, \ldots, y_{j-1})$$

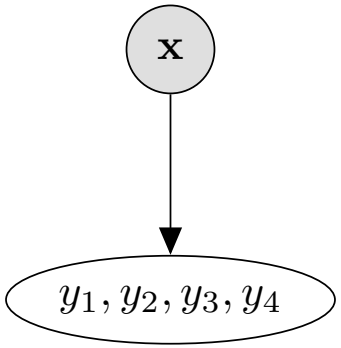
we get our classifier h, such that $\hat{\mathbf{y}} = h(\mathbf{x})$. But where does P come from? As in single-label learning, P must be *estimated*/learned from training data. We may either

- learn $P(\mathbf{y}|\mathbf{x})$ directly (e.g., label-powerset approach), or
- learn the $P(y_j|\mathbf{x}, y_1, \ldots, y_{j-1})$ individually (binary relevance approach)

How to do this learning? It depends what loss function we are interested in.

Each Label Combination as a Class Value

We may formulate one multi-class problem (taking many values),



$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}}{\operatorname{argmax}} P(\mathbf{y}|\mathbf{x})$$

where the space $\mathcal{Y}$ contains all possible combinations:

$$\mathcal{Y} = \{[0,0,0,0],[0,0,0,1],\ldots,[1,1,1,1]\}$$

i.e., the powerset[3] of $\mathcal{L} = \{1,\ldots,L\}$; of size $|\mathcal{Y}| = \mathcal{P}(\mathcal{L}) = 2^{|\mathcal{L}|}$. Although, typically, $\mathcal{Y} = \{\mathbf{y}^{(i)},\ldots,\mathbf{y}^{(N)}\}$, i.e., the distinct combinations in the training set; thus $|\mathcal{Y}| \leq \min(N, 2^L)$.

[3] Known in the literature as the label powerset method (LP)

Transformation

1. Transform dataset . . .

$\mathbf{X}$	Y_1	Y_2	Y_3	Y_4
$\mathbf{x}^{(1)}$	0	1	1	0
$\mathbf{x}^{(2)}$	1	0	0	0
$\mathbf{x}^{(3)}$	0	1	1	0
$\mathbf{x}^{(4)}$	1	0	0	1
$\mathbf{x}^{(5)}$	0	0	0	1

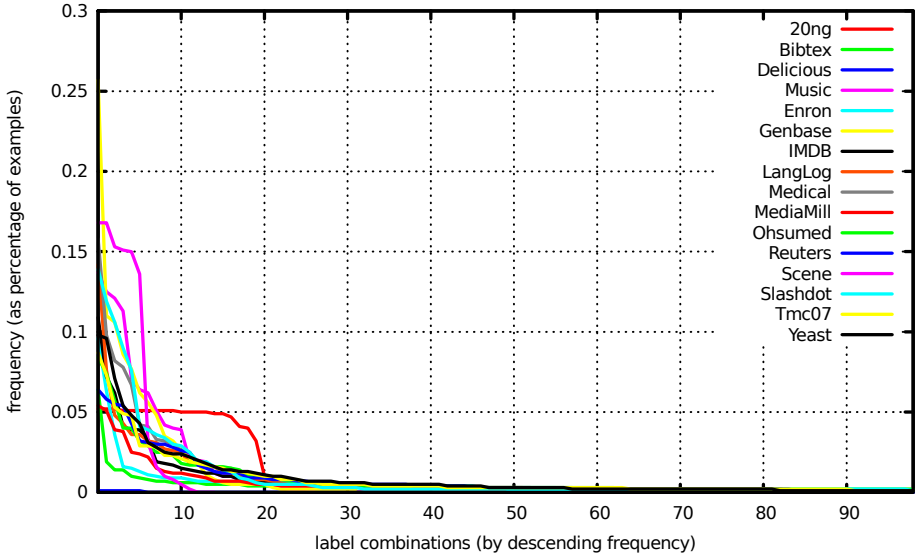
. . . into a multi-*class* dataset:

$\mathbf{X}$	$Y \in 2^L$
$\mathbf{x}^{(1)}$	0110
$\mathbf{x}^{(2)}$	1000
$\mathbf{x}^{(3)}$	0110
$\mathbf{x}^{(4)}$	1001
$\mathbf{x}^{(5)}$	0001

2. . . . and train any off-the-shelf multi-*class* classifier.

Issues with the Label-powerset Method

- Complexity: (up to 2^L combinations)
- Imbalance: few examples per class label
- Overfitting: how to predict new value?



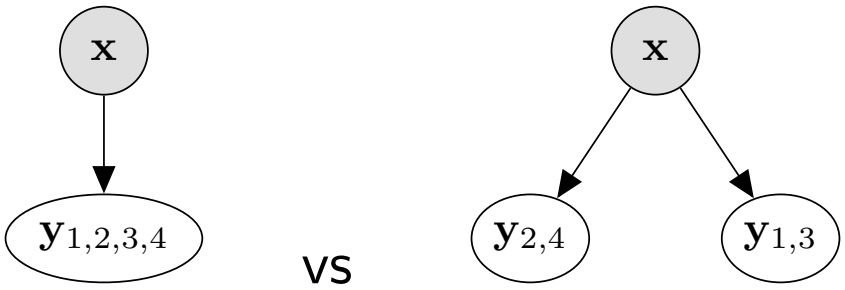
Note: horizontal axis has been truncated

Solutions involve:

- Pruning
- Dealing with subsets

Meta Labels

Decompose the labelset into M sets of k labels.

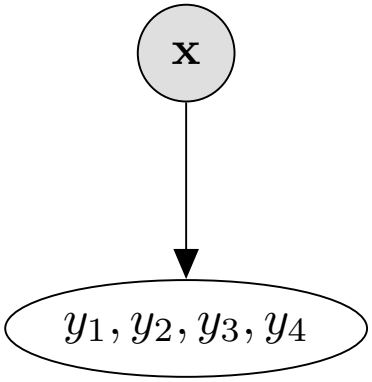


	$\hat{y}_1$	$\hat{y}_2$	$\hat{y}_3$	$\hat{y}_4$
$h^1(\mathbf{x})$		0		0
$h^2(\mathbf{x})$	1		1	
$\hat{\mathbf{y}}$	1	0	1	0

- Apply LP to the label subsets
- Combine later via voting
- Complexity reduction from $O(2^L)$ to $O(M \cdot 2^k)$, predictive performance usually improves too.

Same Goal, Different Approaches

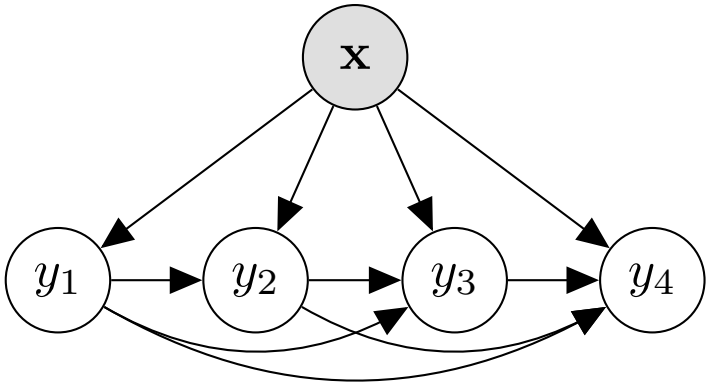
Meta Labels



$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}}{\operatorname{argmax}}\, P(\mathbf{y}|\mathbf{x})$$

- goal: reduce size of $\mathcal{Y}$ (i.e., the number of combinations)

Classifier Chains



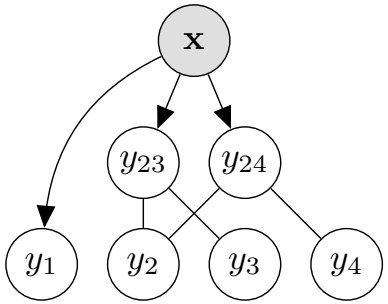
$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}}{\operatorname{argmax}}\, P(\mathbf{y}|\mathbf{x})$$

$$= \underset{\mathbf{y} \in \{0,1\}^L}{\operatorname{argmax}} \underbrace{P(y_1|\mathbf{x}) \prod_{j=2}^{L} P(y_j|\mathbf{x}, y_1, \ldots, y_{j-1})}_{\text{chain rule}}$$

- goal: reduce connectivity among $Y_1, \ldots, Y_L$

Same Goal, Different Approaches

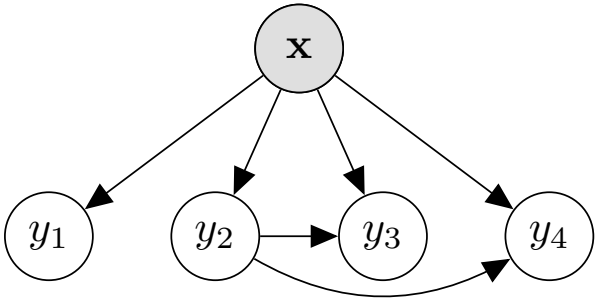
Meta Labels



$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}}{\operatorname{argmax}}\, P(\mathbf{y}|\mathbf{x})$$

- goal: reduce size of $\mathcal{Y}$ (i.e., the number of combinations)

Classifier Chains



$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}}{\operatorname{argmax}}\, P(\mathbf{y}|\mathbf{x})$$

$$= \underset{\mathbf{y} \in \{0,1\}^L}{\operatorname{argmax}} \underbrace{P(y_1|\mathbf{x}) \prod_{j=2}^{L} P(y_j|\mathbf{x}, y_1, \ldots, y_{j-1})}_{\text{chain rule}}$$

- goal: reduce connectivity among $Y_1, \ldots, Y_L$

Both methods: Model label dependence with a minimum of structure, i.e., predictive performance vs computational complexity.

Outline

1. Introduction
2. Applications
3. Probabilistic Models
4. Evaluation Metrics
5. Label Dependence and Structure
6. Inference in Probabilistic Models
7. (Multi-output) Neural Networks
8. (Multi-output) k-Nearest Neighbours
9. (Multi-output) Decision Trees
10. Big Multi-Labelled Data
11. Summary

Multi-label Evaluation

For an instance $\mathbf{x}$, our model provides predictions

$$[\hat{y}_1, \ldots, \hat{y}_L] = \hat{\mathbf{y}} = h(\mathbf{x})$$

$\mathbf{x}^{(i)}$	$\mathbf{y}^{(i)}$	$\hat{\mathbf{y}}^{(i)}$
$\mathbf{x}^{(1)}$	[1 0 1 0]	[1 0 0 1]
$\mathbf{x}^{(2)}$	[0 1 0 1]	[0 1 0 1]
$\mathbf{x}^{(3)}$	[1 0 0 1]	[1 0 0 1]
$\mathbf{x}^{(4)}$	[0 1 1 0]	[0 1 0 0]
$\mathbf{x}^{(5)}$	[1 0 0 0]	[1 0 0 1]

How good is model h?

Multi-label Evaluation Metrics

	$\mathbf{y}^{(i)}$	$\hat{\mathbf{y}}^{(i)}$
$\mathbf{x}^{(1)}$	[1 0 1 0]	[1 0 0 1]
$\mathbf{x}^{(2)}$	[0 1 0 1]	[0 1 0 1]
$\mathbf{x}^{(3)}$	[1 0 0 1]	[1 0 0 1]
$\mathbf{x}^{(4)}$	[0 1 1 0]	[0 1 0 0]
$\mathbf{x}^{(5)}$	[1 0 0 0]	[1 0 0 1]

Hamming loss

$$L_{\text{Hamming}}(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{N \cdot L} \sum_{i=1}^{N} \sum_{i=1}^{L} \mathbb{I}[\hat{y}_j^{(i)} \neq y_j^{(i)}]$$
$$= 0.20$$

No consideration of label dependence; easy to get low loss on sparse labels.

Multi-label Evaluation Metrics

	$\mathbf{y}^{(i)}$	$\hat{\mathbf{y}}^{(i)}$
$\mathbf{x}^{(1)}$	[1 0 1 0]	[1 0 0 1]
$\mathbf{x}^{(2)}$	[0 1 0 1]	[0 1 0 1]
$\mathbf{x}^{(3)}$	[1 0 0 1]	[1 0 0 1]
$\mathbf{x}^{(4)}$	[0 1 1 0]	[0 1 0 0]
$\mathbf{x}^{(5)}$	[1 0 0 0]	[1 0 0 1]

0/1 LOSS

$$
\begin{aligned}
L_{0/1}(\mathbf{Y}, \hat{\mathbf{Y}}) &= \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(\hat{\mathbf{y}}^{(i)} \neq \mathbf{y}^{(i)}) \\
&= \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\Big(\underbrace{\sum_{i=1}^{L} \mathbb{I}[\hat{y}_j^{(i)} \neq y_j^{(i)}]}_{\text{Hamming Loss}} = 0 \Big) \\
&= 0.60
\end{aligned}
$$

If a single label relevance incorrect, loss of 1 for that instance.

Multi-label Evaluation Metrics

	$\mathbf{y}^{(i)}$	$\hat{\mathbf{y}}^{(i)}$
$\mathbf{x}^{(1)}$	[1 0 1 0]	[1 0 0 1]
$\mathbf{x}^{(2)}$	[0 1 0 1]	[0 1 0 1]
$\mathbf{x}^{(3)}$	[1 0 0 1]	[1 0 0 1]
$\mathbf{x}^{(4)}$	[0 1 1 0]	[0 1 0 0]
$\mathbf{x}^{(5)}$	[1 0 0 0]	[1 0 0 1]

Jaccard Index

$$
\begin{aligned}
J(\mathbf{Y}, \hat{\mathbf{Y}}) &= \frac{1}{N} \sum_{i=1}^{N} \frac{|\hat{\mathbf{y}}^{(i)} \wedge \mathbf{y}^{(i)}|}{|\hat{\mathbf{y}}^{(i)} \vee \mathbf{y}^{(i)}|} \\
&= \frac{1}{5}(\frac{1}{3} + 1 + 1 + \frac{1}{2} + \frac{1}{2}) \\
&= 0.67
\end{aligned}
$$

(Where $\vee$ and $\wedge$ are the logical OR and AND operations, applied vector-wise)

Multi-label Evaluation Metrics

We can evaluate posterior probabilities/confidences directly.

	$\mathbf{y}^{(i)}$	$[P(y_j \mid \mathbf{x}^{(i)}])]_{j=1}^{L}$
$\mathbf{x}^{(1)}$	[1 0 1 0]	[0.9 0.0 0.4 0.6]
$\mathbf{x}^{(2)}$	[0 1 0 1]	[0.1 0.8 0.0 0.8]
$\mathbf{x}^{(3)}$	[1 0 0 1]	[0.8 0.0 0.1 0.7]
$\mathbf{x}^{(4)}$	[0 1 1 0]	[0.1 0.7 0.1 0.2]
$\mathbf{x}^{(5)}$	[1 0 0 0]	[1.0 0.0 0.0 1.0]

Log loss

- $y_3 = 1$, $h_3(\mathbf{x}) = 0.4$ incurs loss of $-\log(0.4) = 0.92$
- $y_3 = 1$, $h_3(\mathbf{x}) = 0.1$ incurs loss of $-\log(0.1) = 2.30$

aka cross entropy for multiple classes, $y \in \{1, \ldots, K\}$.

Multi-label Evaluation Metrics

	$\mathbf{y}^{(i)}$	$[P(y_j \mid \mathbf{x}^{(i)}])]_{j=1}^{L}$
$\mathbf{x}^{(1)}$	[1 0 1 0]	[0.9 0.0 0.4 0.6]
$\mathbf{x}^{(2)}$	[0 1 0 1]	[0.1 0.8 0.0 0.8]
$\mathbf{x}^{(3)}$	[1 0 0 1]	[0.8 0.0 0.1 0.7]
$\mathbf{x}^{(4)}$	[0 1 1 0]	[0.1 0.7 0.1 0.2]
$\mathbf{x}^{(5)}$	[1 0 0 0]	[1.0 0.0 0.0 1.0]

Ranking loss – to encourage good ranking;
evaluates the average fraction of label pairs miss-ordered for $\mathbf{x}$:

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{(j,k):y_j > y_k} \left(\mathbb{I}[r_i(j) < r_i(k)] + \frac{1}{2} \mathbb{I}[r_i(j) = r_i(k)] \right)$$

where $r_i(j) :=$ ranking of label j for instance $\mathbf{x}^{(i)}$

Multi-label Evaluation Metrics

	$\mathbf{y}^{(i)}$	$[P(y_j \mid \mathbf{x}^{(i)}])]_{j=1}^{L}$
$\mathbf{x}^{(1)}$	[1 0 1 0]	[0.9 0.0 0.4 0.6]
$\mathbf{x}^{(2)}$	[0 1 0 1]	[0.1 0.8 0.0 0.8]
$\mathbf{x}^{(3)}$	[1 0 0 1]	[0.8 0.0 0.1 0.7]
$\mathbf{x}^{(4)}$	[0 1 1 0]	[0.1 0.7 0.1 0.2]
$\mathbf{x}^{(5)}$	[1 0 0 0]	[1.0 0.0 0.0 1.0]

Ranking loss – to encourage good ranking;
evaluates the average fraction of label pairs miss-ordered for $\mathbf{x}$:

$$\frac{1}{5}(\frac{1}{4} + \frac{0}{4} + \frac{0}{4} + \frac{1.5}{4} + \frac{1}{4})$$

Multi-label Evaluation Metrics

Other metrics include

- ONE ERROR – if top ranked label is not in set of true labels
- COVERAGE – average "depth" to cover all true labels
- PRECISION
- RECALL
- macro-averaged F-MEASURE (ordinary averaging of a binary measure)
- micro-averaged F-MEASURE (labels as different instances of a 'global' label)
- PRECISION vs. RECALL curves

As generally, a metric should be considered based on the application. We cannot optimize all losses at once!

Minimizer for 0/1 Loss

$$L_{0/1} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(\hat{\mathbf{y}}^{(i)} \neq \mathbf{y}^{(i)})$$

We write out the expected loss, and try to minimize it. The single-label case:

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}}[\ell(\hat{y})] &= \sum_{k \in \{0,1\}} \ell_{\mathbf{x}}(y = k, \hat{y}) \cdot P(y = k|\mathbf{x}) \\
&= \sum_{k \in \{0,1\}} (1 - \delta_{y\hat{y}}) \cdot P(y = k|\mathbf{x}) \quad \triangleright \text{ plug in } \ell(\cdot, \cdot) \\
&= 1 - P(y|\mathbf{x}) \\
\operatorname*{argmin}_{\hat{y} \in \{0,1\}} \mathbb{E}_{\mathbf{x}}[\ell(\hat{y})] &= \operatorname*{argmin}_{\hat{y} \in \{0,1\}} \Big[1 - P(y|\mathbf{x})\Big] \quad \triangleright \text{ want to minimize this} \\
\hat{y} &= \operatorname*{argmax}_{y \in \{0,1\}} P(y|\mathbf{x}) \quad \triangleright \text{ . . . , i.e., MAP estimate!}
\end{aligned}$$

Minimizer for 0/1 Loss

$$L_{0/1} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(\hat{\mathbf{y}}^{(i)} \neq \mathbf{y}^{(i)})$$

We write out the expected loss, and try to minimize it. The single-label case:

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}}[\ell(\hat{y})] &= \sum_{k \in \{0,1\}} \ell_{\mathbf{x}}(y = k, \hat{y}) \cdot P(y = k|\mathbf{x}) \\
&= \sum_{k \in \{0,1\}} (1 - \delta_{y\hat{y}}) \cdot P(y = k|\mathbf{x}) \quad \triangleright \text{ plug in } \ell(\cdot, \cdot) \\
&= 1 - P(y|\mathbf{x}) \\
\operatorname*{argmin}_{\hat{y} \in \{0,1\}} \mathbb{E}_{\mathbf{x}}[\ell(\hat{y})] &= \operatorname*{argmin}_{\hat{y} \in \{0,1\}} \Big[1 - P(y|\mathbf{x})\Big] \quad \triangleright \text{ want to minimize this} \\
\hat{y} &= \operatorname*{argmax}_{y \in \{0,1\}} P(y|\mathbf{x}) \quad \triangleright \text{ . . . , i.e., MAP estimate!}
\end{aligned}$$

Same in the multi-label case, $k \in \{0,1\}^L$ and thus $y \in \{0,1\}^L$.

Hamming Loss Decomposes Across Labels

$$
\begin{aligned}
L_{\text{Hamming}}(\mathbf{Y}, \hat{\mathbf{Y}}) &= \frac{1}{N \cdot L} \sum_{i=1}^{N} \sum_{i=1}^{L} \mathbb{I}[\hat{y}_j^{(i)} \neq y_j^{(i)}] \\
&= \frac{1}{N} \sum_{i=1}^{N} \left(\mathbb{I}[\hat{y}_1^{(i)} \neq y_1^{(i)}] + \cdots + \mathbb{I}[\hat{y}_L^{(i)} \neq y_L^{(i)}] \right) \\
&= \ell_{0/1}(\mathbf{y}_1, \hat{\mathbf{y}}_1) + \cdots + \ell_{0/1}(\mathbf{y}_L, \hat{\mathbf{y}}_L)
\end{aligned}
$$

and for each j-th label,

$$
\begin{aligned}
\hat{y}_j &= \operatorname*{argmax}_{y_j \in \{0,1\}} \sum_{y_k \in \{0,1\}} P(y_j|, y_k, \mathbf{x}) \\
&= \operatorname*{argmax}_{y \in \{0,1\}} P_j(y|\mathbf{x})
\end{aligned}
$$

Hamming Loss Decomposes Across Labels

$$
\begin{aligned}
L_{\text{Hamming}}(\mathbf{Y}, \hat{\mathbf{Y}}) &= \frac{1}{N \cdot L} \sum_{i=1}^{N} \sum_{i=1}^{L} \mathbb{I}[\hat{y}_j^{(i)} \neq y_j^{(i)}] \\
&= \frac{1}{N} \sum_{i=1}^{N} \left(\mathbb{I}[\hat{y}_1^{(i)} \neq y_1^{(i)}] + \cdots + \mathbb{I}[\hat{y}_L^{(i)} \neq y_L^{(i)}] \right) \\
&= \ell_{0/1}(\mathbf{y}_1, \hat{\mathbf{y}}_1) + \cdots + \ell_{0/1}(\mathbf{y}_L, \hat{\mathbf{y}}_L)
\end{aligned}
$$

and for each j-th label,

$$
\begin{aligned}
\hat{y}_j &= \operatorname*{argmax}_{y_j \in \{0,1\}} \sum_{y_k \in \{0,1\}} P(y_j|, y_k, \mathbf{x}) \\
&= \operatorname*{argmax}_{y \in \{0,1\}} P_j(y|\mathbf{x})
\end{aligned}
$$

In theory, Hamming loss is optimized by naive binary relevance.

Loss Functions in Multi-label Classification

Hamming loss is not very interesting for structured data. We would not optimize an image recognition algorithm for each pixel individually! But for optimizing 0/1 loss we have a daunting search space:

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \{0,1\}^L}{\operatorname{argmax}} P(y_1|\mathbf{x}) \prod_{j=2}^{L} P(y_j|\mathbf{x}, y_1, \ldots, y_{j-1})$$

Whatever our search method, the space gets smaller if we can identify independence.

Dependence	Space	Comment
Full	2^L	Should use full cascade or similar
Split	$2^{L/2} + 2^{L/2}$	use something in between
None	$2L$	Maximizing 0/1 and Hamming loss equivalent!

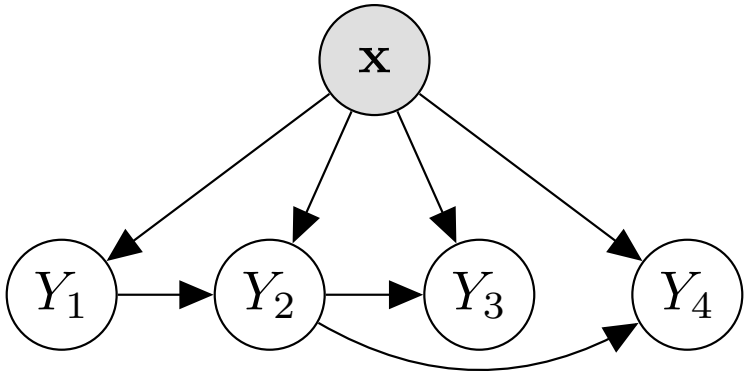
i.e., try to identify label dependence; use it to build a structure.

Outline

1. Introduction
2. Applications
3. Probabilistic Models
4. Evaluation Metrics
5. Label Dependence and Structure
6. Inference in Probabilistic Models
7. (Multi-output) Neural Networks
8. (Multi-output) k-Nearest Neighbours
9. (Multi-output) Decision Trees
10. Big Multi-Labelled Data
11. Summary

Structure for a Multi-label Classifier

We can formulate any structure,

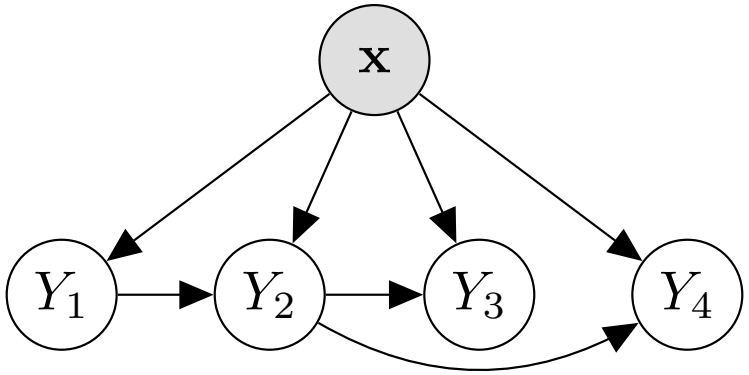


$$\hat{y}_j = \underset{y_j \in \{0,1\}}{\operatorname{argmax}} P(y_j | \mathbf{x}, \mathrm{pa}_j)$$

where pa_j = parents of node j. This defines a structure, i.e., graph $\mathcal{G}$.

Structure for a Multi-label Classifier

We can formulate any structure,

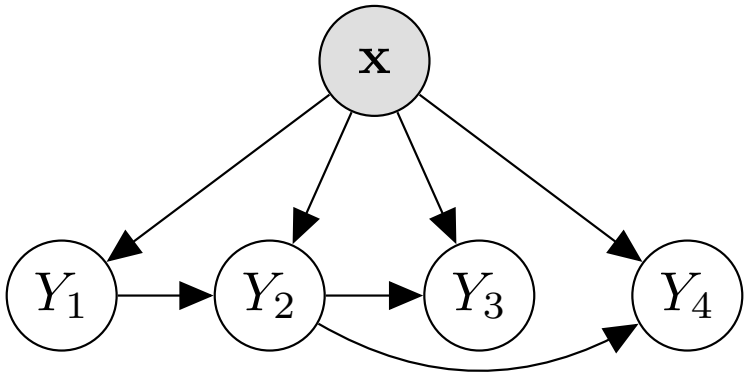


$$\hat{y}_j = \underset{y_j \in \{0,1\}}{\operatorname{argmax}} P(y_j | \mathbf{x}, \mathrm{pa}_j)$$

where pa_j = parents of node j. This defines a structure, i.e., graph $\mathcal{G}$. What structure should we use? – The graph $\mathcal{G}^*$ that provides the best score.

Learning Structure

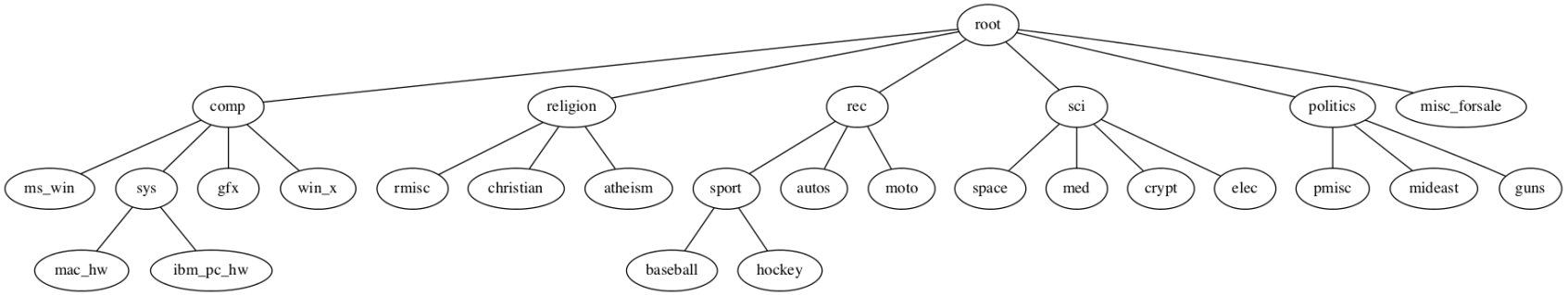
- From a Bayesian network point of view (a probabilistic graphical model), structure represents conditional dependence between variables
- From a neural network point of view (may be considered a special case of a Bayesian network), edges represent functional dependence



i.e., we need to explore the idea of dependence.

Hierarchy and dependence

But wait, (perhaps) I have a structure!



Is a dataset hierarchy a representation of label dependence?

Hierarchy and dependence

But wait, (perhaps) I have a structure!

1 Coarse genre
1.1 Company Business, Strategy, etc. 1.2 Purely Personal 1.3 Personal but in professional context (e.g., it was good working with you) 1.4 Logistic Arrangements (meeting scheduling, technical support, etc) 1.5 Employment arrangements (job seeking, hiring, recommendations, etc) 1.6 Document editing/checking (collaboration) 1.7 Empty message (due to missing attachment) 1.8 Empty message
1.1 Company Business, Strategy, etc.
1.1.1 regulations and regulators (includes price caps) 1.1.2 internal projects – progress and strategy 1.1.3 company image – current 1.1.4 company image – changing / influencing 1.1.5 political influence / contributions / contacts 1.1.6 california energy crisis / california politics 1.1.7 internal company policy 1.1.8 internal company operations 1.1.9 alliances / partnerships 1.1.10 legal advice 1.1.11 talking points 1.1.12 meeting minutes 1.1.13 trip reports
2 Included/forwarded information
2.1 Includes new text in addition to forwarded material 2.2 Forwarded email(s) including replies 2.3 Business letter(s) / document(s) 2.4 News article(s) 2.5 Government / academic report(s) 2.6 Government action(s) (such as results of a hearing, etc) 2.7 Press release(s) 2.8 Legal documents (complaints, lawsuits, advice) 2.9 Pointers to url(s) 2.10 Newsletters 2.11 Jokes, humor (related to business) 2.12 Jokes, humor (unrelated to business) 2.13 Attachment(s) (assumed missing)
3 Emotional tone (if not neutral)
3.1 jubilation 3.2 hope / anticipation 3.3 humor 3.3 camaraderie 3.5 admiration 3.6 gratitude 3.7 friendship / affection 3.8 sympathy / support 3.9 sarcasm 3.10 secrecy / confidentiality 3.11 worry / anxiety 3.12 concern 3.13 competitiveness / aggressiveness 3.14 triumph / gloating 3.15 pride 3.16 anger / agitation 3.17 sadness / despair 3.18 shame 3.19 dislike / scorn

Is a dataset hierarchy a representation of label dependence?

Hierarchy and dependence

But wait, (perhaps) I have a structure!

Is a dataset hierarchy a representation of label dependence?

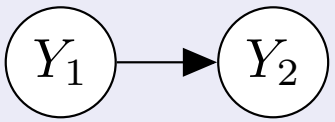
- Yes, but not necessarily the best one!
- Hierarchies normally only represent similarities, but exclusivity is also dependence.
- A random hierarchy/structure may perform as well

Marginal Dependence

Marginal dependence

When the joint is **not** the product of the marginals, i.e.,

$$P(Y_2) \neq P(Y_2|Y_1)$$
$$P(Y_1)P(Y_2) \neq P(Y_1, Y_2)$$



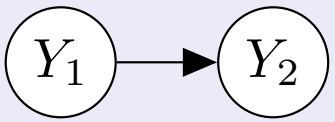
We can estimate from pairwise co-occurrence frequencies in training data, correlation matrix, etc. (the empirical probability P).

Marginal Dependence

Marginal dependence

When the joint is **not** the product of the marginals, i.e.,

$$P(Y_2) \neq P(Y_2|Y_1)$$
$$P(Y_1)P(Y_2) \neq P(Y_1, Y_2)$$



We can estimate from pairwise co-occurrence frequencies in training data, correlation matrix, etc. (the empirical probability P).

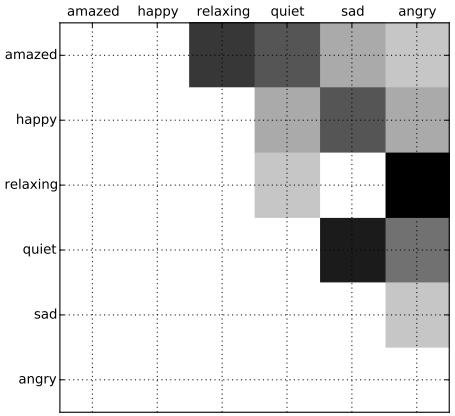


Figure: Music dataset - Mutual Information:
$I(Y_1, Y_2) = \sum_{y_1,y_2} P(y_1, y_2) \log \frac{P(y_1,y_2)}{P(y_1)P(y_2)}$

Toy Problem – Marginal Dependence

X_1	X_2	Y_1	Y_2	$P(\mathbf{x}, \mathbf{y})$
0	0	0	0	0.25
0	1	0	1	0.25
1	0	0	1	0.25
1	1	1	0	0.25
...	...	...	...	0

Is there marginal label dependence between labels Y_1 and Y_2?

Toy Problem – Marginal Dependence

X_1	X_2	Y_1	Y_2	$P(\mathbf{x}, \mathbf{y})$
0	0	0	0	0.25
0	1	0	1	0.25
1	0	0	1	0.25
1	1	1	0	0.25
...	...	...	...	0

Is there marginal label dependence between labels Y_1 and Y_2?

$$P(Y_1, Y_2) = P(Y_1) \cdot P(Y_2) \quad \triangleright \text{ if independent}$$

$$P(Y_1, Y_2 = 1, 1) \neq P(Y_1 = 1) \cdot P(Y_2 = 1)$$

$$0.0 \neq 0.25 \cdot 0.5 = 0.125$$

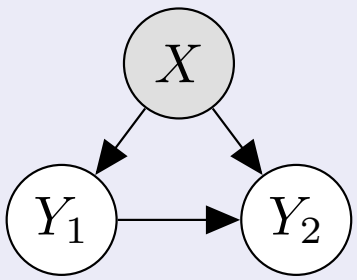
and so on (for $\mathbf{y} \in \{[0,0], [0,1], [1,0], [1,1]\}$). So, yes – there is *marginal* dependence! But what about the input $\mathbf{x}$? We are really interested in $P(\mathbf{y}|\mathbf{x})$ rather than $P(\mathbf{y})$.

Conditional Label Dependence

Conditional dependence

. . . conditioned on input observation **x**.

$$P(y_2|y_1, \mathbf{x}) \neq P(y_2|\mathbf{x})$$

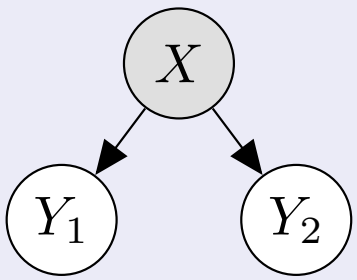


Conditional Label Dependence

Conditional *in*dependence

. . . conditioned on input observation $\mathbf{x}$.

$$P(y_1|\mathbf{x}) = P(y_1|, y_2, \mathbf{x})$$
$$P(y_2|\mathbf{x}) = P(y_2|, y_1, \mathbf{x})$$

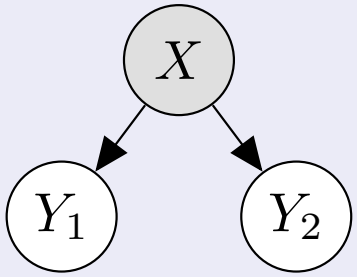


Conditional Label Dependence

Conditional dependence

. . . conditioned on input observation $\mathbf{x}$.

$$P(y_1|\mathbf{x}) = P(y_1|, y_2, \mathbf{x})$$
$$P(y_2|\mathbf{x}) = P(y_2|, y_1, \mathbf{x})$$



But how to measure $P(y_1|\mathbf{x})$, $P(y_2|\mathbf{x}, y_1)$, etc? Recall,

$$\hat{y}_2 = h_2(\mathbf{x}) = \underset{y_2 \in \{0,1\}}{\operatorname{argmax}} P(y_2|\mathbf{x}, y_1)$$

i.e., our models. We need to build and compare models. If the independent model outperforms the joint model, we have conditional dependence!

We can compare two models using a loss function, i.e.,

$$\epsilon_j = \ell\Big(y_j, h_j(\mathbf{x})\Big)$$
$$\epsilon'_j = \ell\Big(y_j, h'(\mathbf{x})_j\Big)$$

where h' models labels together. If $\epsilon'_j < \epsilon_j$, then the labels are conditionally dependent.

We can compare two models using a loss function, i.e.,

$$\epsilon_j = \ell\Big(y_j, h_j(\mathbf{x})\Big)$$
$$\epsilon'_j = \ell\Big(y_j, h'(\mathbf{x})_j\Big)$$

where h' models labels together. If $\epsilon'_j < \epsilon_j$, then the labels are conditionally dependent. Notice that

- h_j is the base model; and therefore,
- conditional dependence is dependent on the base model!

Given two labels, and two binary classifiers (one for each), we can also compare

$$\epsilon_j = \ell\Big(y_j, h_j(\mathbf{x})\Big)$$
$$\epsilon_k = \ell\Big(y_k, h_k(\mathbf{x})\Big)$$

The function ℓ takes into account $\mathbf{x}$. Therefore, we can measure dependence among the errors as a measure of conditional dependence.

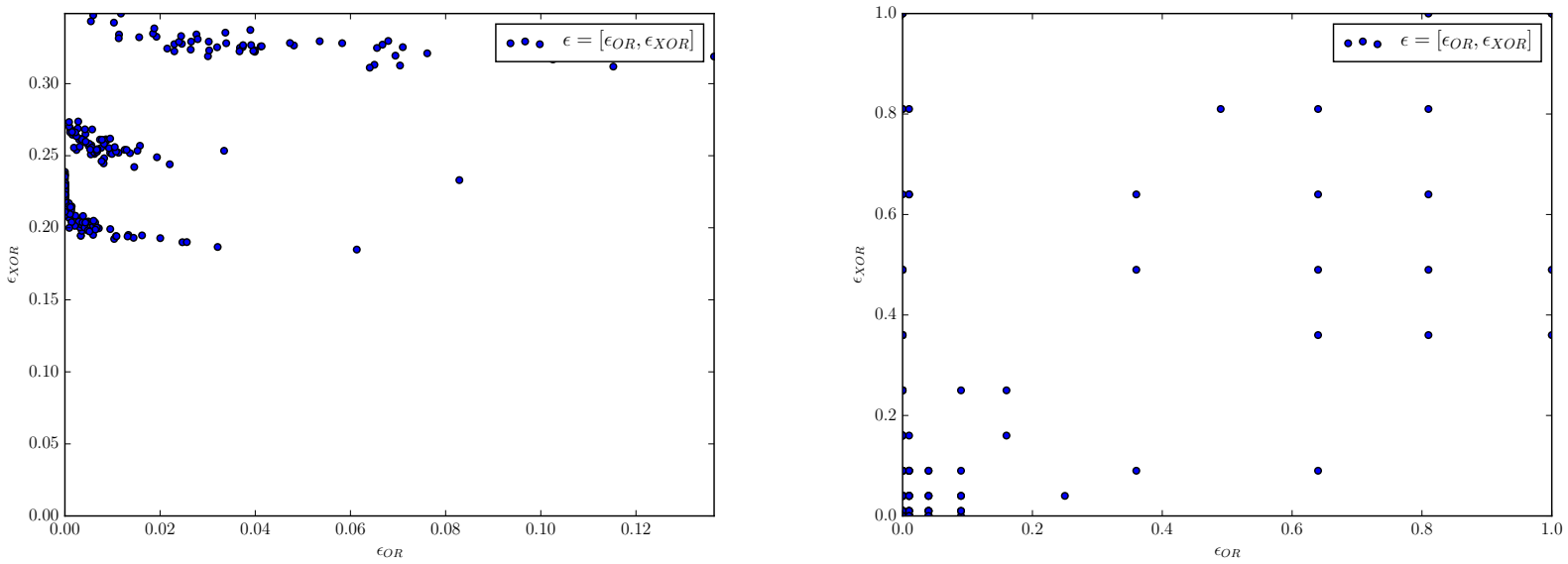


Figure: Errors from logistic regression (left) and decision tree (right).

Conditional dependence in Multi-Output Learning

Suppose two problems:

$$y_1 = f_1(\mathbf{x}) + \epsilon_1, \text{ and } y_2 = f_2(\mathbf{x}) + \epsilon_2$$

If

1. ϵ_1 and ϵ_2 are independent from $\mathbf{x}$, *and*
2. ϵ_1 and ϵ_2 are also independent from each other,

then y_1 and y_2 are conditionally independent given $\mathbf{x}$.

Conditional dependence in Multi-Output Learning

Suppose two problems:

$$y_1 = f_1(\mathbf{x}) + \epsilon_1, \text{ and } y_2 = f_2(\mathbf{x}) + \epsilon_2$$

If

1. ϵ_1 and ϵ_2 are independent from $\mathbf{x}$, *and*
2. ϵ_1 and ϵ_2 are also independent from each other,

then y_1 and y_2 are conditionally independent given $\mathbf{x}$.

We may (in *some cases*) assume 1. on account of MLE (although this depends on the model, data, basis functions, etc.). So therefore we may measure dependence among $\epsilon_1, \ldots, \epsilon_L$ to infer conditional dependence among $Y_1, \ldots, Y_L$.

Toy Problem – *Conditional* Dependence

X_1	X_2	Y_1	Y_2	$P(\mathbf{x}, \mathbf{y})$
0	0	0	0	0.25
0	1	0	1	0.25
1	0	0	1	0.25
1	1	1	0	0.25
...	...	...	...	0

Is there conditional dependence between labels Y_1 and Y_2?

Toy Problem – *Conditional* Dependence

X_1	X_2	Y_1	Y_2	$P(\mathbf{x}, \mathbf{y})$
0	0	0	0	0.25
0	1	0	1	0.25
1	0	0	1	0.25
1	1	1	0	0.25
...	...	...	...	0

Is there conditional dependence between labels Y_1 and Y_2? We may test the conditional distributions, independence if:

$$P_{\mathbf{x}}(Y_1, Y_2) = P_{\mathbf{x}}(Y_1) \cdot P_{\mathbf{x}}(Y_2)$$
$$P(Y_2|x_1, x_2) = P(Y_2|Y_1, x_1, x_2)$$

Toy Problem – *Conditional* Dependence (cont. ...)

Supposing that $P(X_j) = 0.5$ (for both) and

X_1	X_2	$P(Y_2 \mid X_1, X_2)$
0	0	0.0
0	1	1.0
1	0	1.0
1	1	0.0
...	...	0.0

X_1	X_2	Y_1	$P(Y_2 \mid X_1, X_2, Y_1)$
0	0	0	0.0
0	1	0	1.0
1	0	0	1.0
1	1	1	0.0
...	...	...	0.0

then we see that for $\mathbf{x} = [1, 0]$,

$$P(Y_2 \mid \mathbf{x}) = \sum_{y_1 \in \{0,1\}} P(Y_2 \mid \mathbf{x}, y_1)$$

$$1 = 1 + 0 = 1$$

so there is conditional independence; $Y_2 \perp\!\!\!\perp Y_1 \mid X_1, X_2$.

The Logical Problem

Example (The Logical Toy Problem)

X_1	X_2	OR Y_1	AND Y_2	XOR Y_3
0	0	0	0	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	0

- Each label can be determined from the input independently of other labels

The Logical Problem

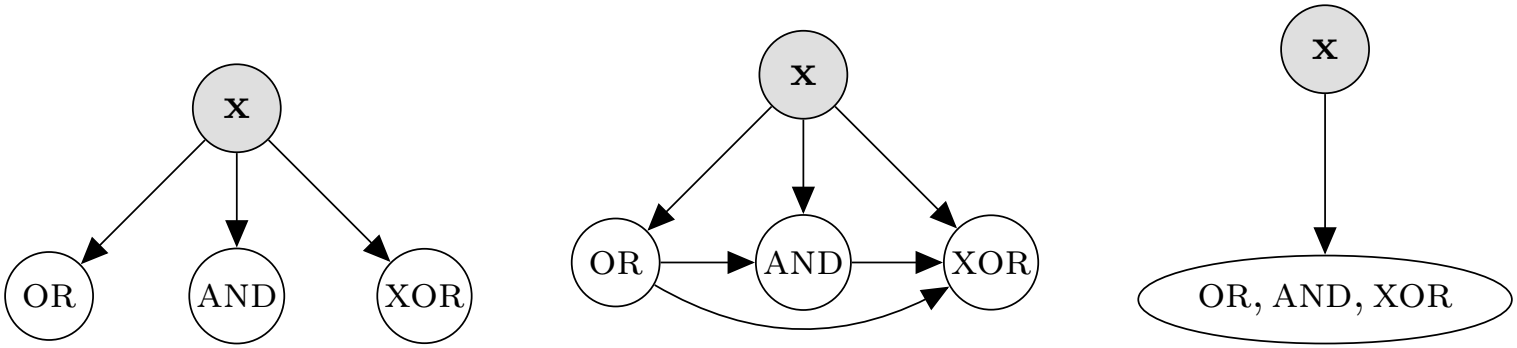


Figure: BR (left), CC (middle), LP (right)

Table: The Logical problem, base classifier logistic regression.

Metric	BR	CC	LP
Hamming score	0.83	1.00	1.00
Exact match	0.50	1.00	1.00

- "Each label can be determined from the input independently of other labels" ...
- ... 'can be' ... given the right base predictive model!

$P(Y_2|X_1, X_2, Y_1) \neq \hat{P}(Y_2|X_1, X_2, Y_1)$

The predictive model is built on[4] $\hat{P}$.

X_1	X_2	Y_1	$P(Y_2\|X_1, X_2)$
0	0	0	0.0
0	1	0	1.0
1	0	0	1.0
1	1	1	0.0
...	...	...	0.0

X_1	X_2	Y_1	$\hat{P}(Y_2\|X_1, X_2)$
0	0	0	1.0
0	1	0	1.0
1	0	0	1.0
1	1	1	0.0
...	...	...	0.0

X_1	X_2	Y_1	$P(Y_2\|X_1, X_2, Y_1)$
0	0	0	0.0
0	1	0	1.0
1	0	0	1.0
1	1	1	0.0
...	...	...	0.0

X_1	X_2	Y_1	$\hat{P}(Y_2\|X_1, X_2, Y_1)$
0	0	0	0.0
0	1	0	1.0
1	0	0	1.0
1	1	1	0.0
...	...	...	0.0

and it may be estimated badly; not enough training data, or iterations, or we were using a rough heuristic (e.g., *greedy* search) to search the probability tree quickly.

[4] Implicitly, except on this slide

The Logical Problem

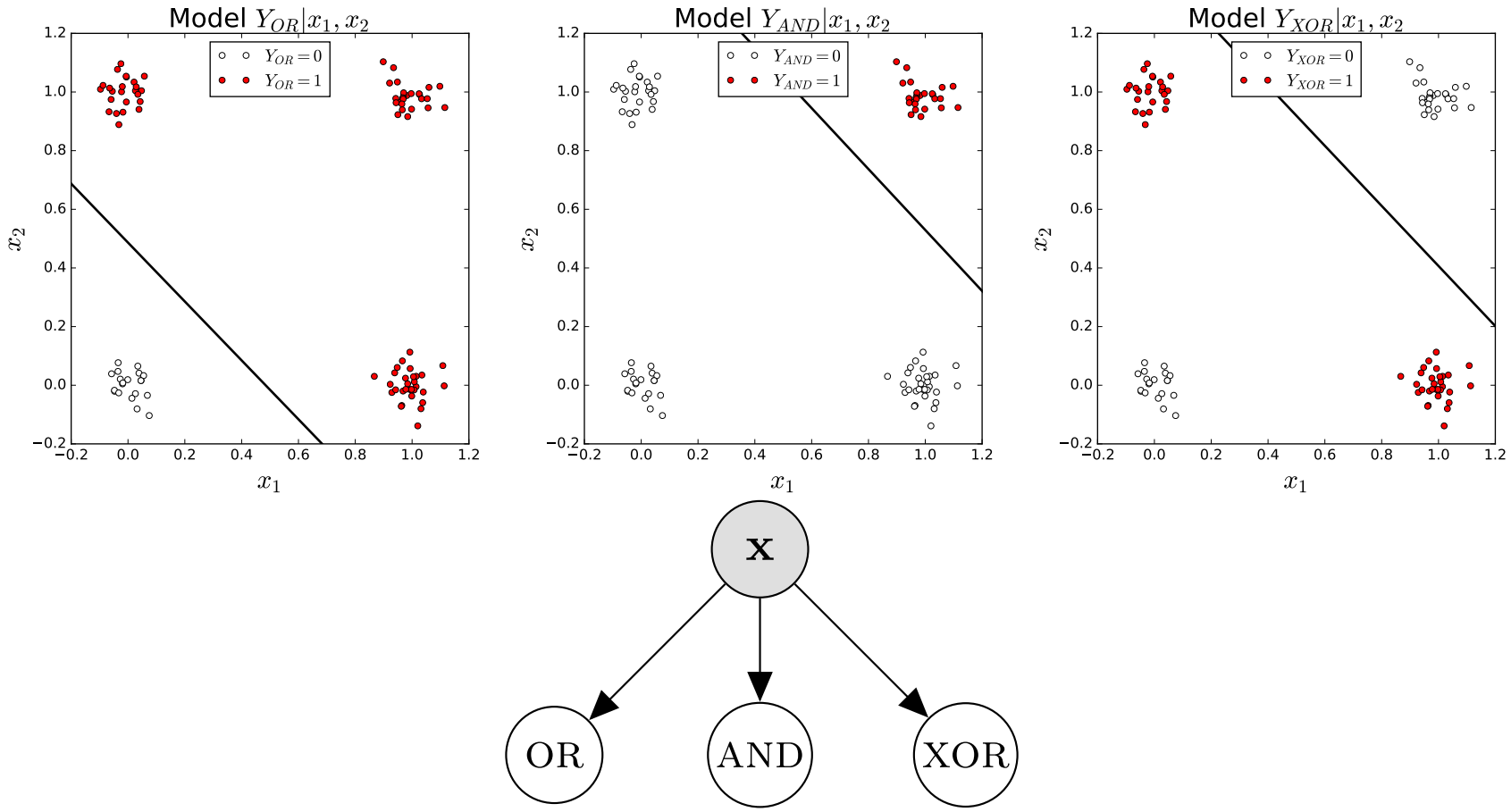


Figure: Independent models with linear decision boundaries (solid line, estimated with logistic regression) not viable for Y_{XOR} label

Solution via Structure

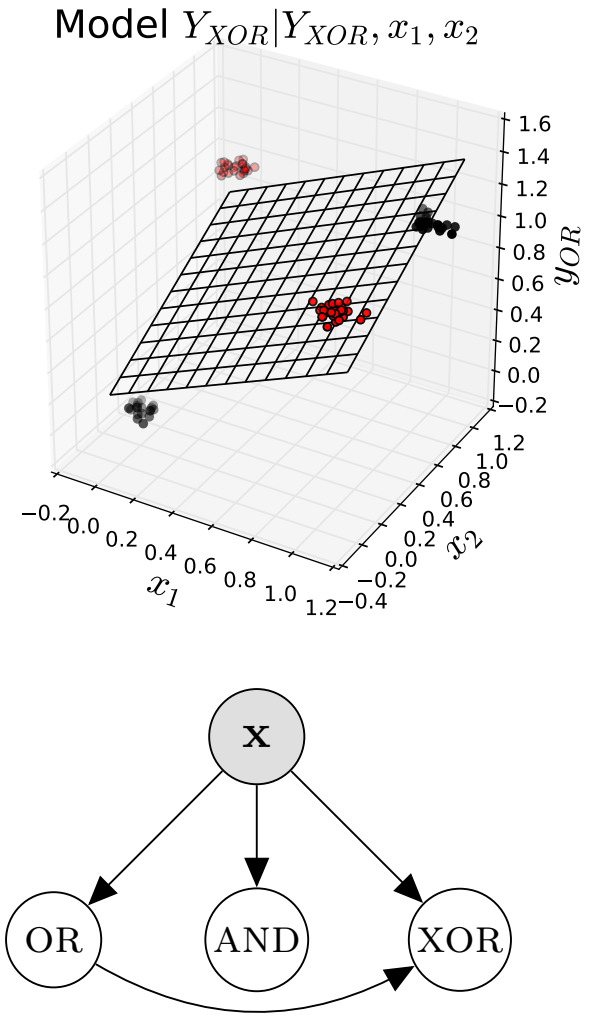
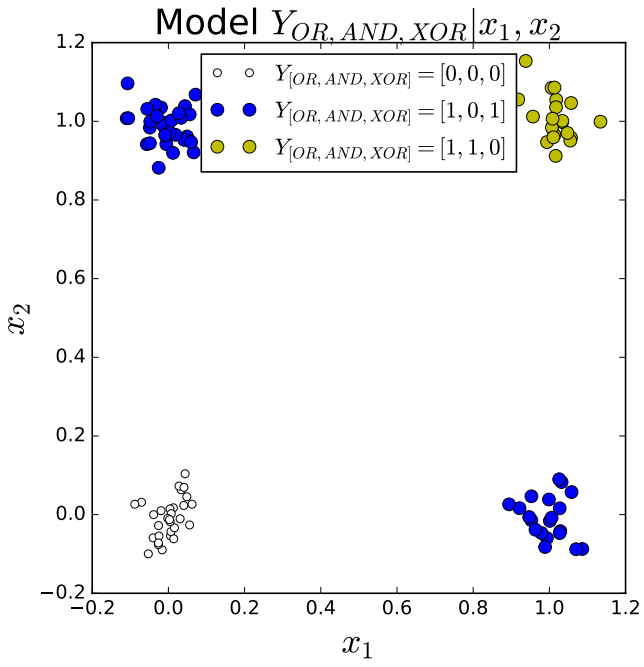


Figure: Classifier chains (CC): linear model now applicable to Y_{XOR}

Solution via Multi-class Decomposition



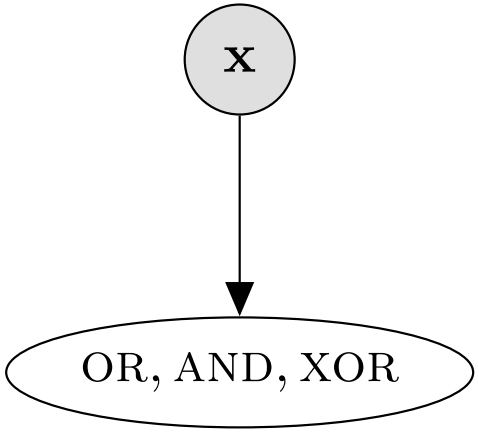


Figure: Modelling labels together in a single class variable taking 2^L values: solvable with one-vs-one multi-class decomposition for any (e.g., linear) base classifier.

Solution via Conditional Independence

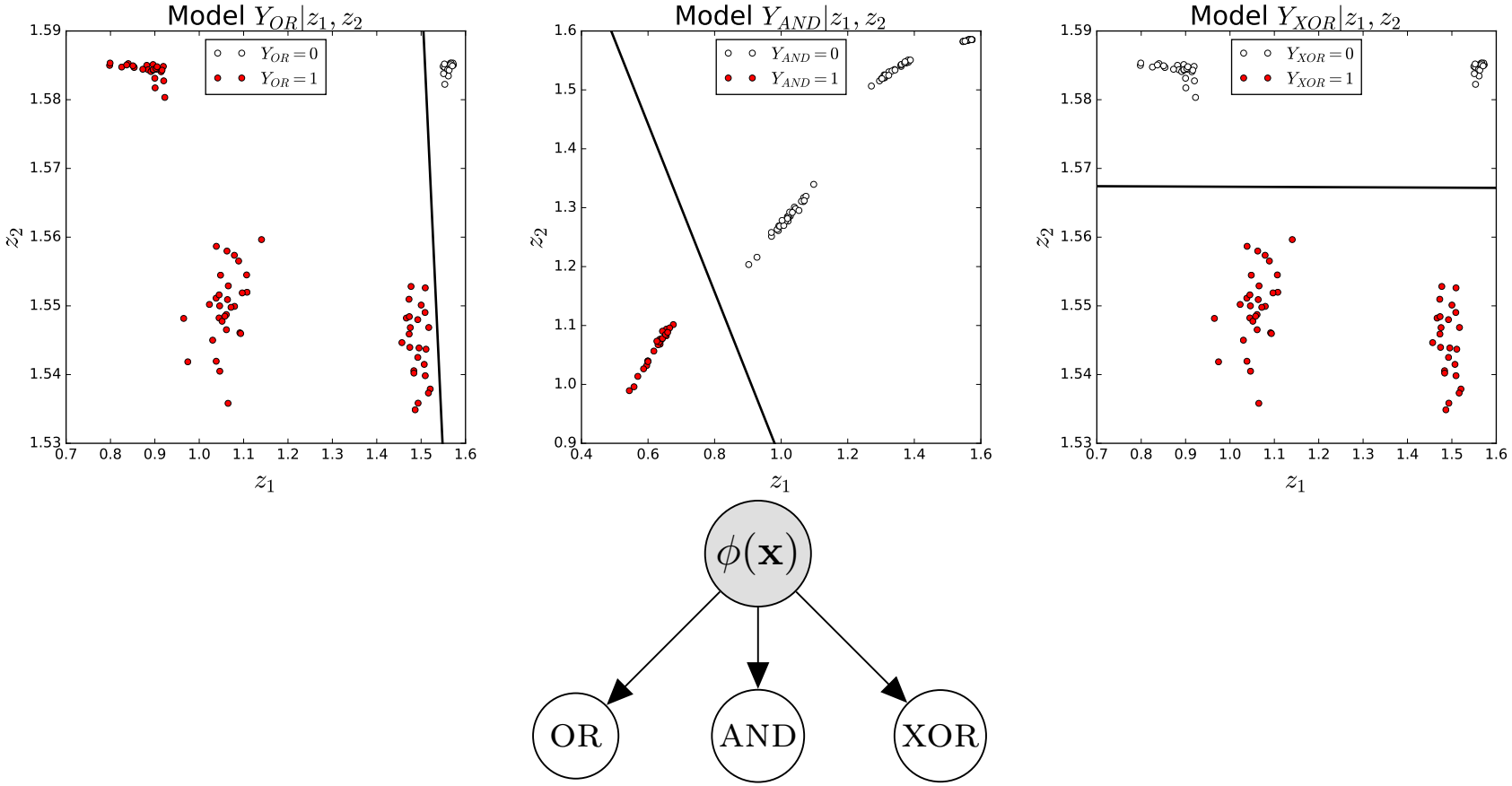


Figure: Solution via latent structure (e.g., random RBF) to new input space $\mathbf{z}$; creating independence: $P(y_{\text{XOR}}|\mathbf{z}, y_{\text{OR}}, y_{\text{AND}}) \approx P(y_{\text{XOR}}|\mathbf{z})$.

Solution via Suitable Base-classifier

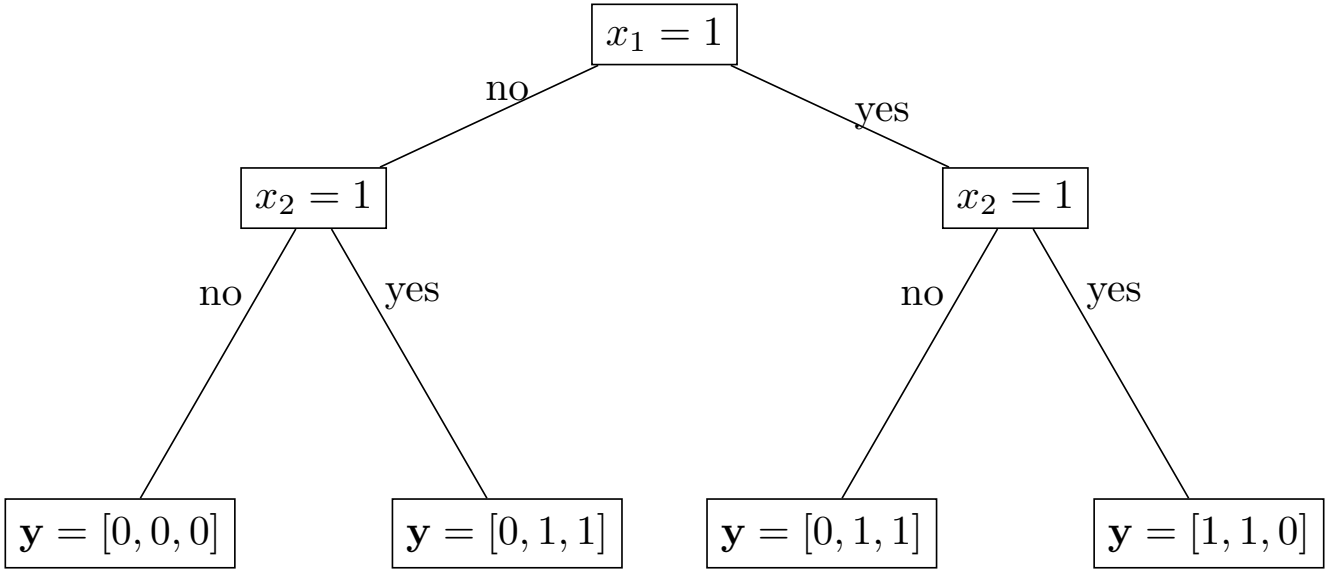
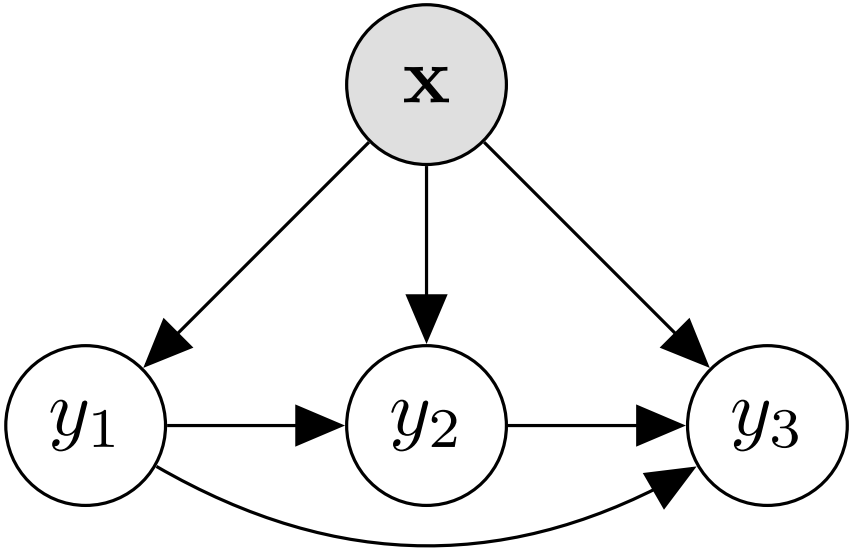


Figure: Solution via non-linear classifier (e.g., Decision Tree). Leaves hold examples, where $\mathbf{y} = [y_{\text{AND}}, y_{\text{OR}}, y_{\text{XOR}}]$

Outline

1. Introduction
2. Applications
3. Probabilistic Models
4. Evaluation Metrics
5. Label Dependence and Structure
6. Inference in Probabilistic Models
7. (Multi-output) Neural Networks
8. (Multi-output) k-Nearest Neighbours
9. (Multi-output) Decision Trees
10. Big Multi-Labelled Data
11. Summary

Inference

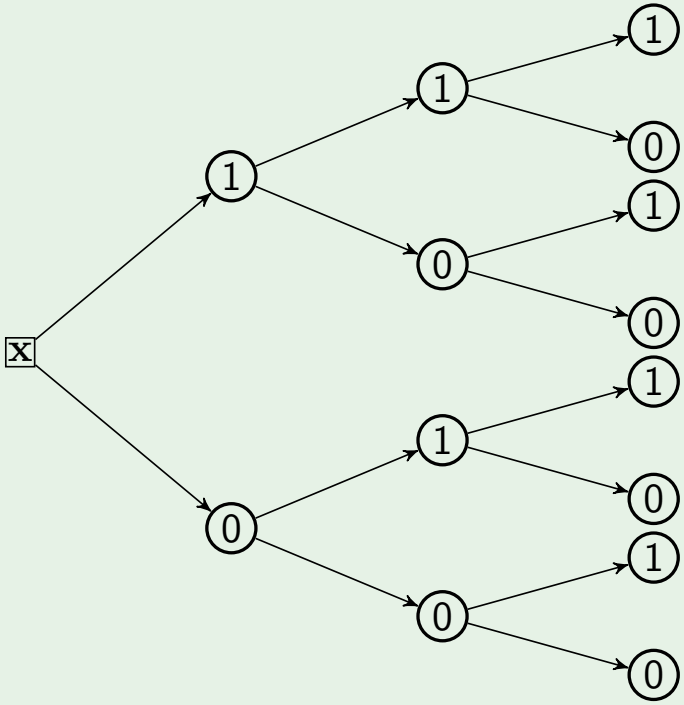


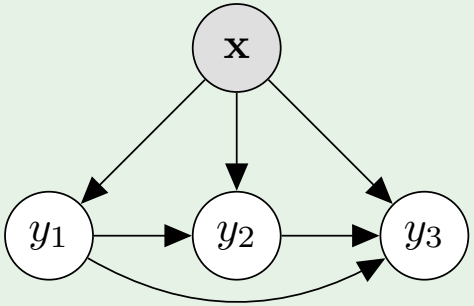
- We have trained out base models $\mathbf{h} = (h_1, h_2, h_3)$
- Now we observe a new $\mathbf{x}$. We formulate a query for our model:

$$\hat{\mathbf{y}} = \mathbf{h}(\mathbf{x}) = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}} P(\mathbf{y}|\mathbf{x})$$

What is the best $\hat{\mathbf{y}}$?

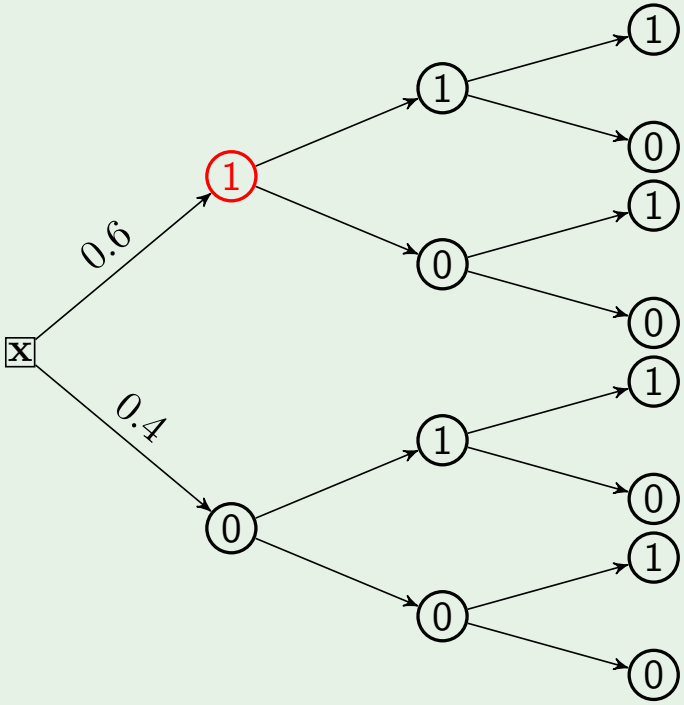
Greedy Search

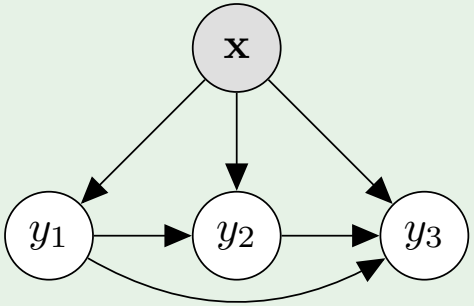




$$\hat{\mathbf{y}} = \mathbf{h}(\mathbf{x}) = [?, ?, ?]$$

Greedy Search

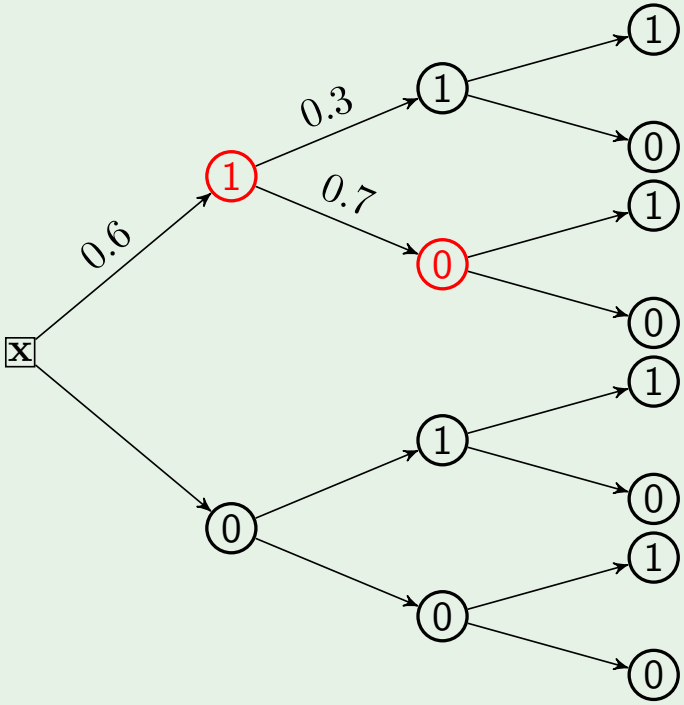


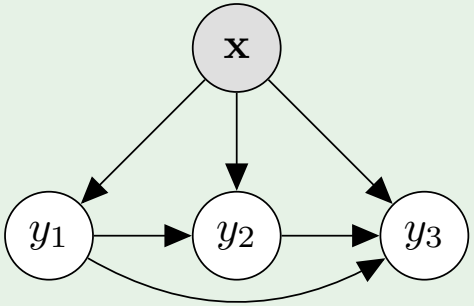


1. $\hat{y}_1 = h_1(\mathbf{x}) = \text{argmax}_{y_1} P(y_1|\mathbf{x}) = 1$

$\hat{\mathbf{y}} = \mathbf{h}(\mathbf{x}) = [1, ?, ?]$

Greedy Search

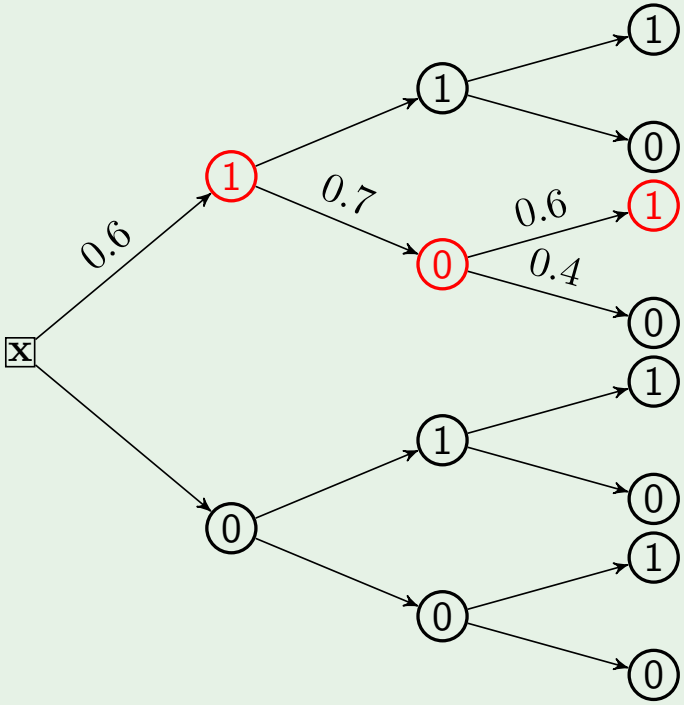


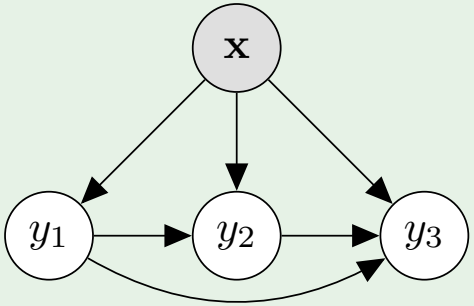


1. $\hat{y}_1 = h_1(\mathbf{x}) = \text{argmax}_{y_1} P(y_1|\mathbf{x}) = 1$
2. $\hat{y}_2 = h_2(\mathbf{x}, \hat{y}_1) = \ldots = 0$

$\hat{\mathbf{y}} = \mathbf{h}(\mathbf{x}) = [1, 0, ?]$

Greedy Search

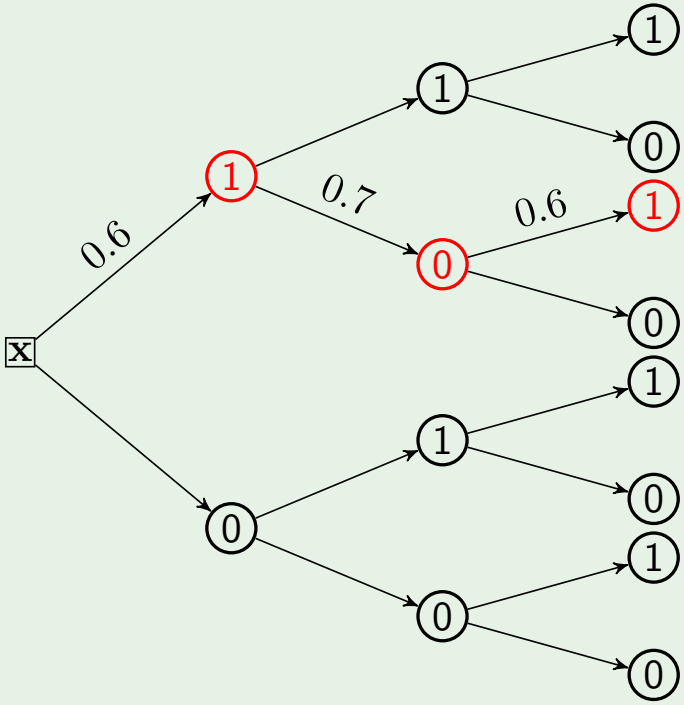


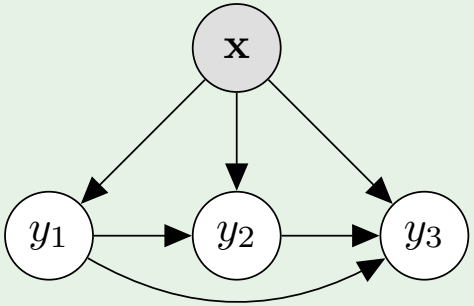


1. $\hat{y}_1 = h_1(\mathbf{x}) = \text{argmax}_{y_1} P(y_1|\mathbf{x}) = 1$
2. $\hat{y}_2 = h_2(\mathbf{x}, \hat{y}_1) = \ldots = 0$
3. $\hat{y}_3 = h_3(\mathbf{x}, \hat{y}_1, \hat{y}_2) = \ldots = 1$

$\hat{\mathbf{y}} = \mathbf{h}(\mathbf{x}) = [1, 0, 1]$

Greedy Search





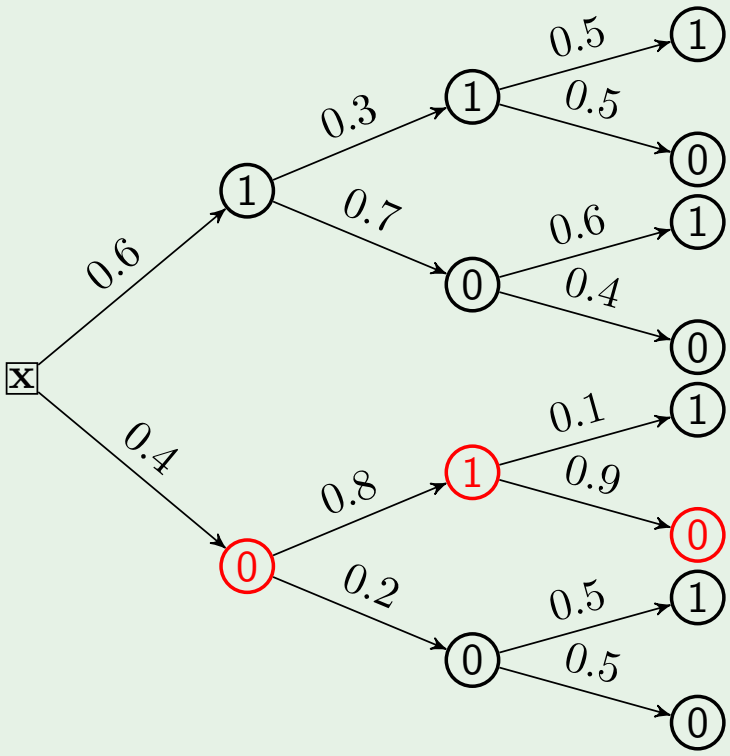
1. $\hat{y}_1 = h_1(\mathbf{x}) = \text{argmax}_{y_1} P(y_1|\mathbf{x}) = 1$
2. $\hat{y}_2 = h_2(\mathbf{x}, \hat{y}_1) = \ldots = 0$
3. $\hat{y}_3 = h_3(\mathbf{x}, \hat{y}_1, \hat{y}_2) = \ldots = 1$

$$\hat{\mathbf{y}} = \mathbf{h}(\mathbf{x}) = [1, 0, 1]$$

Fast, but susceptible to error proagation, since in fact,

$$\underset{\mathbf{y}\in\{0,1\}^L}{\text{argmax}}\, P(\mathbf{y}|\mathbf{x}) \neq [\underset{y_1\in\{0,1\}}{\text{argmax}}\, P(y_1|\mathbf{x}), \ldots, \underset{y_3\in\{0,1\}}{\text{argmax}}\, P(y_1|\mathbf{x}, \hat{y}_1, \hat{y}_2)]$$

Bayes-Optimal (Full) Search

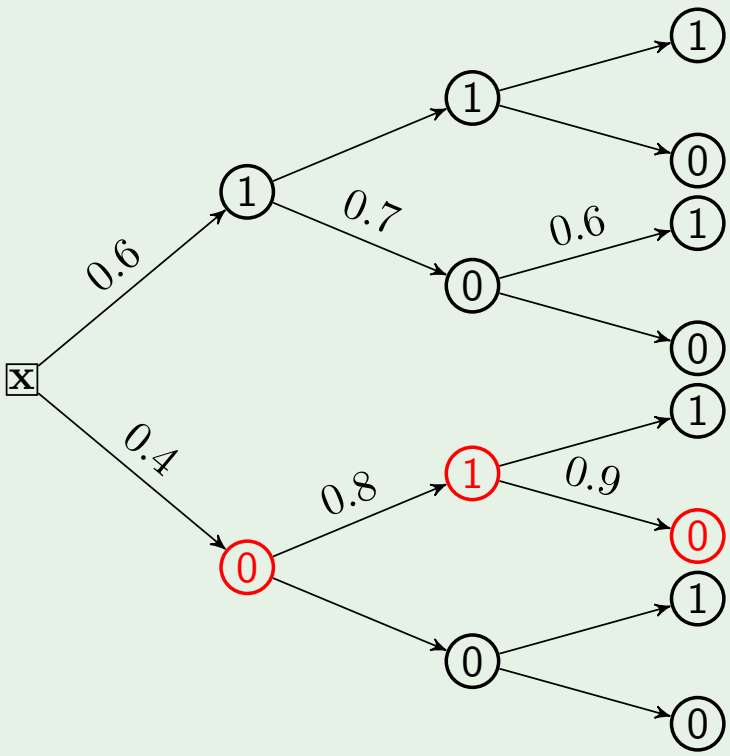


1. $P(\mathbf{y} = [0,0,0]|\mathbf{x}) = 0.040$
2. $P(\mathbf{y} = [0,0,1]|\mathbf{x}) = 0.040$
3. $P(\mathbf{y} = [0,1,0]|\mathbf{x}) = 0.288$
4. ...
6. $P(\mathbf{y} = [1,0,1]|\mathbf{x}) = 0.252$
7. ...
8. $P(\mathbf{y} = [1,1,1]|\mathbf{x}) = 0.090$

return $\text{argmax}_{\mathbf{y}}\, P(\mathbf{y}|\mathbf{x})$

- Search space of $\{0,1\}^L$ paths is too much

Monte-Carlo Search / Likelihood Weighting

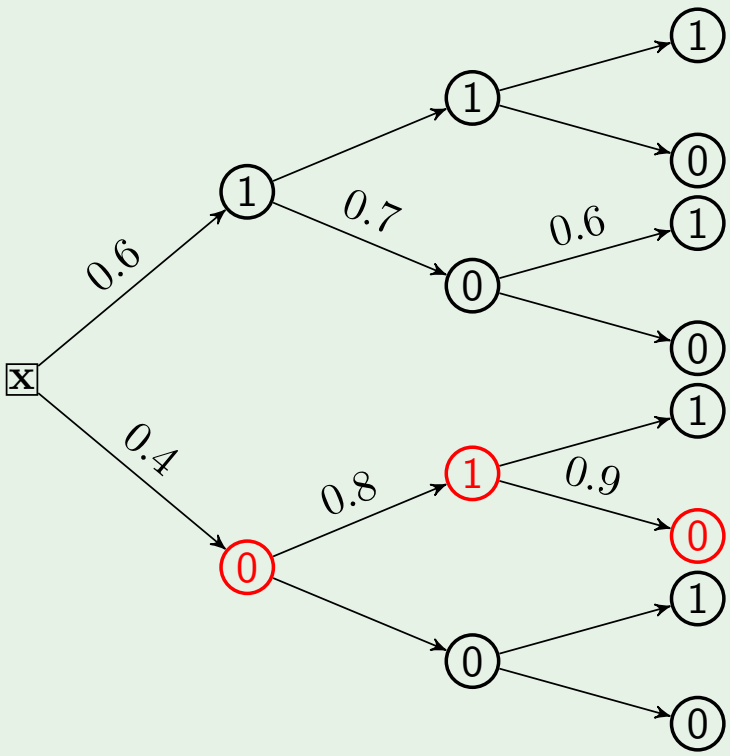


Generate samples $\{\mathbf{y}_t\}_{t=1}^{T}$, directly from P, where

$$y_1^{(t)} \sim P(y_1|\mathbf{x})$$
$$y_2^{(t)} \sim P(y_2|\mathbf{x}, y_1^{(t)})$$
$$y_3^{(t)} \sim P(y_3|\mathbf{x}, y_1^{(t)}, y_2^{(t)})$$

provides $\mathbf{y}_t = [y_1^{(t)}, y_2^{(t)}, y_3^{(t)}]$.

Monte-Carlo Search / Likelihood Weighting



And thus,

- $P([1,0,1]|\mathbf{x}) = 0.252$
- $P([0,1,0]|\mathbf{x}) = 0.288$

$$\text{return } \operatorname*{argmax}_{\mathbf{y}_t} P(\mathbf{y}_t|\mathbf{x})$$

Note that,

$$\lim_{T\to\infty} \texttt{count}(\mathbf{y}_t)/T = P(\mathbf{y}|\mathbf{x})$$

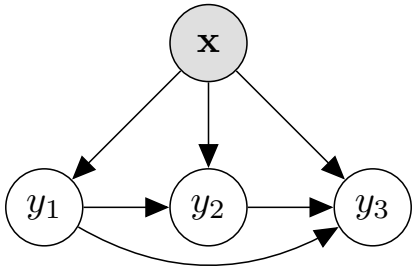
hence at the limit, we expect 28.8% of samples to be $[0,1,0]$.

Outline

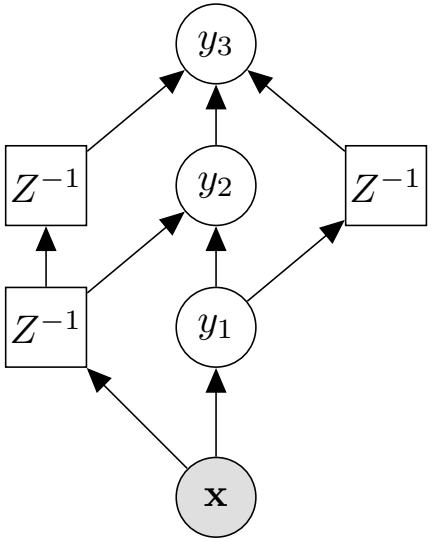
1. Introduction
2. Applications
3. Probabilistic Models
4. Evaluation Metrics
5. Label Dependence and Structure
6. Inference in Probabilistic Models
7. (Multi-output) Neural Networks
8. (Multi-output) k-Nearest Neighbours
9. (Multi-output) Decision Trees
10. Big Multi-Labelled Data
11. Summary

A Multi-label Classifier as a Neural Network

We can move from the probabilistic graphical model view of



to a neural network view . . .

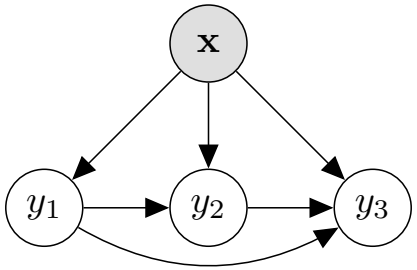


Where Z^{-1} is a delay node

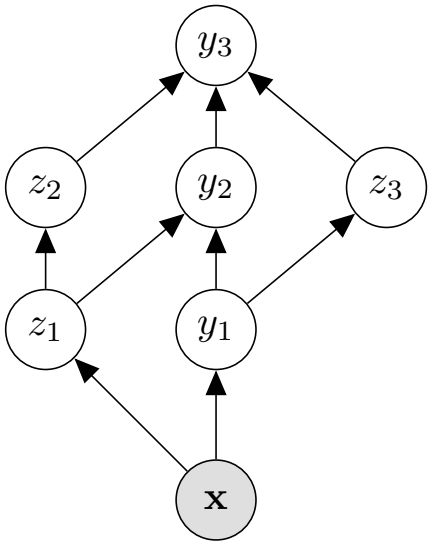
with forward propagation. This can be generalized . . .

A Multi-label Classifier as a Neural Network

We can move from the probabilistic graphical model view of



to a neural network view ...



...or nodes where $w_j = 1$ and linear activation function, such that $y_2 = h(w_j \cdot x, y_1) = h(x, y_1)$.

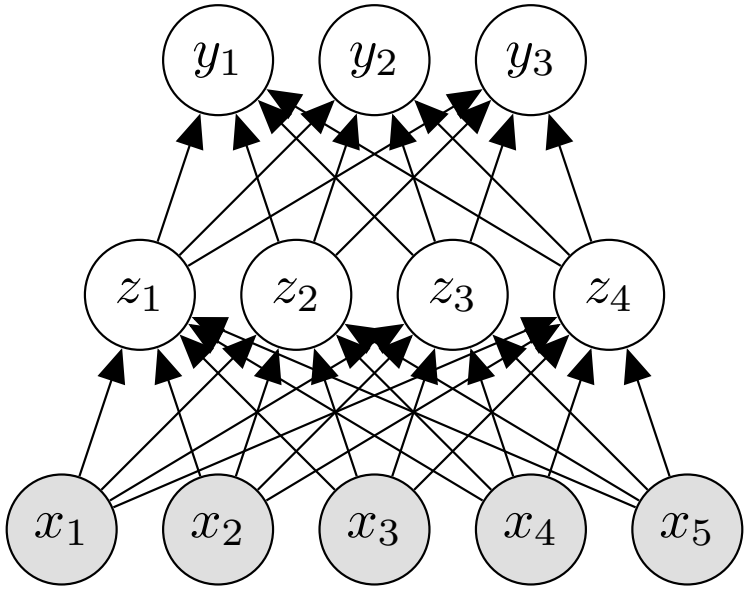
with forward propagation. This can be generalized ...

Multi-layer Perceptron

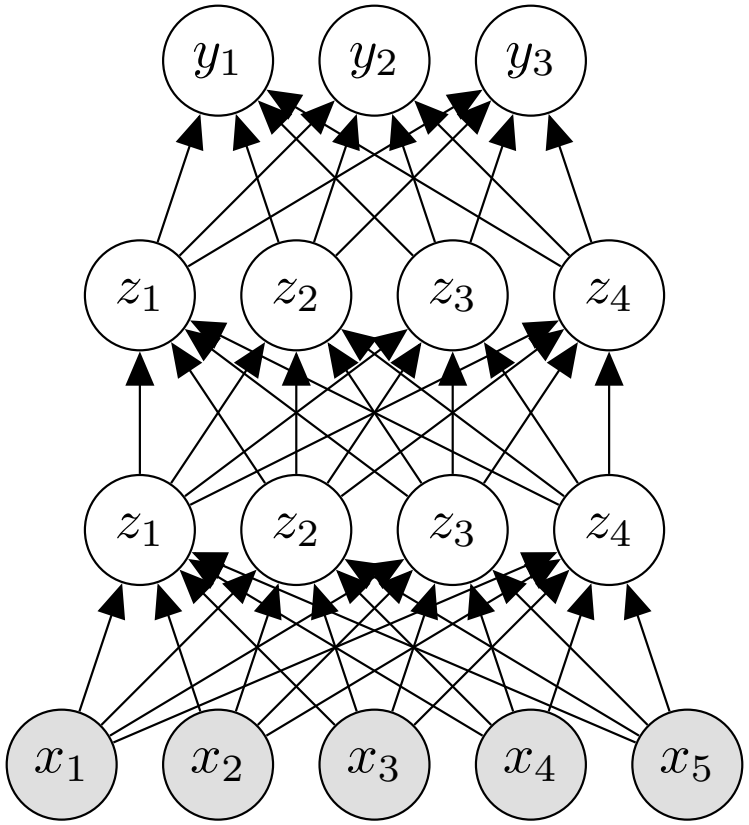
- We propagate upward, such that

$$z_k = f({\mathbf{w}_k^{(1)}}^\top \mathbf{x}) \quad \text{and then} \quad y_j = f({\mathbf{w}_k^{(2)}}^\top \mathbf{x})$$

- Usually we consider the softmax function so that outputs sum to 1. In multi-label classification, there is no need for this. We can just use a logistic loss.
- Train with, e.g., gradient descent + error back-propagation

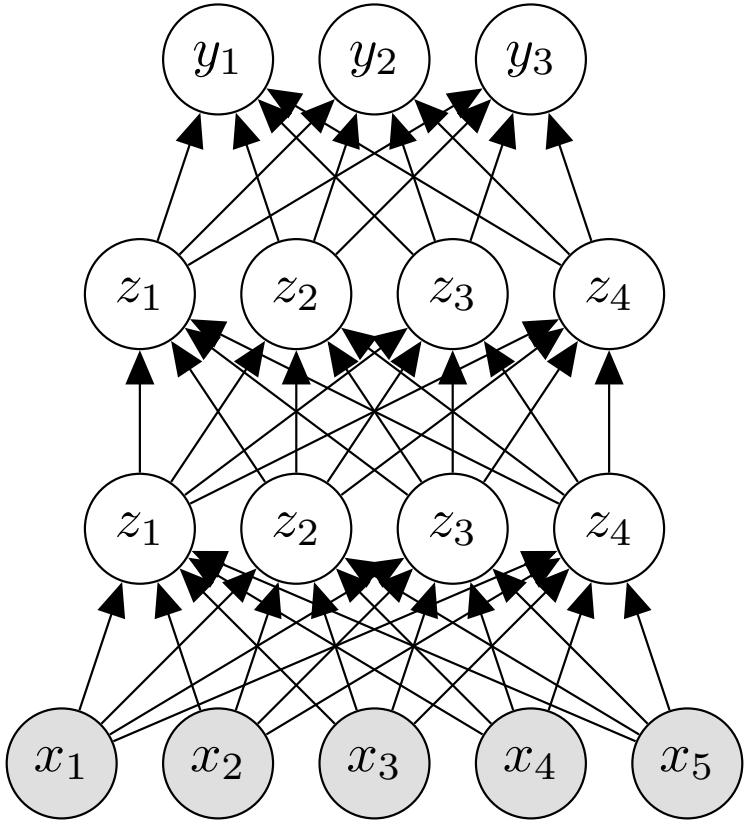


Multi-output Neural Networks



- Add many layers = deep learning scheme.
- Engineer features from the input *or the label space* (e.g., meta labels);
- Learn our own hidden nodes with back propagation

Multi-output Neural Networks



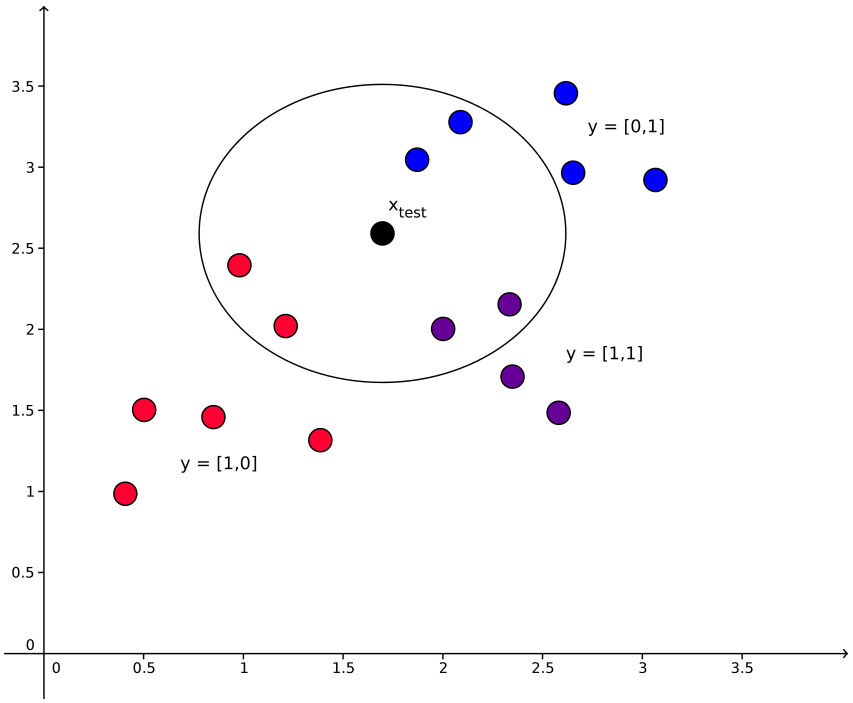
- Label dependence is removed by inner layers,
- May be effective, but many iterations to train, how many nodes, layers; what learning rate?, etc. (hyper-parameter tuning).

Outline

1. Introduction
2. Applications
3. Probabilistic Models
4. Evaluation Metrics
5. Label Dependence and Structure
6. Inference in Probabilistic Models
7. (Multi-output) Neural Networks
8. (Multi-output) k-Nearest Neighbours
9. (Multi-output) Decision Trees
10. Big Multi-Labelled Data
11. Summary

Multi-label k-Nearest Neighbours

In kNN, we compare each test instance $\mathbf{x}$ to the k-nearest instances in the training set (i.e., its neighbourhood).



Multi-label k-Nearest Neighbours

In kNN, we compare each test instance $\mathbf{x}$ to the k-nearest instances in the training set (i.e., its neighbourhood). In the single-label binary case:

$$P(Y = 1|\mathbf{x}) = \frac{1}{k} \sum_{n \in \mathrm{Ne}_k^{\mathbf{x}}} y_n$$

where $\mathrm{Ne}_k^{\mathbf{x}}$ contains [the indices of] the k *closest* points to $\mathbf{x}$ (e.g., Euclidean distance). Classification can be made with a threshold,

$$\hat{y} = h(\mathbf{x}) = \begin{cases} 1 & P(Y = 1|\mathbf{x}) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Multi-label k-Nearest Neighbours

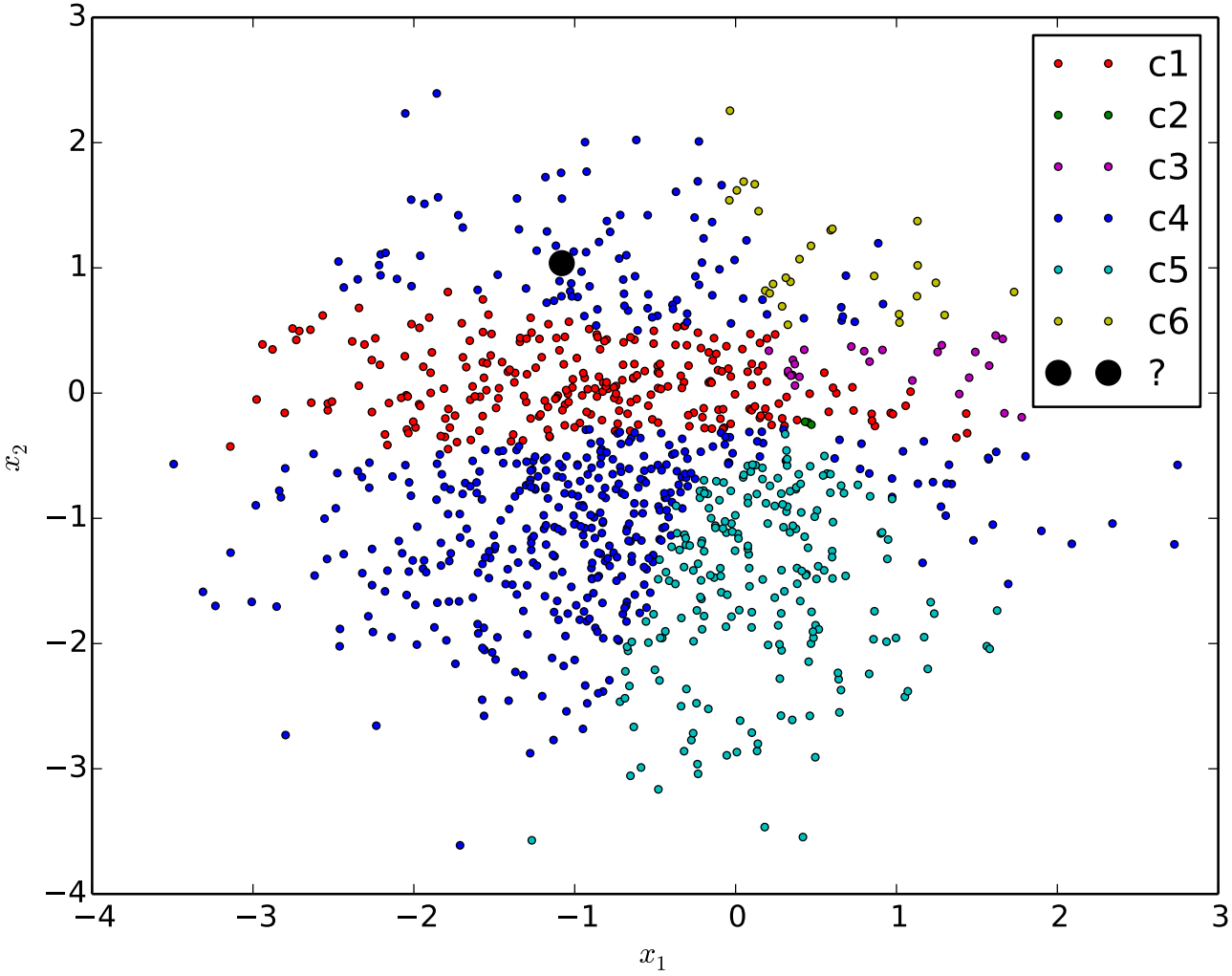
In kNN, we compare each test instance $\mathbf{x}$ to the k-nearest instances in the training set (i.e., its neighbourhood). For the j-th label (in the multi-label case):

$$P(Y_j = 1|\mathbf{x}) = \frac{1}{k} \sum_{n \in \mathrm{Ne}_k^{\mathbf{x}}} y_j^{(n)}$$

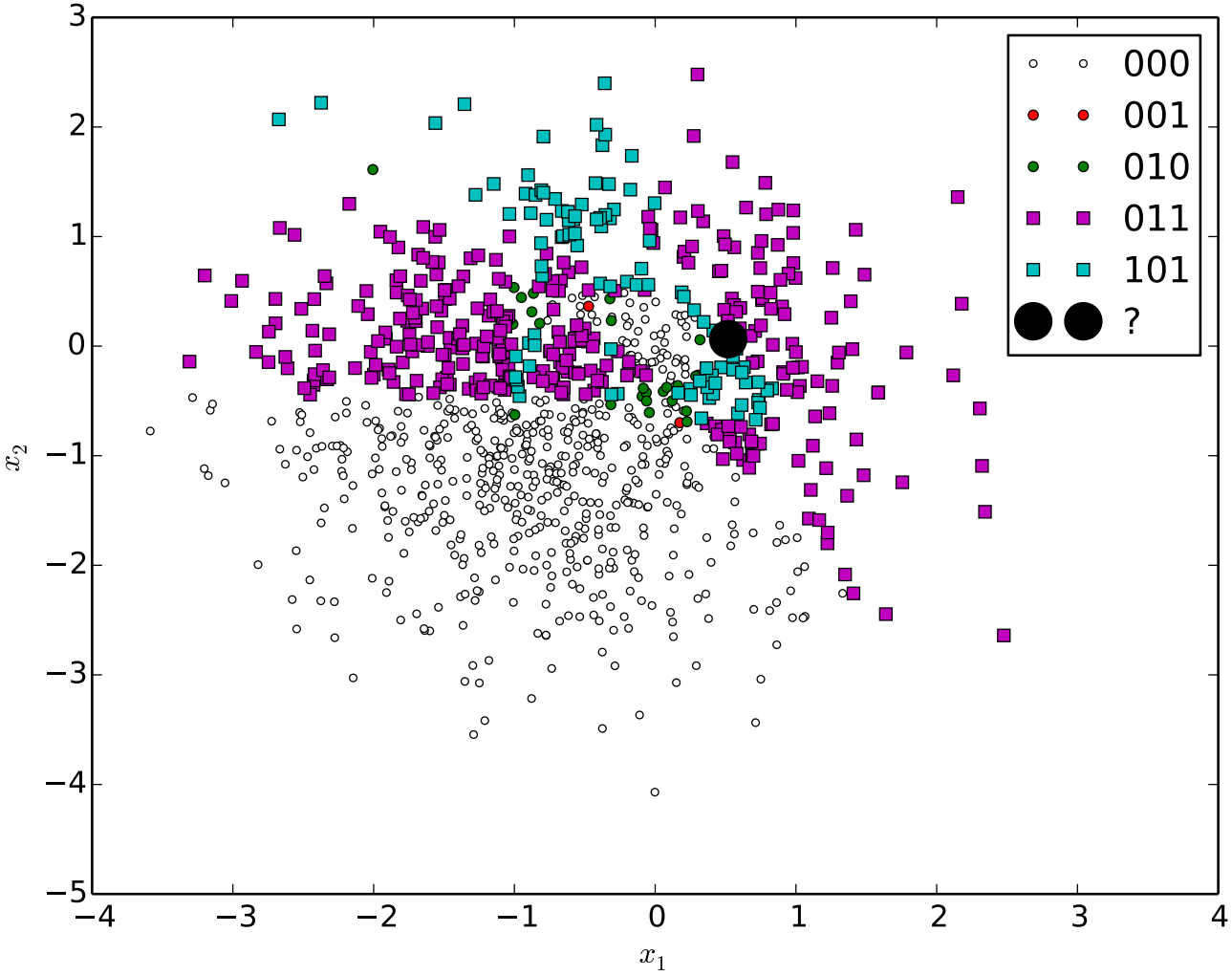
and classification can be made with a threshold,

$$\hat{y}_j = h_j(\mathbf{x}) = \begin{cases} 1 & P(Y_j = 1|\mathbf{x}) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

and we return $\hat{\mathbf{y}} = [\hat{y}_1, \ldots, \hat{y}_L]$.



Single-label *k*NN

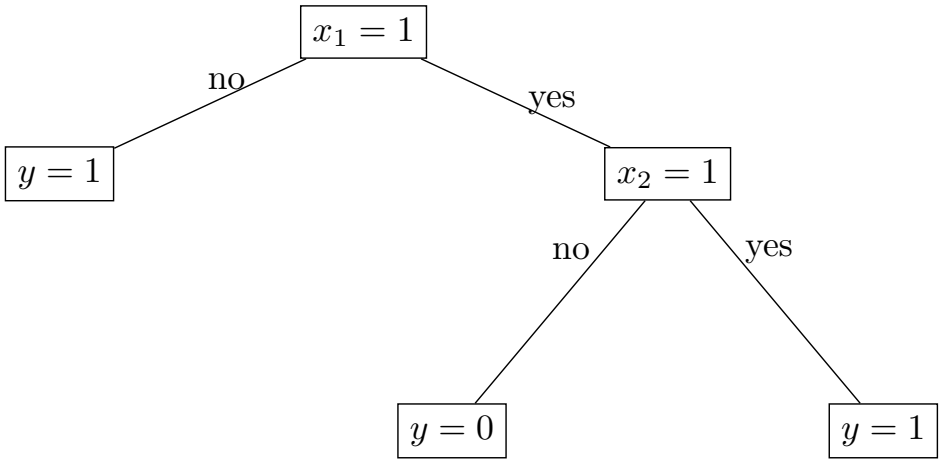


Multi-label kNN

Outline

1. Introduction
2. Applications
3. Probabilistic Models
4. Evaluation Metrics
5. Label Dependence and Structure
6. Inference in Probabilistic Models
7. (Multi-output) Neural Networks
8. (Multi-output) k-Nearest Neighbours
9. (Multi-output) Decision Trees
10. Big Multi-Labelled Data
11. Summary

Decision Trees



X_1	X_2	Y
0	1	1
0	0	1
1	0	0
1	1	1

- Start with dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ at the root
- Iteratively split into subsets $S_{x_1=0}$ and $S_{x_1=1}$, etc., e.g.,
  - first leaf contains set $S =$

X_2	Y
1	1
0	1

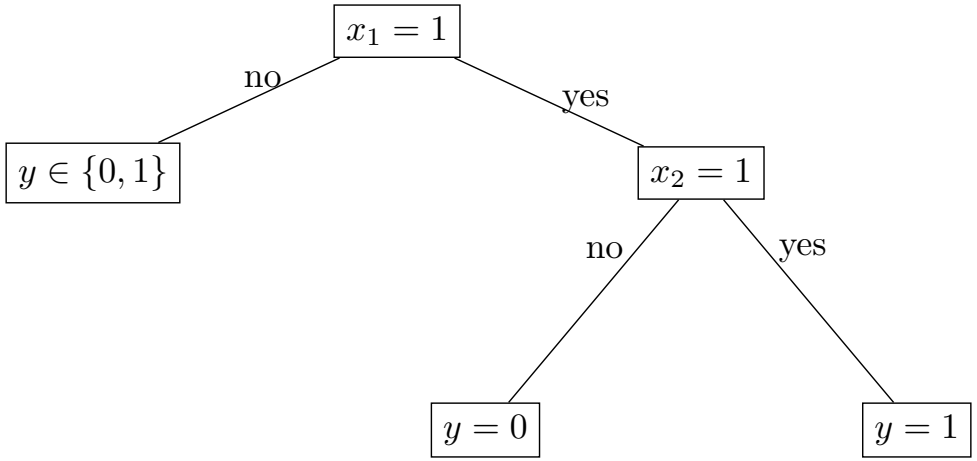
  - second node contains set $S =$

X_2	Y
0	0
1	1

  - second leaf contains $S =$

X_2	Y
0	0

Decision Trees



We do not split randomly; we take the *best* split, e.g., using:

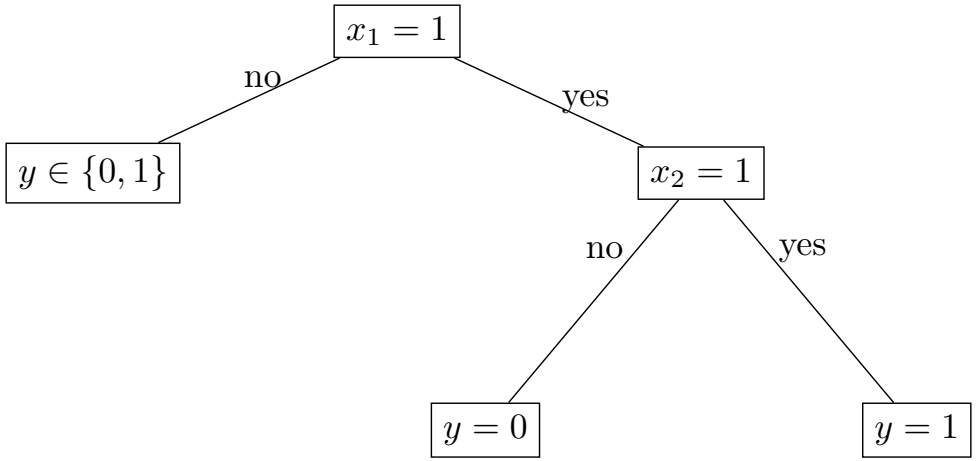
$$H(S) = -\sum_{k \in \{0,1\}} \Big(P(y = k) \log_2 P(y = k) \Big) \quad \triangleright \text{ entropy}$$

$$G(S, X) = H(S) - \underbrace{\sum_{v \in \mathcal{X}} \frac{|S_v|}{|S|} \cdot H(S_v)}_{H(S|X)} \quad \triangleright \text{ information gain}$$

$$S_v = \{(\mathbf{x}, y) \in S | x = v\}$$

where $y \in \{0, 1\}$, $x \in \mathcal{X}$ (for attribute X), e.g., $\mathcal{X} = \{0, 1\}$.

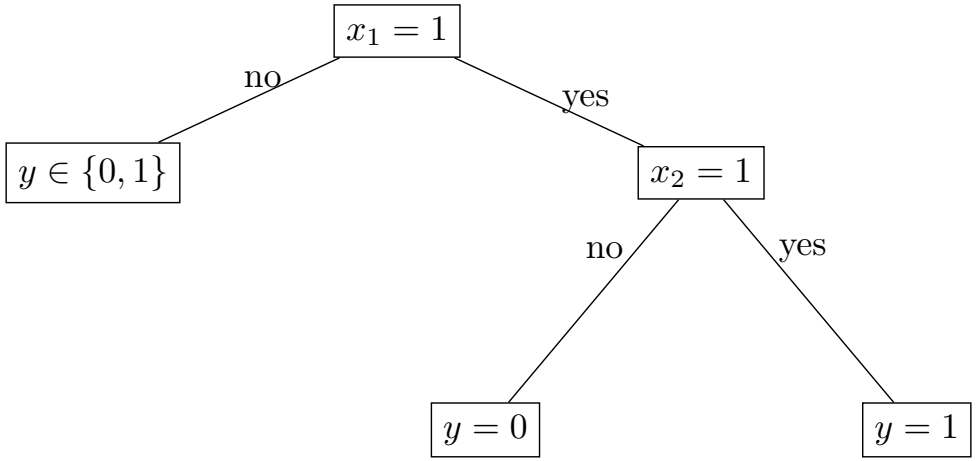
Decision Trees



Induce(S) returns Tree T:

1. if stop(S), return Leaf(S)
2. best $=$ argmax$_{j \in \{1,\ldots,D\}}$ $G(S, X_j)$
3. Create decision node that tests x_{best}
4. For $v \in \{0, 1\}$:
   - $S_v = \{(\mathbf{x}_{\neg\text{best}}, y) | x_{\text{best}} = v\}$
     ▷ (Induce sub-datasets S_v based on $(x_{\text{best}} = v)$)
   - $T_v =$ Induce(S_v)
   - attach T_v to the corresponding branch
5. return Tree (root x_{best}, branches on T_0, T_1)

Decision Trees



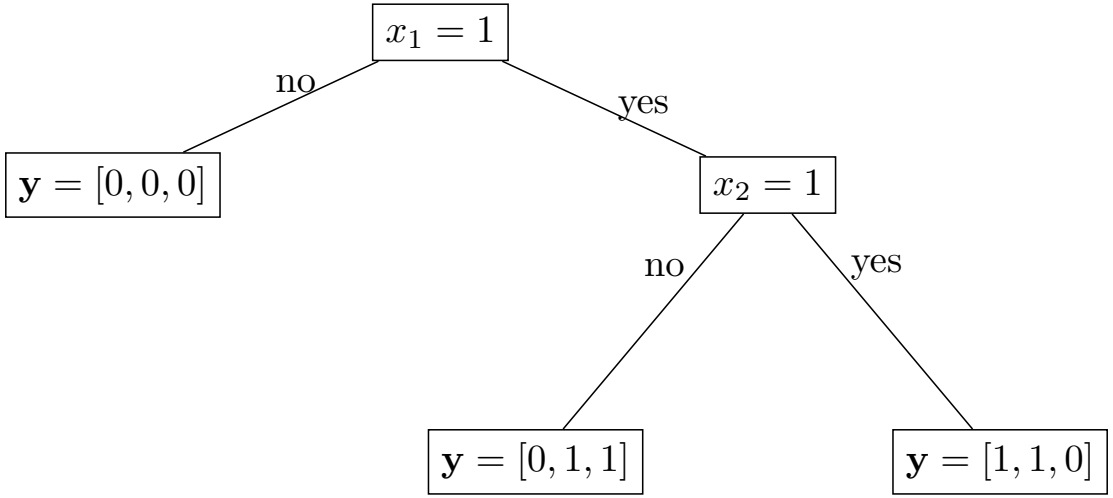
- Non-linear decision boundary
- We may stop early, prune afterwards, etc.
- Interpretable
- Other splitting measures can be used (e.g., Gini impurity)
- Can take the majority vote at inpure leaves.
- Susceptible to overfitting, but works well in an ensemble

Multi-label Decision Trees

Using multi-label entropy,

$$H_{\mathsf{ML}}(S) = -\sum_{j=1}^{L} \sum_{k \in \{0,1\}} P(y_j = k) \log_2 P(y_j = k)$$

$$= -\sum_{j=1}^{L} P(y_j) \Big[\log_2 P(y_j) + [1 - P(y_j)] \log_2 [1 - P(y_j)] \Big]$$



Note that now, $(\mathbf{y}, \mathbf{x}) \in S$, i.e., multi-label vector assignments $\mathbf{y}$ are filtering down to the leaves.

Outline

1. Introduction
2. Applications
3. Probabilistic Models
4. Evaluation Metrics
5. Label Dependence and Structure
6. Inference in Probabilistic Models
7. (Multi-output) Neural Networks
8. (Multi-output) k-Nearest Neighbours
9. (Multi-output) Decision Trees
10. Big Multi-Labelled Data
11. Summary

Large scale text classification

Kaggle Competition

The challenge is based on a large dataset created from Wikipedia. The dataset is multi-class, multi-label and hierarchical. The number of categories is roughly 325,000 and number of the documents is 2,400,000.

Scaling Up

LSHTC4: Large Scale Hierarchal Text Classification

A wikipedia-scale problem

- $325,056$ labels
- $2.4M$ examples

- Even with only $1,000$ features, have to learn over $300M$ parameters with BR (linear models)
- ... plus $52,831M$ more with CC
- ... plus ensembles ($\times 10$, $\times 50$?)
- LP transformation generates around $1.47M$ classes

"Extreme" multi-label classification considers up to millions of labels.

Case study: Winning solution of the LSHTC4 challenge

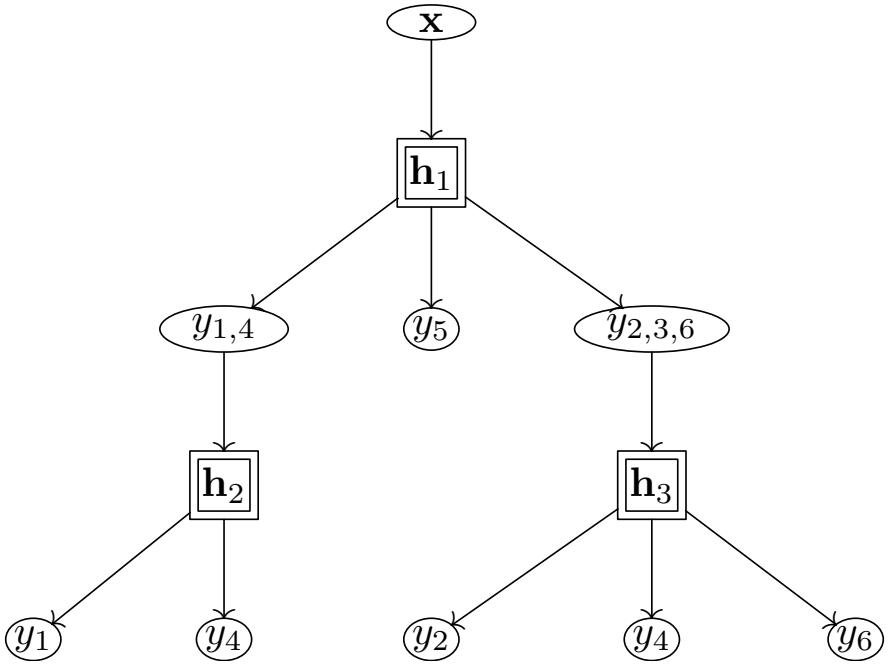
(325,056 labels, 2.4M examples)

1. Ignore the predefined hierarchy
2. Create meta labels
3. ...from random samples of training data
4. ...heavily pruned
5. ...chained together
6. Mix of base classifiers (centroid, decision trees, SVMs)
7. ...randomization on: splits, pruning, chain links, base classifier parameters
8. Train models across 20 machines, combine votes with meta model.

Obtained best performance. No cluster, no GPUs.

Hierarchy of Meta Labels

A useful approach to large-scale multi-label learning:



1. Cluster labels (e.g., randomly, hierarchical k-means, or use pre-defined hierarchy)
2. Apply problem transformation / base classifiers

A significant speed improvement, easy to distribute across machines if necessary.

Outline

1. Introduction
2. Applications
3. Probabilistic Models
4. Evaluation Metrics
5. Label Dependence and Structure
6. Inference in Probabilistic Models
7. (Multi-output) Neural Networks
8. (Multi-output) k-Nearest Neighbours
9. (Multi-output) Decision Trees
10. Big Multi-Labelled Data
11. Summary

Summary

- Multi-label classification: essential for many applications.
  - A generalization of single-label classification
  - A special case of multi-output and structured-output prediction

  Approaches with:
  - Bayesian Networks (and approximations) via *transformation*
  - Neural Networks
  - k-Nearest Neighbours
  - Decision Trees

Classification

Jesse Read

[D&K] IoT Stream Data Mining 2017-2018

December 13, 2017

Multi-label Streams, Concept Drift, and Sequential Data

Jesse Read

[D&K] IoT Stream Data Mining 2017-2018

December 20, 2017

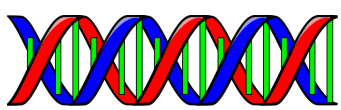
Outline

1. Multi-labelled Streaming Data
2. kNN for Streams
3. Batch-incremental Ensemble
4. Incremental base learner
5. Hoeffding Trees
6. Neural Networks
7. Concept Drift
8. Temporal Dependence
9. Connections to Sequential Data
10. Unlabelled Instances
11. Summary

Multi-label Data Streams

Many applications, e.g.,

- text (email, twitter, web, social networks, . . .)
- images (and video)
- audio
- sensory data (IoT, . . .)
- reinforcement learning (agent in an online environment)
- . . .



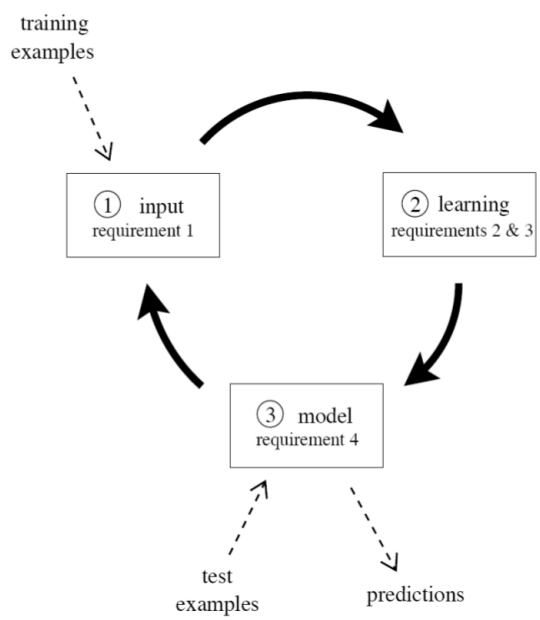
Implications of Data Streams

1. A potentially infinite sequence, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\infty$, but limited memory
2. Prediction $\hat{y}_t$ must be made immediately at time t
3. Average time spent on prediction and learning per-instance – must be less than the average real intra-instance time (i.e., the real time between some $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$)
4. Expect concept drift to occur: the distribution $(\mathbf{x}_t, y_t) \sim p_t(X, Y)$ changes with t.

Data-Stream Classification

Typical loop for data-stream learning, classification, and evaluation. For each instance:

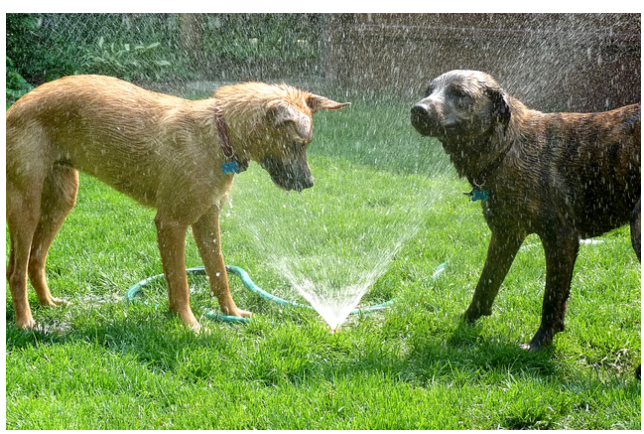
1. Observe $\mathbf{x}$
2. Make a prediction $\hat{y} = h(\mathbf{x})$
3. Observe true label y
4. Measure the error $\epsilon = E(y, \hat{y})$
5. Monitor ϵ signal for concept drift
6. Update the model h with example $(\mathbf{x}, y)$



Implications for Multi-label Classifiers

As compared to the single-labelled case:

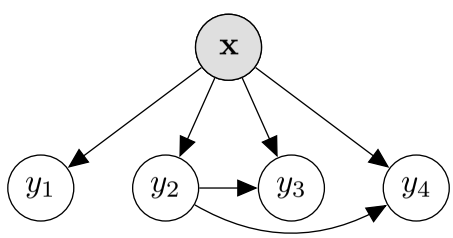
- More computationally complex (label dimension)
- Dynamic label set
- Dynamic label dependence
  (i.e., multi-dimensional concept drift)
- Multi-labelled examples more difficult to obtain
  (L-times more expensive)



Multi-label Learning in Data Streams

A good recipe to perform better than baseline binary relevance method in multi-label classification:

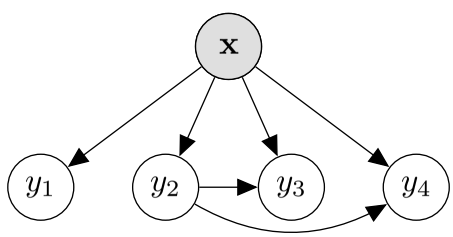
1. Analyze label dependence
2. Search for a good structure
3. Deploy model



Multi-label Learning in Data Streams

A good recipe to perform better than baseline binary relevance method in multi-label classification:

1. ~~Analyze label dependence~~
2. ~~Search for a good structure~~
3. ~~Deploy model~~



No time! Data still arriving! Dependence (and therefore best structure) may become invalidated over time. Model should have been deployed already!

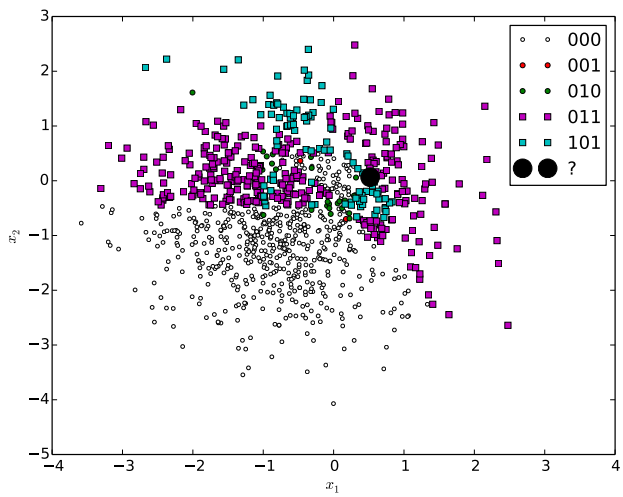
Multi-label Streams Methods

- k-Nearest Neighbours
- A batch-incremental ensemble
- Problem transformation with an incremental base learner
- Decision Rules
- Hoeffding trees
- Neural networks

Outline

1. Multi-labelled Streaming Data
2. kNN for Streams
3. Batch-incremental Ensemble
4. Incremental base learner
5. Hoeffding Trees
6. Neural Networks
7. Concept Drift
8. Temporal Dependence
9. Connections to Sequential Data
10. Unlabelled Instances
11. Summary

Incremental (Multi-label) kNN



kNN is a lazy method, just need to store a batch of instances;

- Compare new example to its neighbours in the batch, make a classification (see slides from last time)
- Add new instances (when labels are available) to the batch, purge out instances (e.g., FIFO) when search is too slow or memory is full.

Outline

1. Multi-labelled Streaming Data
2. kNN for Streams
3. Batch-incremental Ensemble
4. Incremental base learner
5. Hoeffding Trees
6. Neural Networks
7. Concept Drift
8. Temporal Dependence
9. Connections to Sequential Data
10. Unlabelled Instances
11. Summary

Batch-Incremental Ensemble

Build regular models on batches/windows of instances (typically in a [weighted] ensemble).

$$\underbrace{\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_1, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_2, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_3, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_4}_{\text{build } \mathbf{h}_1}, \underbrace{\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_5, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_6, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_7, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_8}_{\text{build } \mathbf{h}_2}, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_9, \begin{bmatrix}\mathbf{x}\\ \hat{\mathbf{y}}\end{bmatrix}_{10}$$

Using a batch size of w examples:

- build model $\mathbf{h}_1$ on $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_w, \mathbf{y}_w)\}$
- build model $\mathbf{h}_2$ on $\{(\mathbf{x}_{w+1}, \mathbf{y}_{w+1}), \ldots, (\mathbf{x}_{2w}, \mathbf{y}_{2w})\}$
- build model $\mathbf{h}_3$ on $\{(\mathbf{x}_{2w+1}, \mathbf{y}_{2w+1}), \ldots, (\mathbf{x}_{3w}, \mathbf{y}_{3w})\}$
- ...

Use the most recent M models; predict via vote :

$$\hat{\mathbf{y}} = \sum_{m=1}^{M} \mathbf{h}_m(\mathbf{x})$$

Batch-Incremental Ensemble

Build regular models on batches/windows of instances (typically in a [weighted] ensemble).

$$\underbrace{\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_1, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_2, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_3, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_4}_{\text{build } \mathbf{h}_1}, \underbrace{\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_5, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_6, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_7, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_8}_{\text{build } \mathbf{h}_2}, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_9, \begin{bmatrix}\mathbf{x}\\ \hat{\mathbf{y}}\end{bmatrix}_{10}$$

Using a batch size of w examples:

- build model $\mathbf{h}_1$ on $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_w, \mathbf{y}_w)\}$
- build model $\mathbf{h}_2$ on $\{(\mathbf{x}_{w+1}, \mathbf{y}_{w+1}), \ldots, (\mathbf{x}_{2w}, \mathbf{y}_{2w})\}$
- build model $\mathbf{h}_3$ on $\{(\mathbf{x}_{2w+1}, \mathbf{y}_{2w+1}), \ldots, (\mathbf{x}_{3w}, \mathbf{y}_{3w})\}$
- ...

Use the most recent M models; predict via weighted vote (more recent = more relevant):

$$\hat{\mathbf{y}} = \sum_{m=1}^{M} \omega_m \cdot \mathbf{h}_m(\mathbf{x}) \quad \triangleright \text{ where } \omega_m \propto m$$

Batch-Incremental Ensemble

Build regular models on batches/windows of instances (typically in a [weighted] ensemble).

$$\underbrace{\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_1, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_2, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_3, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_4}_{\text{build } \mathbf{h}_1}, \underbrace{\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_5, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_6, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_7, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_8}_{\text{build } \mathbf{h}_2}, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_9, \begin{bmatrix}\mathbf{x}\\ \hat{\mathbf{y}}\end{bmatrix}_{10}$$

- A common approach, surprisingly effective
- Open choice of base classifier (e.g., C4.5, SVM, . . .)
- But what batch size to use?
  - Too small = models are too weak
  - Too large = slow to adapt (missing new instances!)

Batch-Incremental Ensemble

Build regular models on batches/windows of instances (typically in a [weighted] ensemble).

$$\underbrace{\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_1, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_2, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_3, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_4}_{\text{build } \mathbf{h}_1}, \underbrace{\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_5, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_6, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_7, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_8}_{\text{build } \mathbf{h}_2}, \begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}_9, \begin{bmatrix}\mathbf{x}\\ \hat{\mathbf{y}}\end{bmatrix}_{10}$$

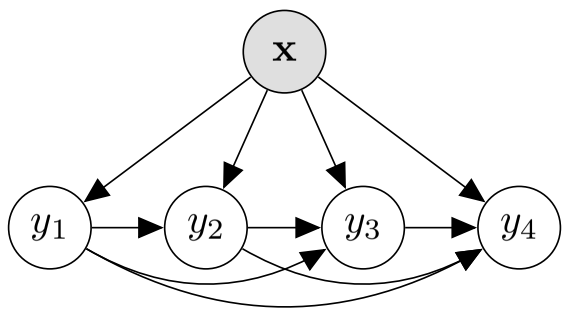
- A common approach, surprisingly effective
- Open choice of base classifier (e.g., C4.5, SVM, . . .)
- But what batch size to use?
  - Too small = models are too weak
  - Too large = slow to adapt (missing new instances!)
- Can have sliding (as opposed to tumbling) windows
  - Extreme case: model built every instance
  - But too many batches = too slow

Outline

1. Multi-labelled Streaming Data
2. kNN for Streams
3. Batch-incremental Ensemble
4. Incremental base learner
5. Hoeffding Trees
6. Neural Networks
7. Concept Drift
8. Temporal Dependence
9. Connections to Sequential Data
10. Unlabelled Instances
11. Summary

Problem Transformation with Incremental Base Learner

Use an incremental learner (Naive Bayes, SGD, . . .) with any problem transformation method (BR, LP, CC, . . .)



- Simple deployment, but
- risk of overfitting (e.g., with classifier chains),
- concept drift may invalidate structure, and a
- limited choice of base learner
  (must be incremental – e.g., decision tree?)

Outline

1. Multi-labelled Streaming Data
2. kNN for Streams
3. Batch-incremental Ensemble
4. Incremental base learner
5. Hoeffding Trees
6. Neural Networks
7. Concept Drift
8. Temporal Dependence
9. Connections to Sequential Data
10. Unlabelled Instances
11. Summary

Rules

For example,

$$\underbrace{X_4 > 1, X_2 \leq 0}_{\text{rule}} \mapsto \underbrace{[1, 0, 1, 0, 0]}_{\mathbf{y}}$$

Can expand rules with the Hoeffding bound: for a variable $x \in R$, the Hoeffding bound states that with probability $1 - \delta$, the true mean of X is at least $\bar{x} - \epsilon$, where

$$\epsilon = \sqrt{\frac{R^2 \ln\left(1/\delta\right)}{2t}}$$

over t examples.

Rules

For example,

$$\underbrace{X_4 > 1, X_2 \leq 0}_{\text{rule}} \mapsto \underbrace{[1, 0, 1, 0, 0]}_{\mathbf{y}}$$

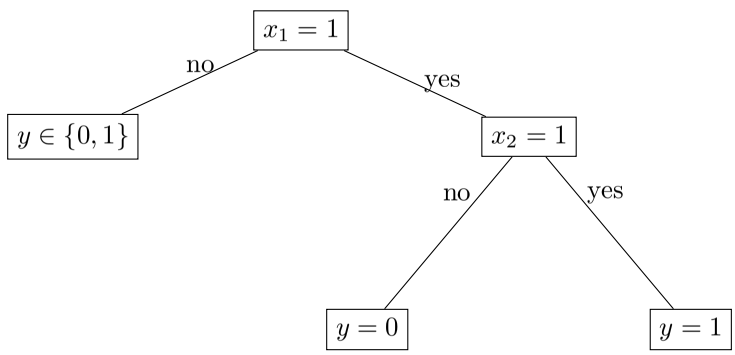
Can expand rules with the Hoeffding bound: for a variable $x \in R$, the Hoeffding bound states that with probability $1 - \delta$, the true mean of X is at least $\bar{x} - \epsilon$, where

$$\epsilon = \sqrt{\frac{R^2 \ln\left(1/\delta\right)}{2t}}$$

over t examples. This gives us an indication of when to expand a rule. (You have to select some small value for δ).

Incremental Decision Trees

Recall decision trees:



We want to construct a tree incrementally, such that the final tree is with high probability, identical to that which a traditional (greedy) method would learn.

On a stream $(\mathbf{x}_t, y_t)|t = 1, \ldots,$ with attributes $\mathbf{X}_\ell = X_1, \ldots, X_{D_\ell}$ at leaf ℓ (initially the root):

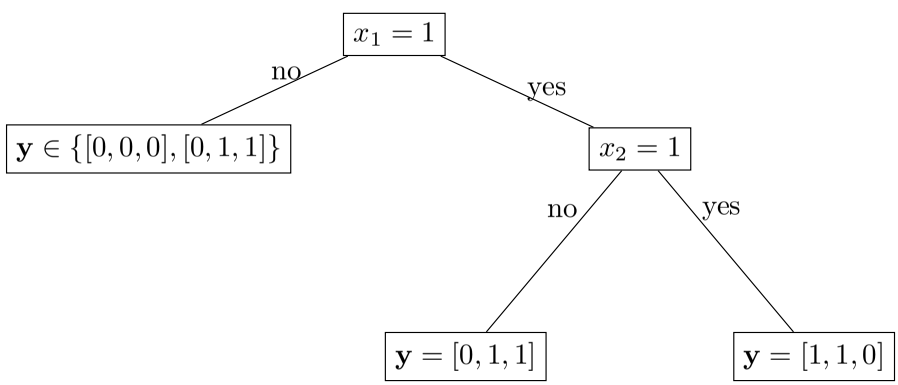
HOEFFDING TREE INDUCTION ALGORITHM (for each instance):

1. Sort $(\mathbf{x}_t, y_t)$ into leaf ℓ
2. Update *sufficient statistics* (to calculate gain $\bar{G}$) in ℓ
3. $n_\ell \leftarrow n_\ell + 1$ ▷ number of examples seen at ℓ
4. If $n_\ell \mod n_{\min} = 0$:
   1. Compute $\bar{G}(X_j)$ for all attributes $\boldsymbol{X}_\ell$ ▷ (in leaf ℓ)
   2. X_a is the attribute with highest $\bar{G}$ and X_b second highest
   3. Compute Hoeffding bound $\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2t}}$
   4. if $\left[\bar{G}(X_a) - \bar{G}(X_b)\right] > \epsilon$, then (knowing that $\Delta G \geq \Delta \bar{G} - \epsilon > 0$ with probability $1 - \delta$):
      1. turn leaf ℓ into node splitting on X_a
      2. create leaves $\ell_1, \ldots, \ell_k$ for all values of $X_a \in \{v_1, \ldots, v_k\}$, with initialized sufficient statistics

i.e., with parameters grace period $n_{\min}$, allowable error in a split decision δ: 1 - desired probability of choosing correct attribute.

Multi-label Incremental Decision Trees

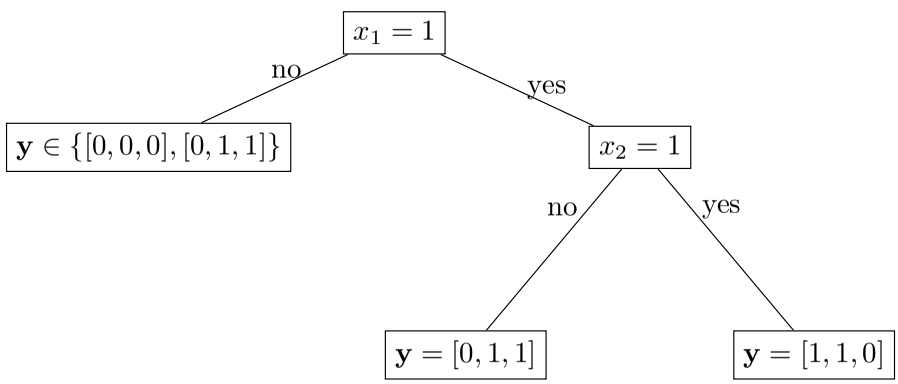
- Use Hoeffding tree algorithm with multi-label entropy
- Examples with *multiple labels* collect at the leaves (we can take the majority labelset, or merge . . .)



i.e., if $\{(\mathbf{x}_\ell, [1,1,0]), (\mathbf{x}_\ell, [1,0,0]), (\mathbf{x}_\ell, [0,1,0])\}$ have been seen at leaf ℓ, we could return score $[0.67, 0.67, 0.0]$ (we already have these statistics).

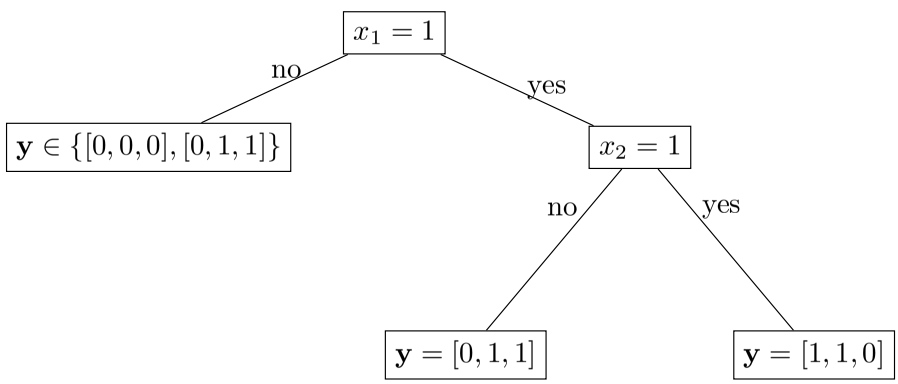
Multi-label Incremental Decision Trees

- A multi-label *incremental* model
- Very *fast* (*single* model for all labels), and usually very *competitive*,



Multi-label Incremental Decision Trees

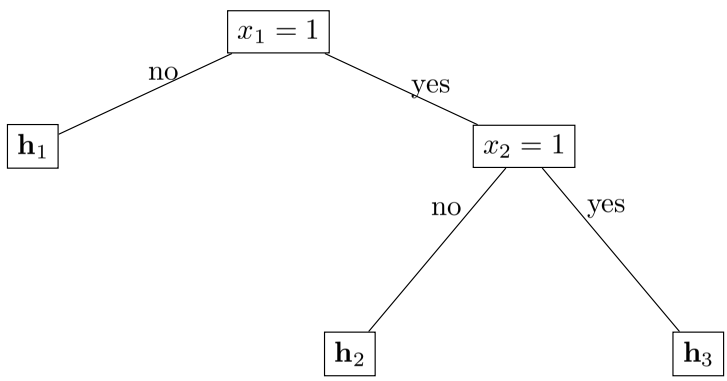
- A multi-label *incremental* model
- Very *fast* (*single* model for all labels), and usually very *competitive*,



- But Hoeffding bound is conservative, tree reluctant to grow, after many examples can remain as root node / stump / majority class classifier, ...
- And when it does grow, it may soon become outdated by concept drift

Classifiers at the Leaves

- Place multi-label classifiers at the leaves of the tree

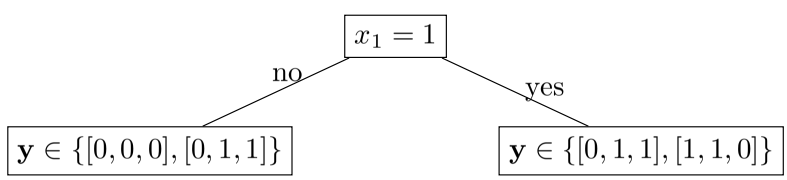


where each $\mathbf{h}_\ell : \mathcal{X}^{D^\ell} \to \mathcal{Y}^{L^\ell}$, i.e., a multi-label classifier dealing with a subset of the input space and a subset of the label space at each leaf ℓ (note: $D^\ell \leq D$, $L^\ell \leq L$).

Hoeffding Adaptive Trees

If the concept changes, do we have to start from scratch? We can use Hoeffding Adaptive Trees (HATs):

- Choose a change-point estimator, e.g., ADWIN, and deploy it at each node
- It will give an alarm when a change at that node is detected
- Chop off the branch from that node, (but we can keep the rest of the tree alive)

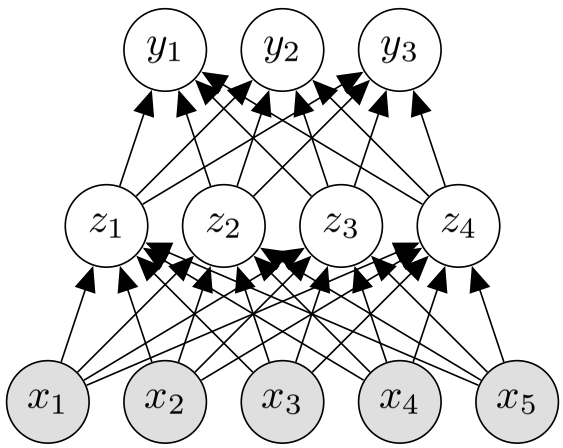


(e.g., after adaptation)

Outline

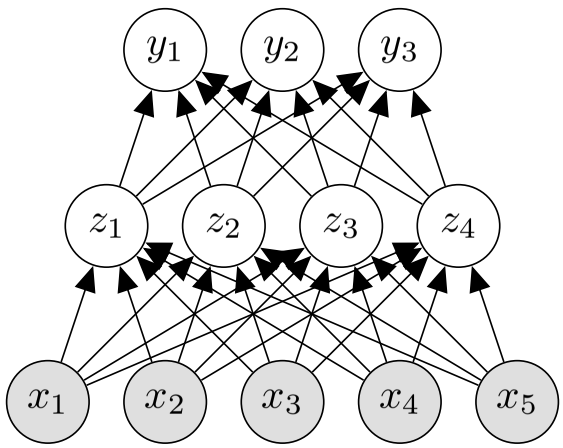
1. Multi-labelled Streaming Data
2. kNN for Streams
3. Batch-incremental Ensemble
4. Incremental base learner
5. Hoeffding Trees
6. Neural Networks
7. Concept Drift
8. Temporal Dependence
9. Connections to Sequential Data
10. Unlabelled Instances
11. Summary

Multiple-layer Perceptron



- Direct application, using stochastic gradient descent (SGD) (incremental) and varieties, with back-propagation.
- Unlike classifier chains (etc.) we do not need a structure on the output layer – if the inner layer conditionally removes the dependence on the input

Multiple-layer Perceptron

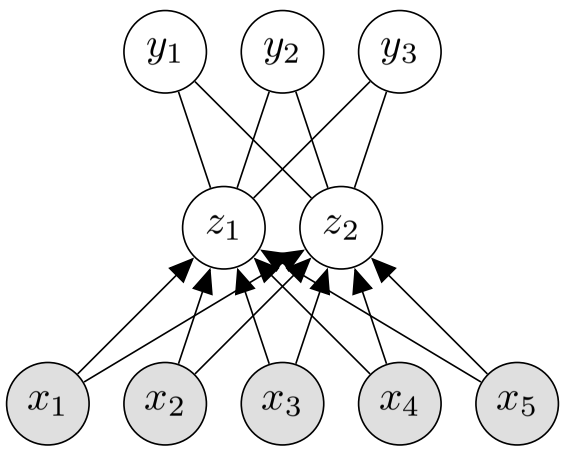


- Direct application, using stochastic gradient descent (SGD) (incremental) and varieties, with back-propagation.
- Unlike classifier chains (etc.) we do not need a structure on the output layer – if the inner layer conditionally removes the dependence on the input
- but empirical data-stream comparisons show that SGD/MLPs achieve poor accuracy against HTs: they are even more difficult to parameterize for/on a stream!

An Independent Label Space

General approach, decoding inner-layer predictions:

$$\mathbf{y} = f^{-1}(\mathbf{z})$$
$$\mathbf{z} = f(\mathbf{y})$$



- Apply binary relevance, on fewer, independent labels!
- Many methods exist (basis functions, PCA, CCA, factor analysis, deep learning, MLPs, cluster analysis, ...)

No need for back-propagation ...

Removing Dependence

We can just run PCA on the label space, make labels independent!

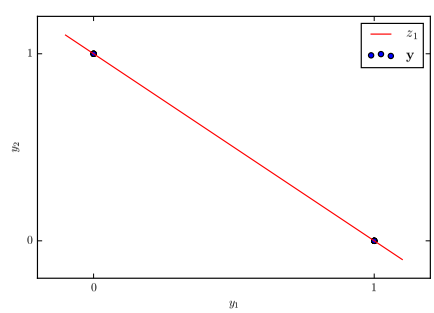
$$\mathbf{Z} = \mathbf{W}\mathbf{Y} \quad \triangleright \ \mathbf{W} \text{ has top } k \text{ components, } k < L$$

$$\mathbf{h} : \mathcal{X} \to \mathcal{Z} \quad \triangleright \text{ model, trained on examples } \{\mathbf{x}_i, \mathbf{z}_i\}_{i=1}^{N}$$

and then, with input $\mathbf{x}$,

$$\mathbf{z} = \mathbf{h}(\mathbf{x}) \quad \triangleright \text{ prediction}$$

$$\hat{\mathbf{y}} = \mathbf{W}^{-1}\mathbf{z} \quad \triangleright \text{ transform}$$

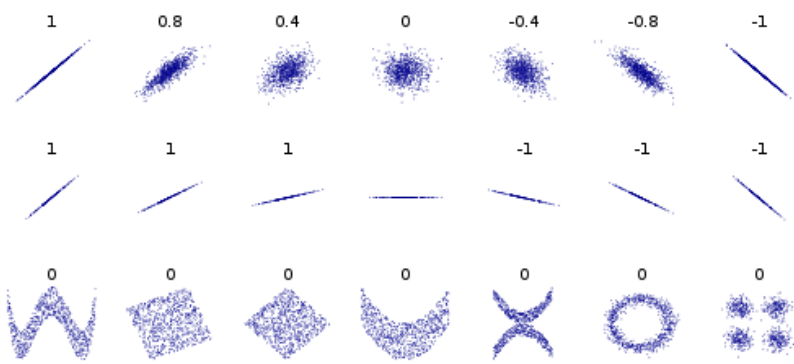


PCA on the Label Space

- Removes correlation, relatively scalable,
- compresses label space (faster learning for BR, etc.)

but . . .

- only linear correlation
- doesn't take into account input space

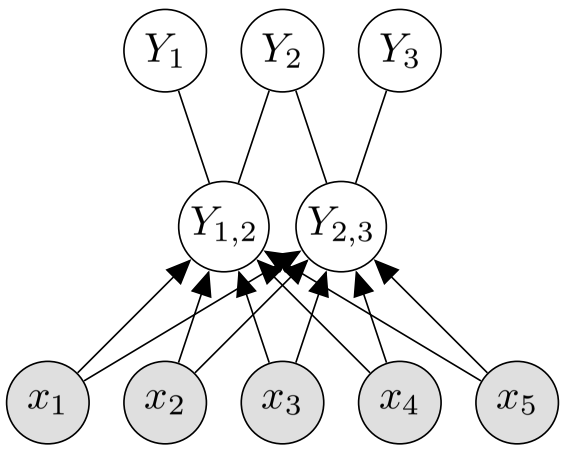


source: Wikipedia

- can use kernels (kernel PCA . . .),
- can use Canonical Correlation Analysis (CCA)

Revisiting Meta Labels

Meta labels form an inner layer, e.g., with $K = 2$ meta labels:



where, e.g., $Y_{1,2} \in \{[0,0],[0,1],[1,1]\}$ and $Y_{2,3} \in \{[1,0],[0,1]\}$.

- We can achieve $K < L$, and $K < D$,
- without any iterative learning!

Voting Using Meta Labels

Table: An example, for a meta-label on $S_1 = \{1, 2\}$, where $\mathbf{Z}_1 = \{[0, 0], [0, 1], [1, 1]\}$. Posterior distribution $\mathbf{P}_1|\mathbf{x}$ used to obtain $\mathbf{P}'_{S_1} = \mathbf{Z}_1 * \mathbf{P}_1$ (bottom row, left). Using indices in S_1, we add these values to the final prediction, with the result from other meta-labels ($S_2 = \{2, 3\}$, in this example), to obtain final posterior for labels (bottom row, right).

S_1:	1	2	$\mathbf{P}_1$
$[\mathbf{Z}_1]_{:,1}$	0	0	0.0
$[\mathbf{Z}_1]_{:,2}$	0	1	0.9
$[\mathbf{Z}_1]_{:,3}$	1	1	0.1
$\mathbf{P}'_{S_1}$	0.1	1.0	

j:	1	2	3
$\mathbf{P}'_{S_1}$	0.1	1.0	
$\mathbf{P}'_{S_2}$		0.7	0.3
$P(y_j = 1\|\mathbf{x})$	0.1	0.85	0.3

Outline

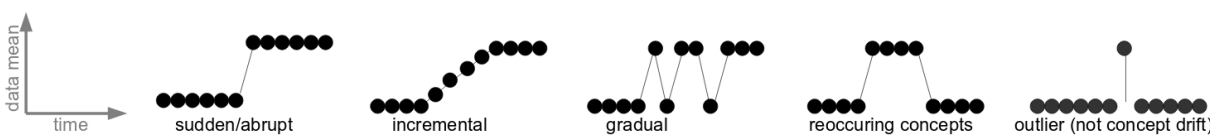
1. Multi-labelled Streaming Data
2. kNN for Streams
3. Batch-incremental Ensemble
4. Incremental base learner
5. Hoeffding Trees
6. Neural Networks
7. Concept Drift
8. Temporal Dependence
9. Connections to Sequential Data
10. Unlabelled Instances
11. Summary

Concept Drift: Types

Types of drift:

1. Sudden/abrupt
2. Incremental
3. Gradual

additionally noting the possibility of reoccurring drift which may involve any of these types and, noting also the related task of dealing with outliers, which is *not* concept drift.



[1]

Sensors

For example, a sensor may be replaced (sudden); or wear out slowly (incremental), or work only sometimes and increasingly infrequently (gradual).

[1] from Gama, J. et al. *A Survey on Concept Drift Adaptation*, 2013

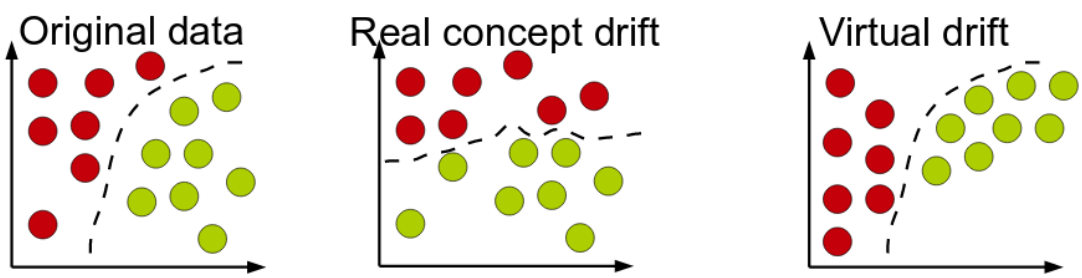
Concept Drift: Real vs Virtual

Data is drawn from some (unknown) distribution

$$(x_t, y_t) \sim P_t(X, Y) = P_t(Y|X)P_t(X)$$

and either component may drift:

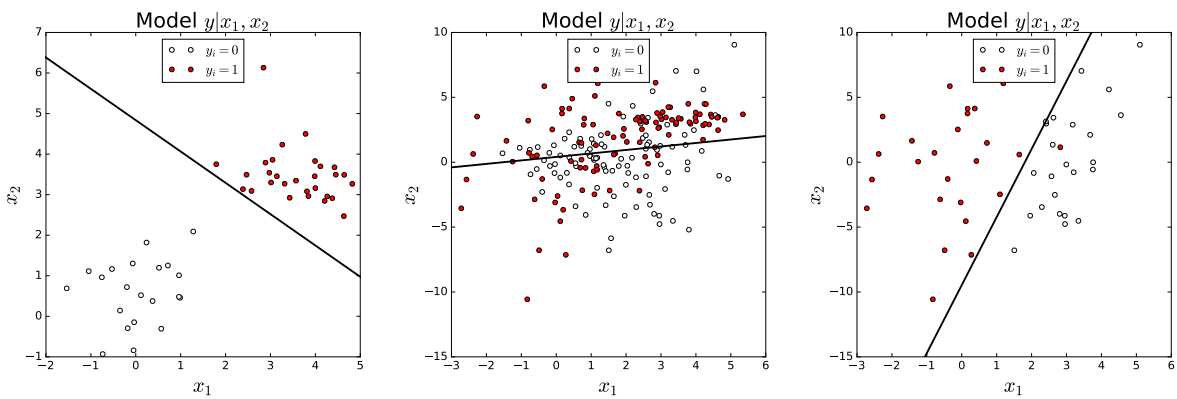
- Real concept drift: changes to $P_t(Y|X)$.
- Virtual drift changes to $P_t(X)$ only.



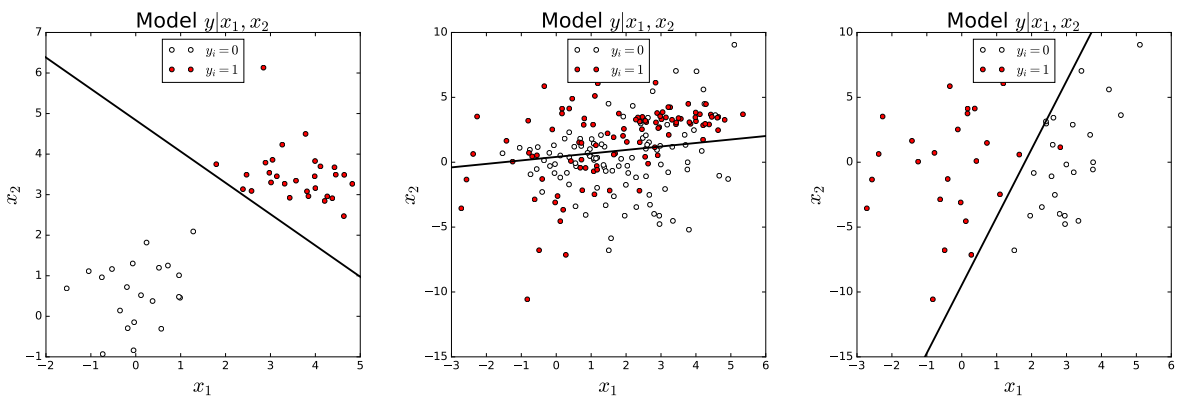
Spam Classification

- Near Christmas we receive many more `Spam` e-mails, but our definition spam has not changed – virtual drift.
- We were interested in the latest flight offers, yet now we decide they are unwanted, and mark them as `Spam` – real drift.

Before (left), during (centre) and after (right) concept drift [linear decision boundary]:



Before (left), during (centre) and after (right) concept drift [linear decision boundary]:



- Model becomes invalid as the concept drifts
- Multi-label concept drift involves also the *label variables*.

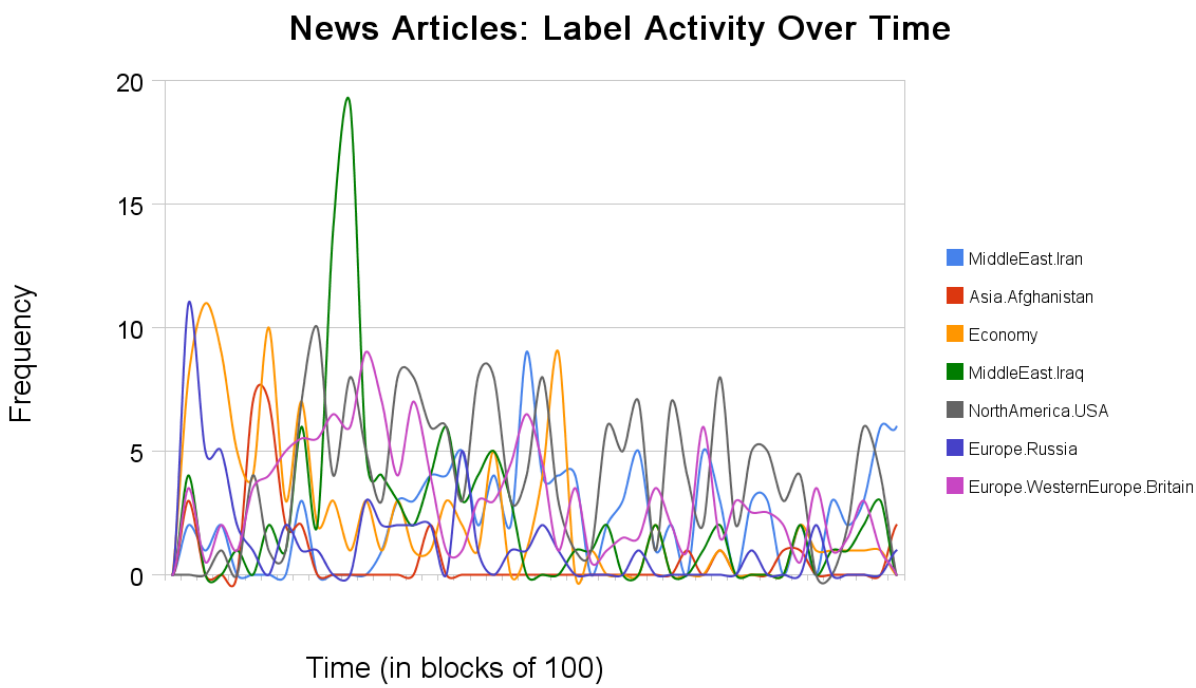


Figure: Multi-label virtual drift: Label frequency / month over time (until about 2007)

Dealing with Concept Drift

Possible approaches to *detecting* and *responding to* concept drift:

1. Just ignore it – batch models must be replaced anyway (we can place higher weight on the most recent models), kNN will eventually have to purge old instances, and SGD adapts constantly (given a learning rate above 0).
2. Monitor a statistic (e.g., predictive performance, accuracy – or a distribution) with a change detector, and reset/recalibrate models when change is detected, e.g., HATs and many ensemble methods.

Dealing with Concept Drift

Possible approaches to *detecting* and *responding to* concept drift:

1. Just ignore it – batch models must be replaced anyway (we can place higher weight on the most recent models), kNN will eventually have to purge old instances, and SGD adapts constantly (given a learning rate above 0).
2. Monitor a statistic (e.g., predictive performance, accuracy – or a distribution) with a change detector, and reset/recalibrate models when change is detected, e.g., HATs and many ensemble methods.

In the multi-label case: it is the same, but

- More classifiers are involved
- Models are more complex, e.g., we need $\mathbf{P}_{\mathbf{x}}(Y_1, \ldots, Y_L)$ rather than just $\mathbf{P}_{\mathbf{x}}(Y)$.

Detection via Monitoring Accuracy

Drift can be detected via error/accuracy:

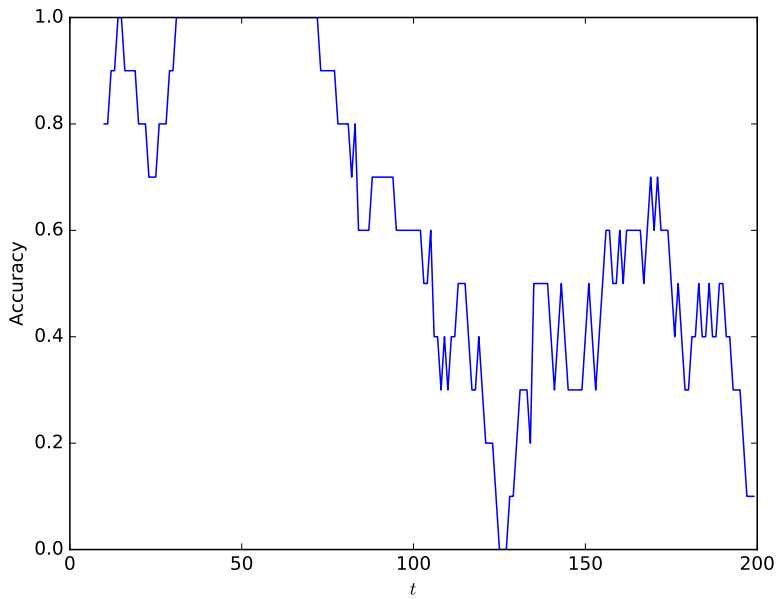
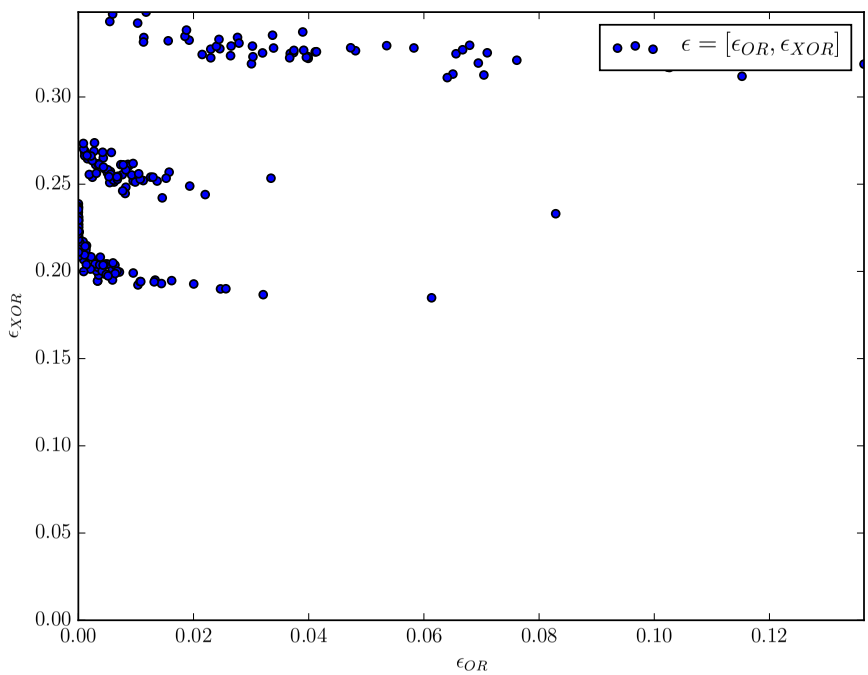


Figure: Accuracy through concept drift ($t = 50, \ldots, 150$).

Detection via Monitoring Distribution

In the multi-label case, we have a more complex distribution:



i.e., density $p(\epsilon)$ – shape may change over time, and structures may need to be adjusted (the best structure may change)

Multi-label Concept Drift

Consider the relative frequencies of labels Y_1 and Y_2 at time t,

$$\mathbf{C}_t = \frac{1}{t}\mathbf{Y}^\top\mathbf{Y} = \begin{bmatrix} \tilde{p}_1 & \tilde{p}_{1,2} \\ \tilde{p}_{2,1} & \tilde{p}_2 \end{bmatrix}$$

where $\tilde{p}_{1,2} > \tilde{p}_1\tilde{p}_2$ indicates marginal dependence!

Possible drift (where $\mathbf{C}_t \neq \mathbf{C}_{t+1}$):

- p_1 increases (label Y_1 relatively more frequent)
- p_1 and p_2 both decrease (label cardinality decreasing)
- $p_{1,2}$ changes relative to $p_1 p_2$ (change in marginal dependence relation between the labels)

Multi-label Concept Drift

And when conditioned on input $\mathbf{x}$, we consider the relative frequencies/values of the errors, where, e.g., $E_{ij} = (y_j^{(i)} - \hat{y}_j^{(i)})^2$:

$$\mathbf{C}_t = \frac{1}{t}\mathbf{E}^\top\mathbf{E} = \begin{bmatrix} \tilde{p}_1 & \tilde{p}_{1,2} \\ \tilde{p}_{2,1} & \tilde{p}_2 \end{bmatrix}$$

(if conditional independence, then $\tilde{p}_{1,2} \approx \tilde{p}_1 \cdot \tilde{p}_2$).

Possible drift (where $\mathbf{C}_t \neq \mathbf{C}_{t+1}$):

- p_1 increases (more errors on 1-th label)
- p_1 and p_2 both increase (more errors)
- $p_{1,2}$ changes relative to p_1, p_2 (change in conditional dependence relation)

Outline

1. Multi-labelled Streaming Data
2. kNN for Streams
3. Batch-incremental Ensemble
4. Incremental base learner
5. Hoeffding Trees
6. Neural Networks
7. Concept Drift
8. Temporal Dependence
9. Connections to Sequential Data
10. Unlabelled Instances
11. Summary

IID Data Streams

In a stream

$$y_t = h(\mathbf{x}_t) + \epsilon_t$$

it is often assumed that instances arrive in i.i.d. form:

- Identically distributed,

$$\mathbf{x}_t \sim p(\mathbf{x})$$

  for the same p always across all $t = 1, \ldots$ within the same concept.

- Independently distributed,

$$p(\mathbf{x}_t) = p(\mathbf{x}|\mathbf{x}_{t-1})$$

Is this a valid assumption for data streams?

Measuring Temporal Dependence

We can measure dependence with (for example) the auto-correlation function (Pearson's correlation coefficient of a variable with itself, lagged $t+1$)[2],

$$\rho_{Y_t, Y_{t+1}} = \frac{\mathrm{Cov}(Y_t, Y_{t+1})}{\mathrm{Std}(Y_t)\mathrm{Std}(Y_{t+1})} \tag{1}$$

$$= \frac{\sum_{t=1}^{T-1}[(y_t - \bar{y})(y_{t+1} - \bar{y})]}{\sqrt{\sum_{t=1}^{T-1}(y_t - \bar{y})^2 \sum_{t=2}^{T}(y_t - \bar{y})^2}} \tag{2}$$

$$\approx \frac{\sum_{t=1}^{T-1}[(y_t - \bar{y})(y_{t+1} - \bar{y})]}{\sum_{t=2}^{T}(y_t - \bar{y})^2} \tag{3}$$

[2] NB: for large T, the difference in the mean of $Y_1, \ldots, Y_{T-1}$ and of $Y_2, \ldots, Y_T$ can be ignored, hence Eq. (3).

Measuring Temporal Dependence

We can measure dependence with (for example) the auto-correlation function (Pearson's correlation coefficient of a variable with itself, lagged $t+1$)[2],

$$\rho_{Y_t, Y_{t+1}} = \frac{\mathrm{Cov}(Y_t, Y_{t+1})}{\mathrm{Std}(Y_t)\mathrm{Std}(Y_{t+1})} \tag{1}$$

$$= \frac{\sum_{t=1}^{T-1}[(y_t - \bar{y})(y_{t+1} - \bar{y})]}{\sqrt{\sum_{t=1}^{T-1}(y_t - \bar{y})^2 \sum_{t=2}^{T}(y_t - \bar{y})^2}} \tag{2}$$

$$\approx \frac{\sum_{t=1}^{T-1}[(y_t - \bar{y})(y_{t+1} - \bar{y})]}{\sum_{t=2}^{T}(y_t - \bar{y})^2} \tag{3}$$

We can generalise to $\rho(k)$ to consider the correlation from y_t and y_{t+k} for any lag k (may even be negative).

[2] NB: for large T, the difference in the mean of $Y_1, \ldots, Y_{T-1}$ and of $Y_2, \ldots, Y_T$ can be ignored, hence Eq. (3).

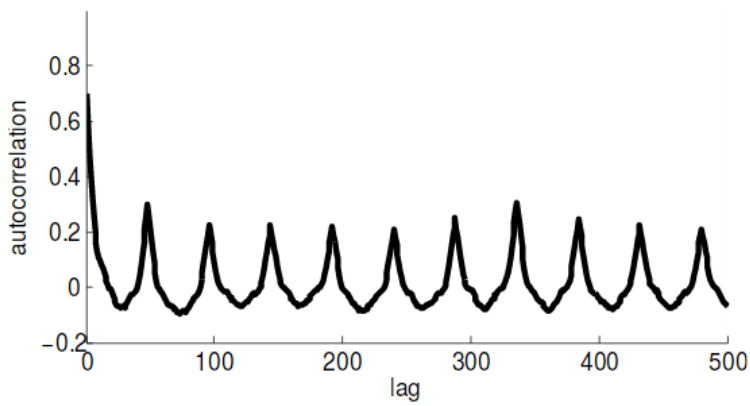


Figure: Auto-correlation function on the Electricity dataset, for $k = 1, 2, \ldots, 500$; source: Indrė Žliobaitė arXiv:1301.3524v1, Jan 2015.

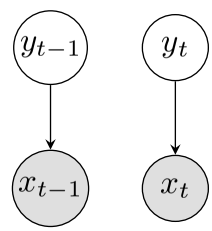
Outline

1. Multi-labelled Streaming Data
2. kNN for Streams
3. Batch-incremental Ensemble
4. Incremental base learner
5. Hoeffding Trees
6. Neural Networks
7. Concept Drift
8. Temporal Dependence
9. Connections to Sequential Data
10. Unlabelled Instances
11. Summary

Naive Bayes

At time t, we see instance x_t, and we wish to make a classification, (e.g., Naive Bayes)

$$\begin{aligned}\hat{y}_t &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_t, x_t) \\ &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(x_t | y_t) P(y_t)\end{aligned}$$

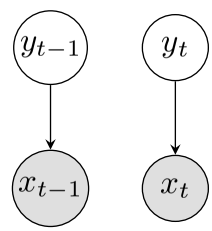


- Not a problem, we maintain an empirical estimate of Ps via counting
- At time $t+1$ we get y_t; we can now update counts with (x_t, y_t)
- We measure error $\epsilon_t = E(y_t - \hat{y}_t)$, look for drift (e.g., ADWIN), etc.

Naive Bayes

At time t, we see instance x_t, and we wish to make a classification, (e.g., Naive Bayes)

$$\begin{aligned}\hat{y}_t &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_t, x_t) \\ &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(x_t | y_t) P(y_t)\end{aligned}$$

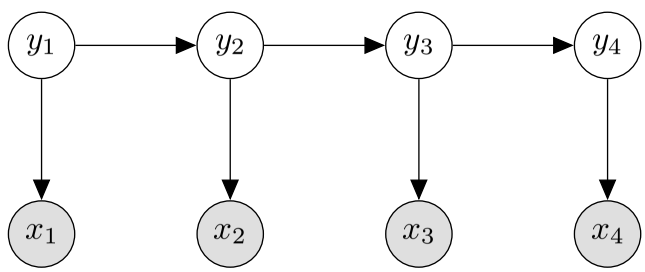


- Not a problem, we maintain an empirical estimate of Ps via counting
- At time $t+1$ we get y_t; we can now update counts with (x_t, y_t)
- We measure error $\epsilon_t = E(y_t - \hat{y}_t)$, look for drift (e.g., ADWIN), etc.

But what if Y_t depends on Y_{t-1} (i.e., temporal dependence)?

Hidden Markov Model

If there is temporal dependence, we cannot 'stop' at $P(y_t, x_t)$, because observations $\ldots, x_{t-1}, x_t$ are connected via ys:



We write out the joint distribution as[3]

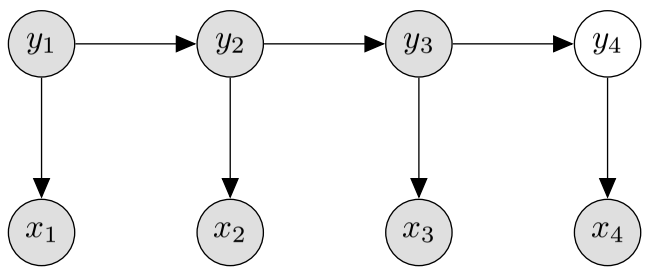
$$\begin{aligned}\hat{y}_t &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_t, \mathbf{x}_t) \\ &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_1) \prod_{\tau=2}^{t} P(x_\tau | y_\tau) P(y_\tau | y_{\tau-1})\end{aligned}$$

Problem, goes back to y_1! Can be a long stream!

[3] Let $\mathbf{x}_t = [x_1, \ldots, x_t]$

Data Stream Classifier with Temporal Dependence

Although with standard data-stream assumptions, we are typically dealing with the filtering problem:

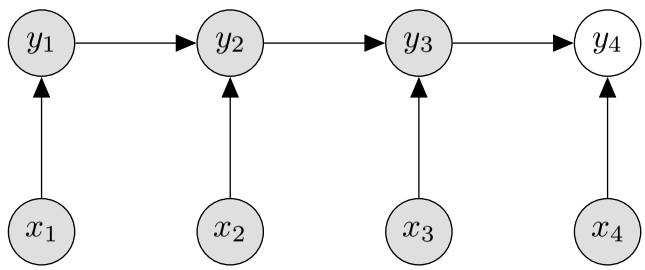


$$\begin{aligned}\hat{y}_t &= \underset{y_t \in \{0,1\}}{\operatorname{argmax}} P(y_t, \mathbf{x}_t, \mathbf{y}_{t-1}) \\ &= \underset{y_t \in \{0,1\}}{\operatorname{argmax}} P(x_t | y_t) P(y_t | y_{t-1})\end{aligned}$$

The recursion stops at $t - 1$.

Maximum Entropy Markov Model (MEMM)

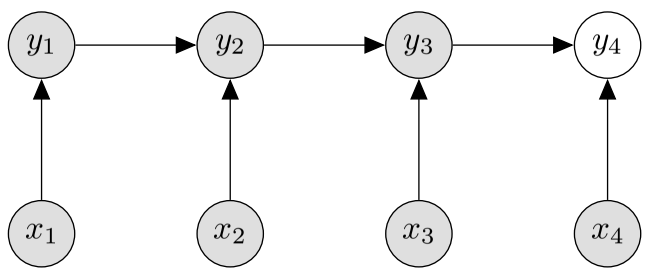
A discriminative approach:



$$\begin{aligned}\hat{y}_t &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_t | \mathbf{x}_t) \\ &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_t | \mathbf{x}_t) P(y_t | y_{t-1}) \\ &= h(x_t, y_{t-1}) \quad \triangleright \text{ Classifier}\end{aligned}$$

Maximum Entropy Markov Model (MEMM)

A discriminative approach:

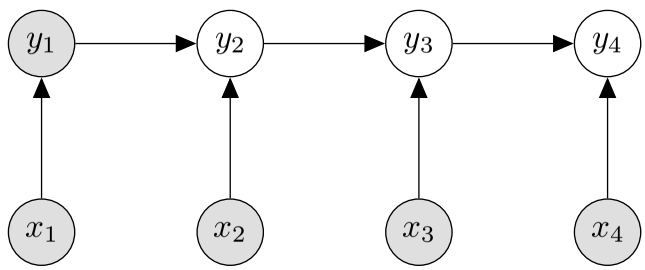


$$\begin{aligned}\hat{y}_t &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_t | \mathbf{x}_t) \\ &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_t | \mathbf{x}_t) P(y_t | y_{t-1}) \\ &= h(x_t, y_{t-1}) \quad \triangleright \text{ Classifier}\end{aligned}$$

But what if, say $t = 4$, but we don't have y_{t-2}, y_{t-3} yet?

Forecasting with MEMMs

...



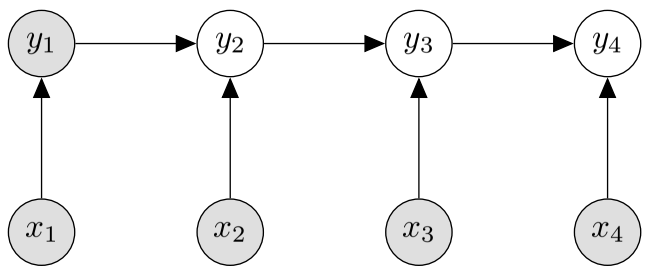
To obtain prediction $\hat{y}_t$, we can use a prediction for $\hat{y}_{t-1}$:

$$\begin{aligned}\hat{y}_t &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_t | \mathbf{x}_t) \\ &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_t | x_t) P(y_t | \hat{y}_{t-1}) \\ &= h(x_t, \hat{y}_{t-1}) \quad \triangleright \text{ Classifier}\end{aligned}$$

And so on, back until our last *observed* y, then we propagate forward.

Forecasting with MEMMs

...



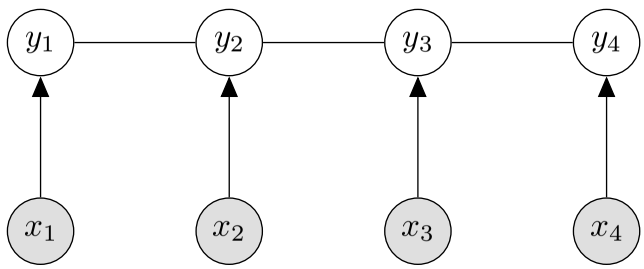
To obtain prediction $\hat{y}_t$, we can use a prediction for $\hat{y}_{t-1}$:

$$\begin{aligned}\hat{y}_t &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_t | \mathbf{x}_t) \\ &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_t | x_t) P(y_t | \hat{y}_{t-1}) \\ &= h(x_t, \hat{y}_{t-1}) \quad \triangleright \text{ Classifier}\end{aligned}$$

And so on, back until our last *observed* y, then we propagate forward. But errors may propagate down the *chain*.

(Linear Chain) Conditional Random Fields (CRFs)

To avoid the error propagation problem, we may use a CRF:

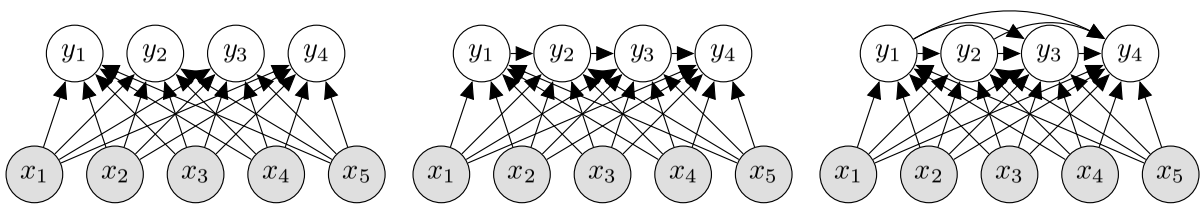


$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \{0,1\}^T}{\operatorname{argmax}} P(\mathbf{y}|\mathbf{x})$$

$$= \underset{\mathbf{y} \in \{0,1\}^T}{\operatorname{argmax}} \prod_{t=2}^{T} f_t(y_{t-1}, y_t, \mathbf{x})$$

What is f_t?

Time Indices $\equiv$ Label Indices



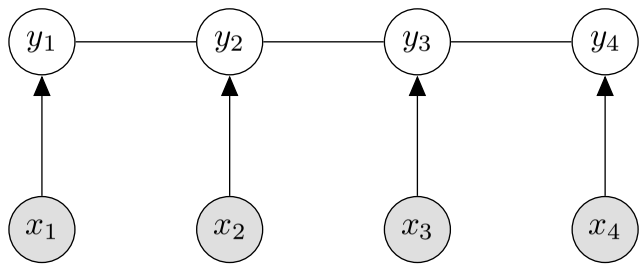
- Labels indices can correspond to steps in time (or space)
- Existing multi-label methodologies can be applied:

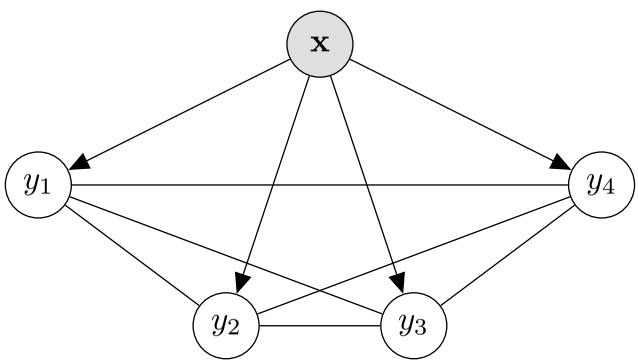
$$\hat{\mathbf{y}} = h(\mathbf{x})$$

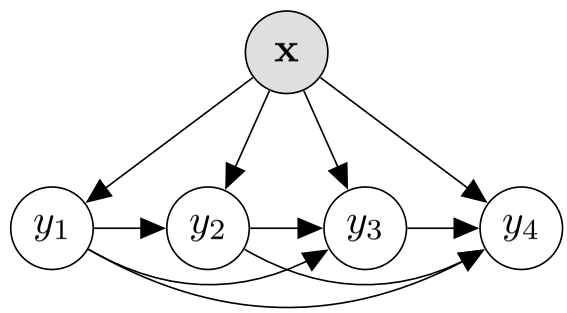
e.g., binary relevance classifiers, meta labels, classifier chains.

From CRF to Probabilistic Classifier Chains (PCC)

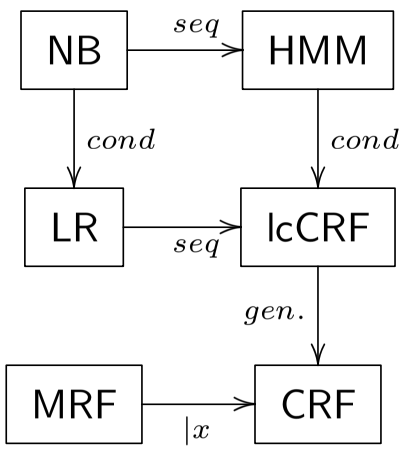
$$
\begin{aligned}
\hat{\mathbf{y}}_t &= \operatorname*{argmax}_{\mathbf{y}\in\{0,1\}^L} p(\mathbf{y}|\mathbf{x};\mathbf{w}) && \triangleright\ \mathbf{w} \text{ is a set of weights} \\
&= \operatorname*{argmax}_{\mathbf{y}\in\{0,1\}^L} \exp\left\{\sum_k w_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\} && \triangleright\ \text{CRF inference} \\
&= \operatorname*{argmax}_{\mathbf{y}\in\{0,1\}^L} \prod_{t=1}^{T} \exp\left\{w_t \cdot f_t(y_t, y_{t-1}, \mathbf{x}_t)\right\} && \triangleright\ e^{a+b} = e^a e^b \\
&= \operatorname*{argmax}_{\mathbf{y}\in\{0,1\}^L} \prod_{t=1}^{T} f_t(y_t, y_{t-1}, \mathbf{x}_t; \mathbf{w}_t) && \triangleright\ \text{a generic fn } f \\
&= \operatorname*{argmax}_{\mathbf{y}\in\{0,1\}^L} f_1(y_1, \mathbf{x}) \prod_{j=2}^{L} f_j(y_1, \ldots, y_{j-1}, \mathbf{x}) && \triangleright\ \text{PCC!} \\
&= \operatorname*{argmax}_{\mathbf{y}\in\{0,1\}^L} P(y_1|\mathbf{x}) \prod_{j=2}^{L} P(y_j|y_1, \ldots, y_{j-1}, \mathbf{x}) && \triangleright\ \text{where } f_j \approx P_j
\end{aligned}
$$







PCC is a flexible CRF (wrt loss function, base classifier, inference method . . .).

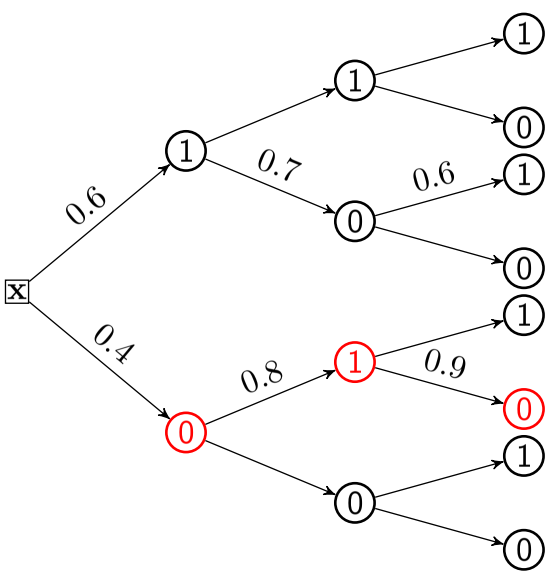


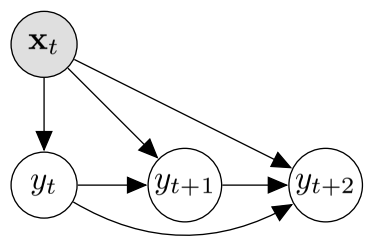
Distribution	Type
$p(y, \mathbf{x})$	NB
$p(\mathbf{y}, \mathbf{x})$	HMM
$p(y \mid \mathbf{x})$	LR
$p(\mathbf{y} \mid \mathbf{x})$	CRF
$p(\mathbf{y})$	MRF

and CRFs can be derived as a kind of classifier chain.

Forecasting with Classifier Chains

We use classifier-chains type inference for forecasting!





Generate samples $\{\mathbf{y}_t\}_{t=1}^{T}$,

$$y_1^{(t)} \sim P(y_1|\mathbf{x})$$
$$y_2^{(t)} \sim P(y_2|\mathbf{x}, y_1^{(t)})$$
$$y_3^{(t)} \sim P(y_3|\mathbf{x}, y_1^{(t)}, y_2^{(t)})$$

i.e., $\mathbf{y}_t = [y_1^{(t)}, y_2^{(t)}, y_3^{(t)}]$, i.e., sampling into the future.

Outline

1. Multi-labelled Streaming Data
2. kNN for Streams
3. Batch-incremental Ensemble
4. Incremental base learner
5. Hoeffding Trees
6. Neural Networks
7. Concept Drift
8. Temporal Dependence
9. Connections to Sequential Data
10. Unlabelled Instances
11. Summary

Unlabelled Instances in the Stream

What if we *never* get true labels for some $\mathbf{x}_t$? In many applications it is unrealistic to expect labels for every instance.

Unlabelled Instances in the Stream

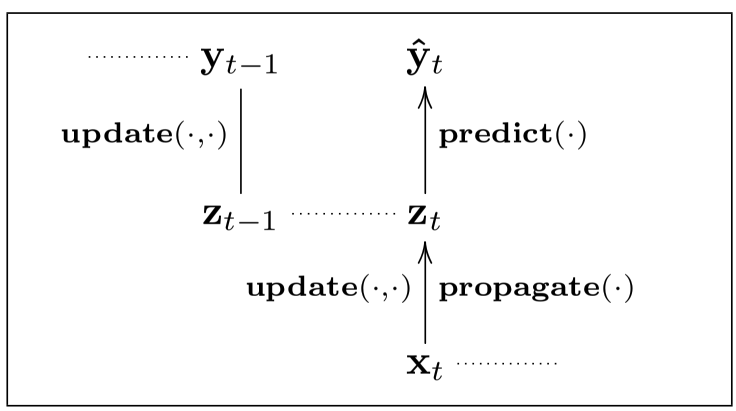
What if we *never* get true labels for some $\mathbf{x}_t$? In many applications it is unrealistic to expect labels for every instance. During learning, we can:

1. Ignore instances with no label
2. Use active learning to get good labels
3. Use predicted labels (self-training)
4. Use an unsupervised process for example clustering, latent-variable representations.

Semi-Supervised Data Streams

With unsupervised model g to provide a new representation of the data:

1. $\mathbf{z}_t = g(\mathbf{x}_t)$ ▷ cluster
2. $\hat{\mathbf{y}}_t = h(\mathbf{z}_t)$ ▷ predict
3. update g with example $(\mathbf{x}_t, \mathbf{z}_t)$
4. update h with example $(\mathbf{z}_{t-1}, \mathbf{y}_{t-1})$ (*if* y_{t-1} is available)

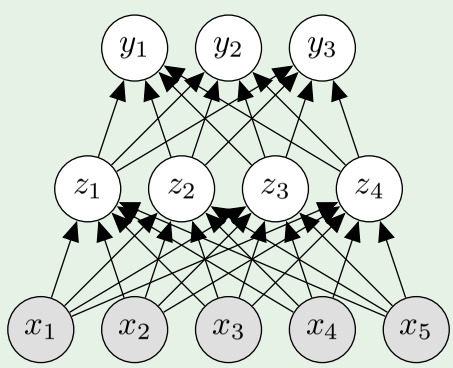


Semi-Supervised Data Streams

With unsupervised model g to provide a new representation of the data:

1. $\mathbf{z}_t = g(\mathbf{x}_t)$ ▷ cluster
2. $\hat{\mathbf{y}}_t = h(\mathbf{z}_t)$ ▷ predict
3. update g with example $(\mathbf{x}_t, \mathbf{z}_t)$
4. update h with example $(\mathbf{z}_{t-1}, \mathbf{y}_{t-1})$ (*if* y_{t-1} is available)

Example



(we can fine tune the inner layer with back propagation).

Outline

1. Multi-labelled Streaming Data
2. kNN for Streams
3. Batch-incremental Ensemble
4. Incremental base learner
5. Hoeffding Trees
6. Neural Networks
7. Concept Drift
8. Temporal Dependence
9. Connections to Sequential Data
10. Unlabelled Instances
11. Summary

Summary

- Multi-label classification can be adapted to the data-stream environment
- This context incurs particular challenges: modelling label dependence is important, but this is difficult in a dynamic environment (concept drift)
- Temporal dependence may exist in data streams: dependence exists across time
- Strong parallels exist between
  - multi-label learning (dependence among labels) and
  - sequence learning (dependence across time)
- We can adapt existing methods for forecasting and making use of unlabelled instances for training.

Multi-label Streams, Concept Drift, and Sequential Data

Jesse Read

[D&K] IoT Stream Data Mining 2017-2018

December 20, 2017

Ensemble Methods

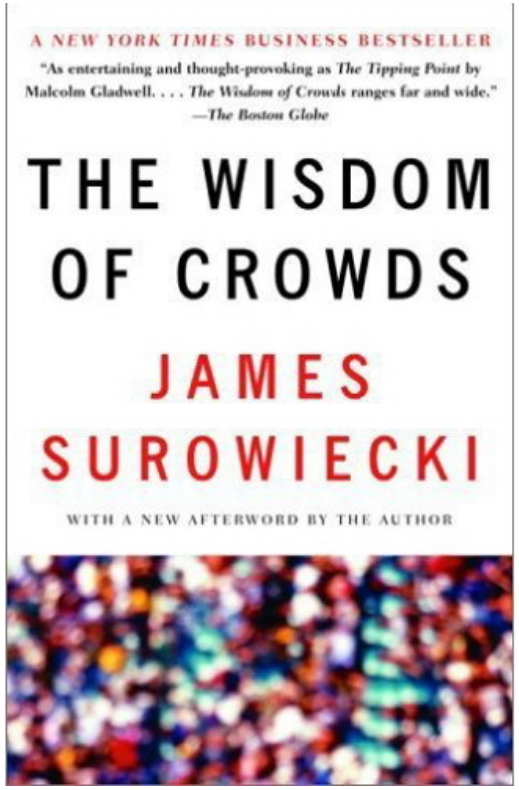
Albert Bifet (@abifet)

Paris, 6 December 2016
albert.bifet@telecom-paristech.fr

Data Streams

Big Data & Real Time

Ensemble Learning: The Wisdom of Crowds



Diversity of opinion, Independence
Decentralization, Aggregation

Bagging

Example

Dataset of 4 Instances : A, B, C, D

Classifier 1: B, A, C, B
Classifier 2: D, B, A, D
Classifier 3: B, A, C, B
Classifier 4: B, C, B, B
Classifier 5: D, C, A, C

Bagging builds a set of M base models, with a bootstrap sample created by drawing random samples with replacement.

Bagging

Example

Dataset of 4 Instances : A, B, C, D

Classifier 1: A, B, B, C
Classifier 2: A, B, D, D
Classifier 3: A, B, B, C
Classifier 4: B, B, B, C
Classifier 5: A, C, C, D

Bagging builds a set of M base models, with a bootstrap sample created by drawing random samples with replacement.

Bagging

Example

Dataset of 4 Instances : A, B, C, D

Classifier 1: A, B, B, C: A(1) B(2) C(1) D(0)
Classifier 2: A, B, D, D: A(1) B(1) C(0) D(2)
Classifier 3: A, B, B, C: A(1) B(2) C(1) D(0)
Classifier 4: B, B, B, C: A(0) B(3) C(1) D(0)
Classifier 5: A, C, C, D: A(1) B(0) C(2) D(1)

Each base model's training set contains each of the original training example K times where $P(K = k)$ follows a binomial distribution.

Bagging

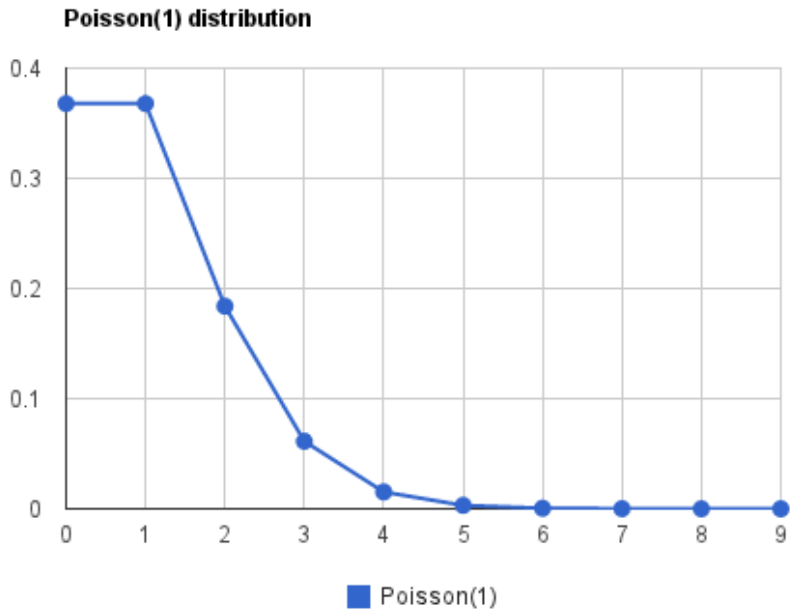


Figure: Poisson(1) Distribution.

Each base model's training set contains each of the original training example K times where $P(K = k)$ follows a binomial distribution.

Oza and Russell's *Online Bagging* for M models

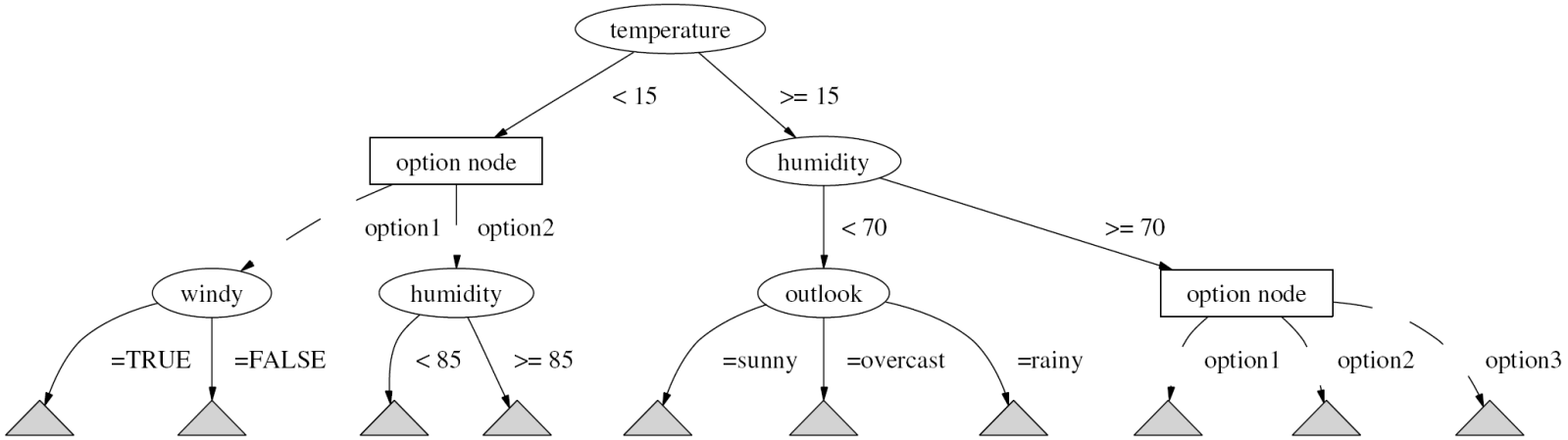
Initialize base models h_m for all $m \in \{1, 2, ..., M\}$
for all training examples **do**
  for $m = 1, 2, ..., M$ **do**
    Set $w = Poisson(1)$
    Update h_m with the current example with weight w

anytime output:
return hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} I(h_t(x) = y)$

Hoeffding Option Tree

Hoeffding Option Trees

Regular Hoeffding tree containing additional option nodes that allow several tests to be applied, leading to multiple Hoeffding trees as separate paths.



Random Forests (Breiman, 2001)

Adding randomization to decision trees

- the input training set is obtained by sampling with replacement, like Bagging
- the nodes of the tree only may use a fixed number of random attributes to split
- the trees are grown without pruning

Accuracy Weighted Ensemble

Mining concept-drifting data streams using ensemble classifiers. Wang et al. 2003

- Process chunks of instances of size W
- Builds a new classifier for each chunk
- Removes old classifier
- Weight each classifier using error

$$w_i = MSE_r - MSE_i$$

where

$$MSE_r = \sum_c p(c)(1 - p(c))^2$$

and

$$MSE_i = \frac{1}{|S_n|} \sum_{(x,c) \in S_n} (1 - f_c^i(x))^2$$

ADWIN Bagging

ADWIN

An adaptive sliding window whose size is recomputed online according to the rate of change observed.

ADWIN has rigorous guarantees (theorems)

- On ratio of false positives and negatives
- On the relation of the size of the current window and change rates

ADWIN Bagging

When a change is detected, the worst classifier is removed and a new classifier is added.

ADWIN *Bagging* for M models

Initialize base models h_m for all $m \in \{1, 2, ..., M\}$
for all training examples **do**
  for $m = 1, 2, ..., M$ **do**
    Set $w = Poisson(1)$
    Update h_m with the current example with weight w
    if ADWIN detects change in error of one of the classifiers **then**
      Replace classifier with higher error with a new one

anytime output:
return hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} I(h_t(x) = y)$

Leveraging Bagging for Evolving Data Streams

Randomization as a powerful tool to increase accuracy and diversity

There are three ways of using randomization:

- Manipulating the input data
- Manipulating the classifier algorithms
- Manipulating the output targets

Input Randomization

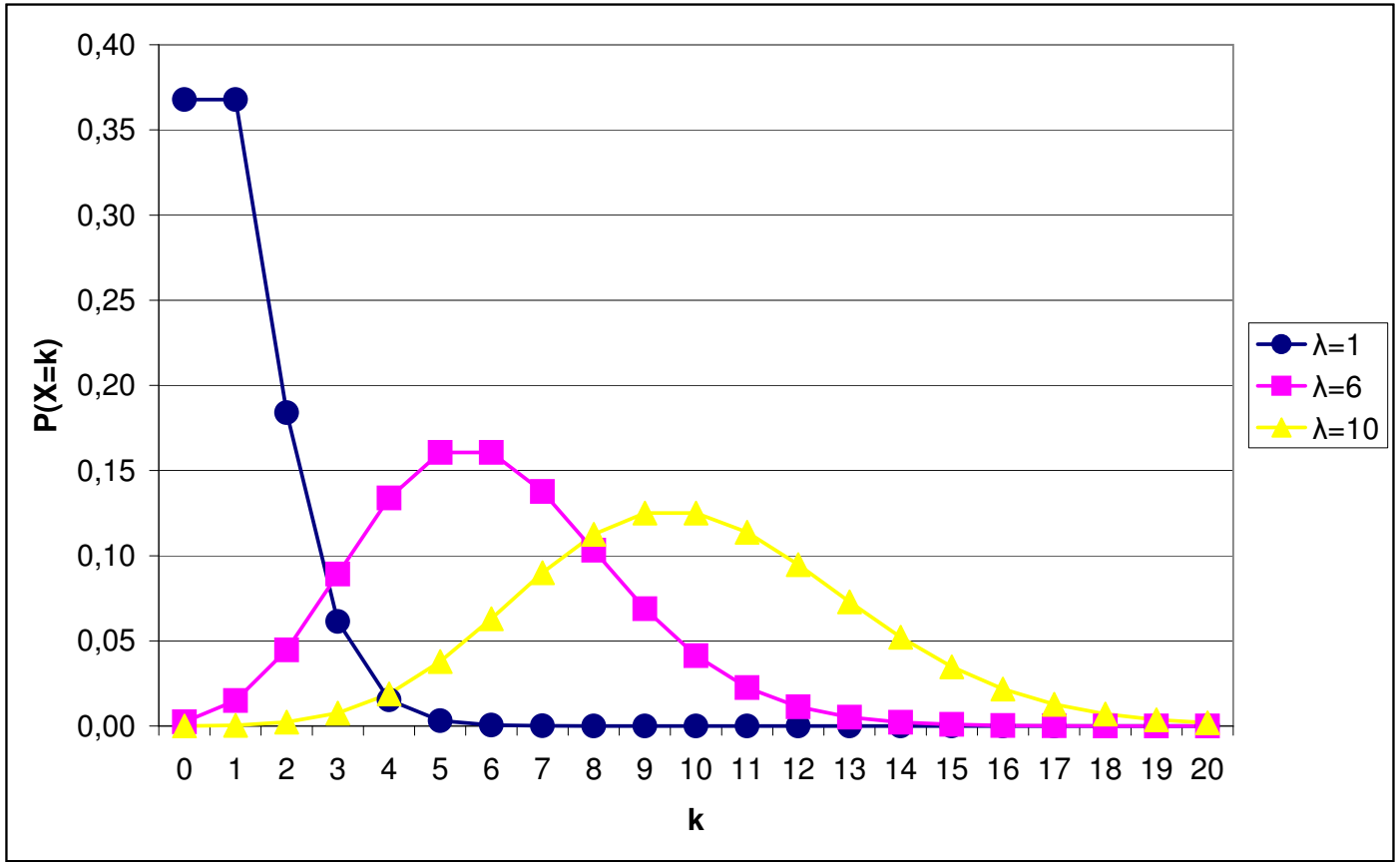


Figure: Poisson Distribution.

ECOC Output Randomization

Table: Example matrix of random output codes for 3 classes and 6 classifiers

	Class 1	Class 2	Class 3
Classifier 1	0	0	1
Classifier 2	0	1	1
Classifier 3	1	0	0
Classifier 4	1	1	0
Classifier 5	1	0	1
Classifier 6	0	1	0

Leveraging Bagging for Evolving Data Streams

Leveraging Bagging

- Using $Poisson(\lambda)$

Leveraging Bagging MC

- Using $Poisson(\lambda)$ and Random Output Codes

Fast Leveraging Bagging ME

- if an instance is misclassified: weight = 1
- if not: weight = $e_T/(1 - e_T)$,

Empirical evaluation

	Accuracy	RAM-Hours
Hoeffding Tree	74.03%	0.01
Online Bagging	77.15%	2.98
ADWIN Bagging	79.24%	1.48
Leveraging Bagging	85.54%	20.17
Leveraging Bagging MC	85.37%	22.04
Leveraging Bagging ME	80.77%	0.87

Leveraging Bagging

- Leveraging Bagging
  - Using $Poisson(\lambda)$
- Leveraging Bagging MC
  - Using $Poisson(\lambda)$ and Random Output Codes
- Leveraging Bagging ME
  - Using weight 1 if misclassified, otherwise $e_T/(1 - e_T)$

Boosting

The strength of Weak Learnability, Schapire 90

A boosting algorithm transforms a weak learner into a strong one

Boosting

A formal description of Boosting (Schapire)

- given a training set $(x_1, y_1), \ldots, (x_m, y_m)$
- $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$
- for $t = 1, \ldots, T$
  - construct distribution D_t
  - find weak classifier

$$h_t : X \implies \{-1, +1\}$$

  with small error $\epsilon_t = \Pr_{D_t}[h_t(x_i) \neq y_i]$ on D_t
- output final classifier

Boosting

Oza and Russell's *Online Boosting*

Initialize base models h_m for all $m \in \{1, 2, ..., M\}$, $\lambda_m^{sc} = 0$, $\lambda_m^{sw} = 0$
for all training examples **do**
    Set "weight" of example $\lambda_d = 1$
    for $m = 1, 2, ..., M$ **do**
        Set $k = Poisson(\lambda_d)$
        for $n = 1, 2, ..., k$ **do**
            Update h_m with the current example
        if h_m correctly classifies the example **then**
            $\lambda_m^{sc} \leftarrow \lambda_m^{sc} + \lambda_d$
            $\epsilon_m = \frac{\lambda_m^{sw}}{\lambda_m^{sw} + \lambda_m^{sc}}$
            $\lambda_d \leftarrow \lambda_d \left(\frac{1}{2(1-\epsilon_m)} \right)$ Decrease λ_d
        else
            $\lambda_m^{sw} \leftarrow \lambda_m^{sw} + \lambda_d$
            $\epsilon_m = \frac{\lambda_m^{sw}}{\lambda_m^{sw} + \lambda_m^{sc}}$
            $\lambda_d \leftarrow \lambda_d \left(\frac{1}{2\epsilon_m} \right)$ Increase λ_d
anytime output:
return hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_{m: h_m(x) = y} - \log \epsilon_m / (1 - \epsilon_m)$

Stacking

Use a classifier to combine predictions of base classifiers

- Example: use a perceptron to do stacking

Restricted Hoeffding Trees

Trees for all possible attribute subsets of size k

- $\binom{m}{k}$ subsets
- $\binom{m}{k} = \frac{m!}{k!(m-k)!} = \binom{m}{m-k}$

Example for 10 attributes

$$\binom{10}{1} = 10 \quad \binom{10}{2} = 45 \quad \binom{10}{3} = 120$$

$$\binom{10}{4} = 210 \quad \binom{10}{5} = 252$$

Clustering

Albert Bifet (@abifet)

Paris, 6 December 2016
albert.bifet@telecom-paristech.fr

Data Streams

Big Data & Real Time

Clustering

Definition

Clustering is the distribution of a set of instances of examples into non-known groups according to some common relations or affinities.

Example

Market segmentation of customers

Example

Social network communities

Clustering

Definition

Given

- a set of instances I
- a number of clusters K
- an objective function $cost(I)$

a clustering algorithm computes an assignment of a cluster for each instance

$$f : I \rightarrow \{1, \ldots, K\}$$

that minimizes the objective function $cost(I)$

Clustering

Definition

Given

- a set of instances I
- a number of clusters K
- an objective function $cost(C, I)$

a clustering algorithm computes a set C of instances with $|C| = K$ that minimizes the objective function

$$cost(C, I) = \sum_{x \in I} d^2(x, C)$$

where

- $d(x, c)$: distance function between x and c
- $d^2(x, C) = min_{c \in C} d^2(x, c)$: distance from x to the nearest point in C

k-means

- 1. Choose k initial centers $C = \{c_1, \ldots, c_k\}$
- 2. while stopping criterion has not been met
  - For $i = 1, \ldots, N$
    - find closest center $c_k \in C$ to each instance p_i
    - assign instance p_i to cluster C_k
  - For $k = 1, \ldots, K$
    - set c_k to be the center of mass of all points in C_i

k-means++

- 1. Choose a initial center c_1
- For $k = 2, \ldots, K$
  - select $c_k = p \in I$ with probability $d^2(p, C)/cost(C, I)$
- 2. while stopping criterion has not been met
  - For $i = 1, \ldots, N$
    - find closest center $c_k \in C$ to each instance p_i
    - assign instance p_i to cluster C_k
  - For $k = 1, \ldots, K$
    - set c_k to be the center of mass of all points in C_i

Performance Measures

Internal Measures

- Sum square distance
- Dunn index $D = \frac{d_{min}}{d_{max}}$
- C-Index $C = \frac{S - S_{min}}{S_{max} - S_{min}}$

External Measures

- Rand Measure
- F Measure
- Jaccard
- Purity

BIRCH

Balanced Iterative Reducing and Clustering using Hierarchies

- Clustering Features $CF = (N, LS, SS)$
  - N: number of data points
  - LS: linear sum of the N data points
  - SS: square sum of the N data points
  - Properties:
    - Additivity: $CF_1 + CF_2 = (N_1 + N_2, LS_1 + LS_2, SS_1 + SS_2)$
    - Easy to compute: average inter-cluster distance and average intra-cluster distance
- Uses CF tree
  - Height-balanced tree with two parameters
    - B: branching factor
    - T: radius leaf threshold

BIRCH

Balanced Iterative Reducing and Clustering using Hierarchies

- **Phase 1:** Scan all data and build an initial in-memory CF tree
- **Phase 2:** Condense into desirable range by building a smaller CF tree (optional)
- **Phase 3:** Global clustering
- **Phase 4:** Cluster refining (optional and off line, as requires more passes)

Clu-Stream

Clu-Stream

- Uses micro-clusters to store statistics on-line
  - Clustering Features $CF = (N, LS, SS, LT, ST)$
    - N: numer of data points
    - LS: linear sum of the N data points
    - SS: square sum of the N data points
    - LT: linear sum of the time stamps
    - ST: square sum of the time stamps
- Uses pyramidal time frame

Clu-Stream

On-line Phase

- For each new point that arrives
  - the point is absorbed by a micro-cluster
  - the point starts a new micro-cluster of its own
    - delete oldest micro-cluster
    - merge two of the oldest micro-cluster

Off-line Phase

- Apply k-means using microclusters as points

Density based methods

DBSCAN

- ϵ-neighborhood(p): set of points that are at a distance of p less or equal to ϵ
- Core object: object whose ϵ-neighborhood has an overall weight at least μ
- A point p is *directly density-reachable* from q if
  - p is in ϵ-neighborhood(q)
  - q is a core object
- A point p is *density-reachable* from q if
  - there is a chain of points $p_1, \ldots, p_n$ such that p_{i+1} is directly density-reachable from p_i
- A point p is *density-connected* from q if
  - there is point o such that p and q are density-reachable from o

Density based methods

DBSCAN

- A *cluster* C of points satisfies
  - if $p \in C$ and q is density-reachable from p, then $q \in C$
  - all points $p, q \in C$ are density-connected
- A *cluster* is uniquely determined by any of its core points
- A *cluster* can be obtained
  - choosing an arbitrary core point as a seed
  - retrieve all points that are density-reachable from the seed

Density based methods

DBSCAN

- select an arbitrary point p
- retrieve all points density-reachable from p
- if p is a core point, a cluster is formed
- If p is a border point
  - no points are density-reachable from p
  - DBSCAN visits the next point of the database
- Continue the process until all of the points have been processed

Density based methods

DenStream

- ϵ-neighborhood(p): set of points that are at a distance of p less or equal to ϵ
- Core object: object whose ϵ-neighborhood has an overall weight at least μ
- Density area: union of the ϵ-neighborhood of core objects

Density based methods

DenStream

For a group of points $p_{i_1}, p_{i_2}, \ldots, p_{i_n}$,
with time stamps $T_{i_1}, T_{i_2}, \ldots, T_{i_n}$

- core-micro-cluster
  - $w = \sum_{j=1}^{n} f(t - T_{i_j})$ where $f(t) = 2^{-\lambda t}$ and $w \geq \mu$
  - $c = \sum_{j=1}^{n} f(t - T_{i_j}) p_{i_j} / w$
  - $r = \sum_{j=1}^{n} f(t - T_{i_j}) dist(p_{i_j}, c) / w$ where $r \leq \epsilon$
- potential core-micro-cluster
  - $w = \sum_{j=1}^{n} f(t - T_{i_j})$ where $f(t) = 2^{-\lambda t}$ and $w \geq \beta\mu$
  - $\overline{CF^1} = \sum_{j=1}^{n} f(t - T_{i_j}) p_{i_j}$
  - $\overline{CF^2} = \sum_{j=1}^{n} f(t - T_{i_j}) p_{i_j}^2$ where $r \leq \epsilon$
- outlier micro-cluster: $w < \beta\mu$

DenStream

On-line Phase

- For each new point that arrives
  - try to merge to a p-micro-cluster
  - else, try to merge to nearest o-micro-cluster
    - if $w > \beta\mu$ then
    - convert the o-micro-cluster to p-micro-cluster
  - otherwise create a new o-microcluster

Off-line Phase

- for each p-micro-cluster c_p
  - if $w < \beta\mu$ then remove c_p
- for each o-micro-cluster c_o
  - if $w < (2^{-\lambda(t-t_o+T_p)} - 1)/(2^{-\lambda T_p} - 1)$ then remove c_o
- Apply DBSCAN using microclusters as points

ClusTree

ClusTree: anytime clustering

- Hierarchical data structure: logarithmic insertion complexity
- Buffer and hitchhiker concept: enable anytime clustering
- Exponential decay
- Aggregation: for very fast streams

StreamKM++: Coresets

Coreset of a set P with respect to some problem

Small subset that approximates the original set P.

- Solving the problem for the coreset provides an approximate solution for the problem on P.

(k, ϵ)-coreset

A (k, ϵ)-coreset S of P is a subset of P that for each C of size k

$$(1 - \epsilon)cost(P, C) \leq cost_w(S, C) \leq (1 + \epsilon)cost(P, C)$$

StreamKM++: Coresets

Coreset Tree

- Choose a leaf l node at random
- Choose a new sample point denoted by q_{t+1} from P_l according to d^2
- Based on q_l and q_{t+1}, split P_l into two subclusters and create two child nodes

StreamKM++

- Maintain $L = \lceil \log_2(\frac{n}{m}) + 2 \rceil$ buckets $B_0, B_1, \ldots, B_{L-1}$