Multi-label Streams, Concept Drift, and Sequential Data

Jesse Read

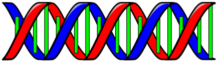
[D&K] IoT Stream Data Mining 2017-2018
December 20, 2017

Outline

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)
- . . .

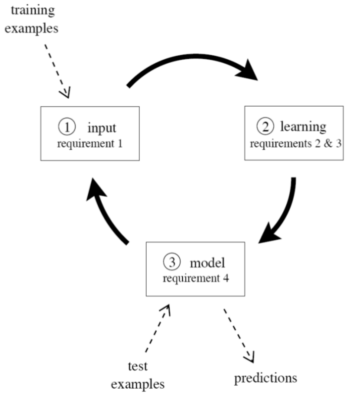
Implications of Data Streams

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:

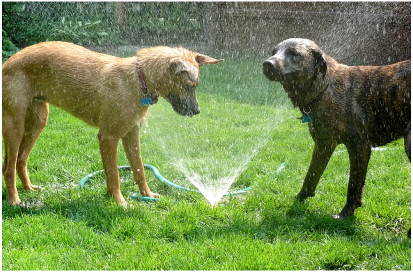
1. Observe $\mathbf{x}$
2. Make a prediction $\hat{y} = h(\mathbf{x})$
3. Observe true label y
4. Measure the error $\epsilon = E(y, \hat{y})$
5. Monitor ϵ signal for concept drift
6. Update the model h with example $(\mathbf{x}, y)$



Implications for Multi-label Classifiers

As compared to the single-labelled case:

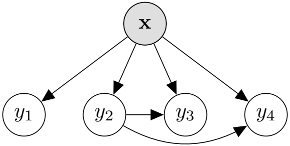
- More computationally complex (label dimension)
- Dynamic label set
- Dynamic label dependence
  (i.e., multi-dimensional concept drift)
- Multi-labelled examples more difficult to obtain
  (L-times more expensive)



Multi-label Learning in Data Streams

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

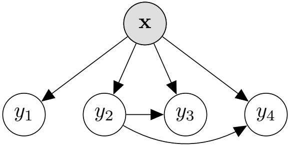
1. Analyze label dependence
2. Search for a good structure
3. Deploy model



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~~



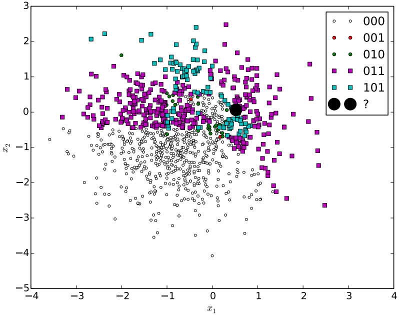
No time! Data still arriving! Dependence (and therefore best structure) may become invalidated over time. Model should have been deployed already!

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

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

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).

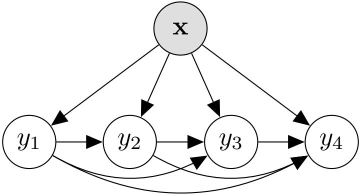
$$\underbrace{\begin{bmatrix}\mathbf{x}\\\mathbf{y}\end{bmatrix}_1, \begin{bmatrix}\mathbf{x}\\\mathbf{y}\end{bmatrix}_2, \begin{bmatrix}\mathbf{x}\\\mathbf{y}\end{bmatrix}_3, \begin{bmatrix}\mathbf{x}\\\mathbf{y}\end{bmatrix}_4}_{\text{build } \mathbf{h}_1}, \underbrace{\begin{bmatrix}\mathbf{x}\\\mathbf{y}\end{bmatrix}_5,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

- 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

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

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,

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

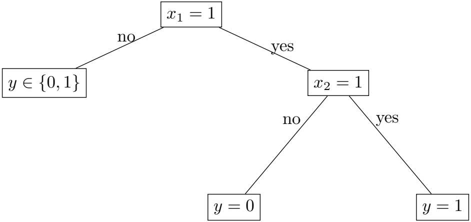
Can expand rules with the Hoeffding bound: for a variable $x \in R$, the Hoeffding bound states that with probability $1 - \delta$, the true mean of X is at least $\bar{x} - \epsilon$, where

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

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):

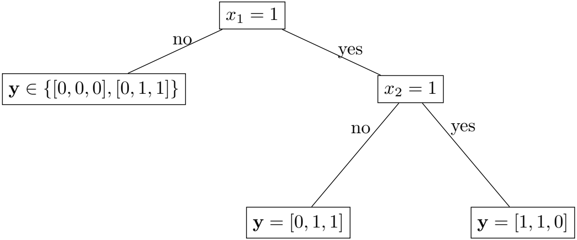
Hoeffding Tree Induction Algorithm (for each instance):

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

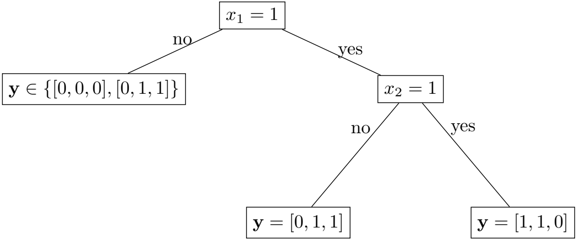
- Use Hoeffding tree algorithm with multi-label entropy
- Examples with *multiple labels* collect at the leaves
(we can take the majority labelset, or merge ...)



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).

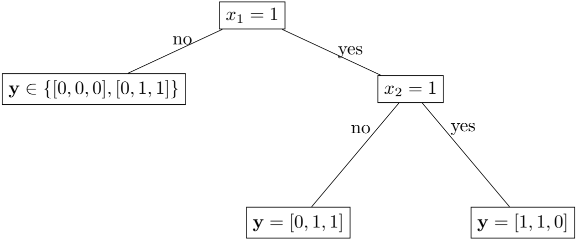
Multi-label Incremental Decision Trees

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



Multi-label Incremental Decision Trees

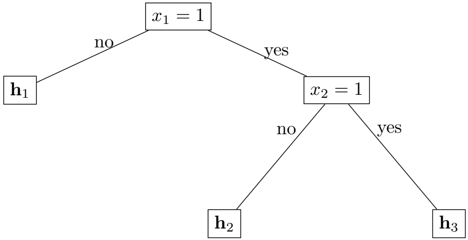
- A multi-label *incremental* model
- Very *fast* (*single* model for all labels), and usually very *competitive*,



- 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

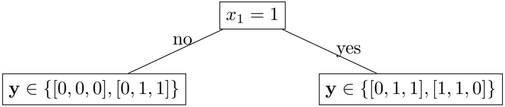


where each $\mathbf{h}_\ell : \mathcal{X}^{D^\ell} \to \mathcal{Y}^{L^\ell}$, i.e., a multi-label classifier dealing with a subset of the input space and a subset of the label space at each leaf ℓ (note: $D^\ell \leq D$, $L^\ell \leq L$).

Hoeffding Adaptive Trees

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

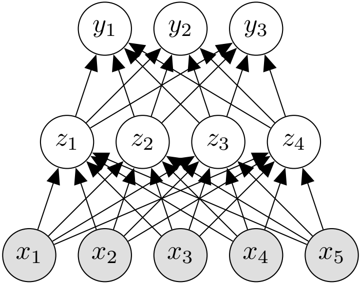
- Choose a change-point estimator, e.g., ADWIN, and deploy it at each node
- It will give an alarm when a change at that node is detected
- Chop off the branch from that node, (but we can keep the rest of the tree alive)



(e.g., after adaptation)

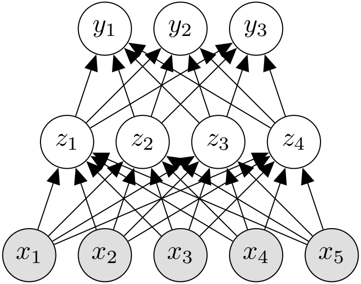
Outline

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

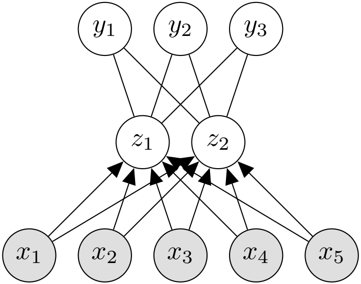


- Direct application, using stochastic gradient descent (SGD) (incremental) and varieties, with back-propagation.
- Unlike classifier chains (etc.) we do not need a structure on the output layer – if the inner layer conditionally removes the dependence on the input
- but empirical data-stream comparisons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

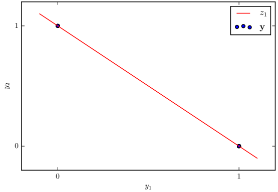
$$\mathbf{Z} = \mathbf{W}\mathbf{Y} \quad \triangleright \mathbf{W} \text{ has top } k \text{ components, } k < L$$

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

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

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

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

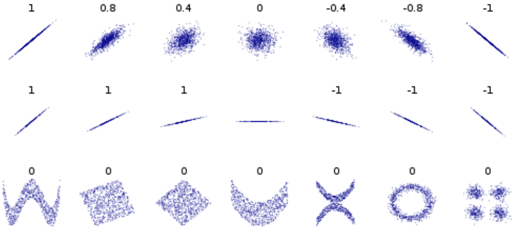


PCA on the Label Space

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

but . . .

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

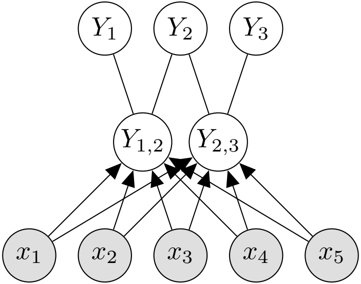


source: Wikipedia

- 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

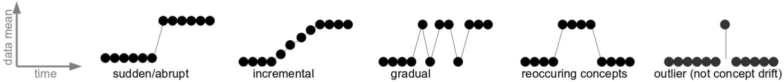
Outline

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

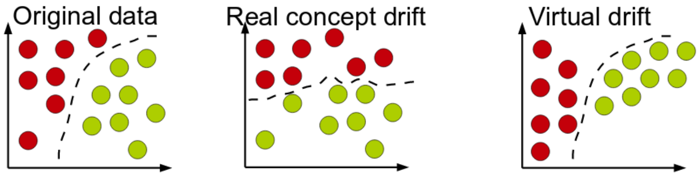
Concept Drift: Real vs Virtual

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:

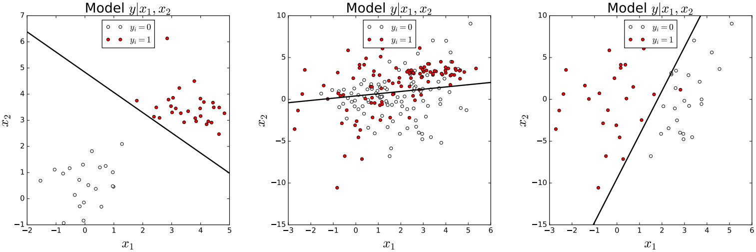
- Real concept drift: changes to $P_t(Y|X)$.
- Virtual drift changes to $P_t(X)$ only.



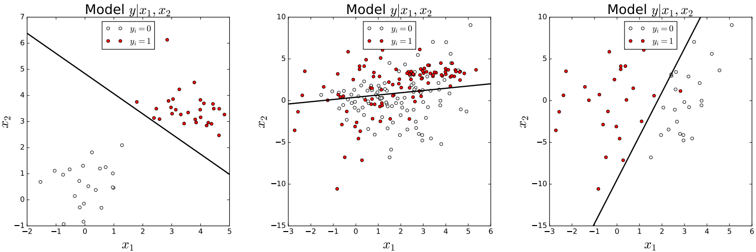
Spam Classification

- 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*.

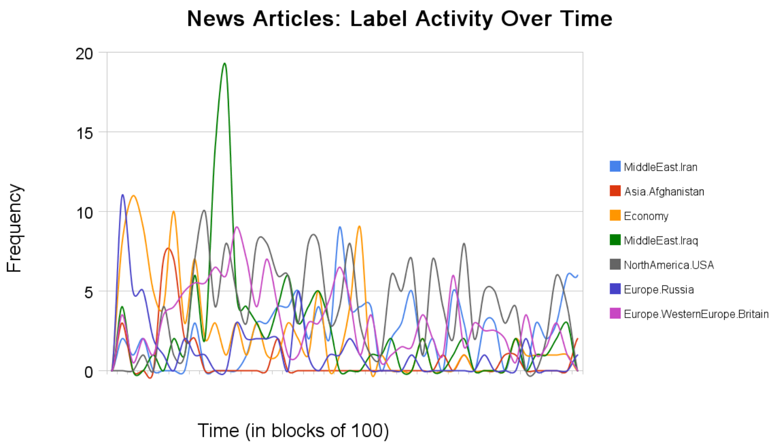


Figure: Multi-label virtual drift: Label frequency / month over time (until about 2007)

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:

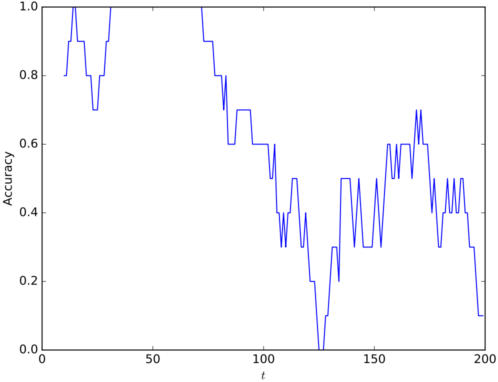
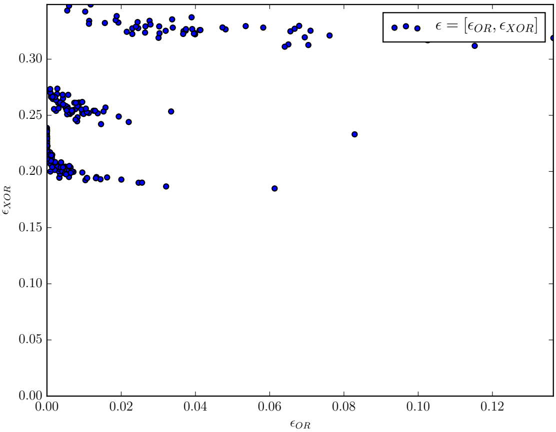


Figure: Accuracy through concept drift ($t = 50, \ldots, 150$).

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

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).

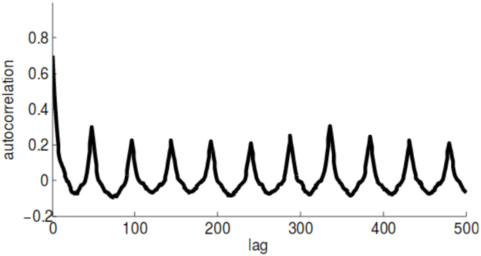


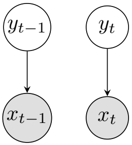
Figure: Auto-correlation function on the Electricity dataset, for $k = 1, 2, \ldots, 500$; source: Indrė Žliobaitė arXiv:1301.3524v1, Jan 2015.

Outline

Naive Bayes

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

$$\hat{y}_t = \underset{y_t \in \{0,1\}}{\operatorname{argmax}} P(y_t, x_t)$$
$$= \underset{y_t \in \{0,1\}}{\operatorname{argmax}} P(x_t|y_t)P(y_t)$$

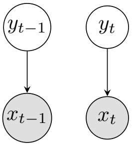


- Not a problem, we maintain an empirical estimate of Ps via counting
- At time $t+1$ we get y_t; we can now update counts with (x_t, y_t)
- We measure error $\epsilon_t = E(y_t - \hat{y}_t)$, look for drift (e.g., ADWIN), etc.

Naive Bayes

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

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

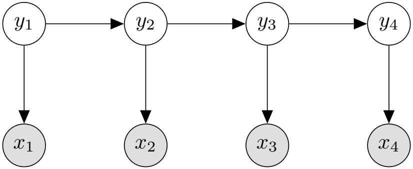


- Not a problem, we maintain an empirical estimate of Ps via counting
- At time $t+1$ we get y_t; we can now update counts with (x_t, y_t)
- We measure error $\epsilon_t = E(y_t - \hat{y}_t)$, look for drift (e.g., ADWIN), etc.

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]

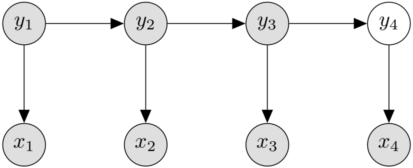
$$
\begin{aligned}
\hat{y}_t &= \underset{y_t \in \{0,1\}}{\operatorname{argmax}} P(y_t, \mathbf{x}_t) \\
&= \underset{y_t \in \{0,1\}}{\operatorname{argmax}} P(y_1) \prod_{\tau=2}^{t} P(x_\tau | y_\tau) P(y_\tau | y_{\tau-1})
\end{aligned}
$$

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:

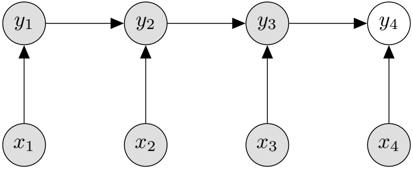


$$\begin{aligned}\hat{y}_t &= \underset{y_t \in \{0,1\}}{\operatorname{argmax}} P(y_t, \mathbf{x}_t, \mathbf{y}_{t-1}) \\ &= \underset{y_t \in \{0,1\}}{\operatorname{argmax}} P(x_t | y_t) P(y_t | y_{t-1})\end{aligned}$$

The recursion stops at $t-1$.

Maximum Entropy Markov Model (MEMM)

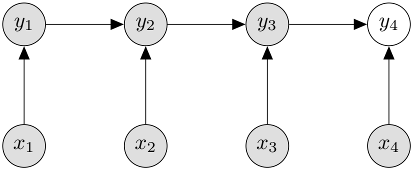
A discriminative approach:



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

Maximum Entropy Markov Model (MEMM)

A discriminative approach:

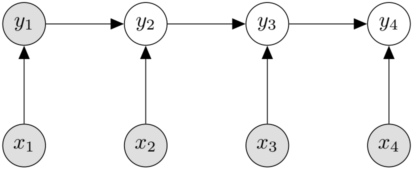


$$
\begin{aligned}
\hat{y}_t &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_t|\mathbf{x}_t) \\
&= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_t|\mathbf{x}_t) P(y_t|y_{t-1}) \\
&= h(x_t, y_{t-1}) \qquad \triangleright \text{ Classifier}
\end{aligned}
$$

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

Forecasting with MEMMs

...



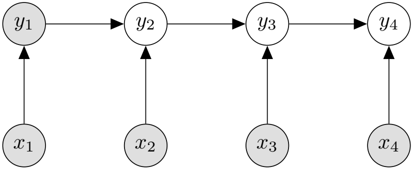
To obtain prediction $\hat{y}_t$, we can use a prediction for $\hat{y}_{t-1}$:

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

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

Forecasting with MEMMs

...



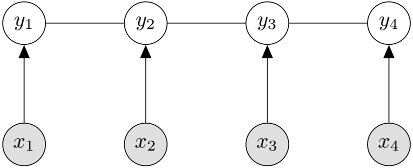
To obtain prediction $\hat{y}_t$, we can use a prediction for $\hat{y}_{t-1}$:

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

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)

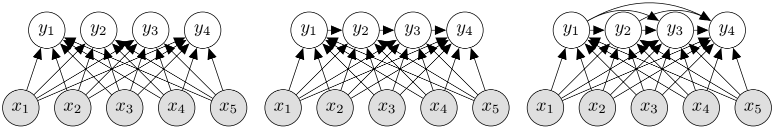
To avoid the error propagation problem, we may use a CRF:



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

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)

$$\hat{\mathbf{y}}_t = \operatorname*{argmax}_{\mathbf{y}\in\{0,1\}^L} p(\mathbf{y}|\mathbf{x};\mathbf{w}) \qquad \triangleright\ \mathbf{w} \text{ is a set of weights}$$

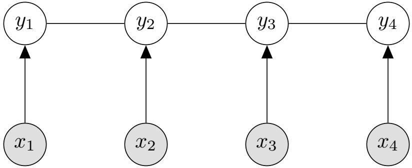
$$= \operatorname*{argmax}_{\mathbf{y}\in\{0,1\}^L} \exp\left\{\sum_k w_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\} \qquad \triangleright \text{ CRF inference}$$

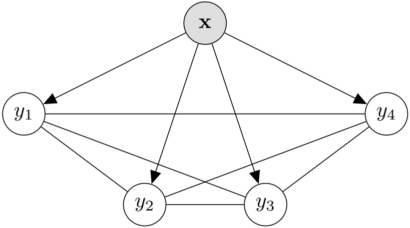
$$= \operatorname*{argmax}_{\mathbf{y}\in\{0,1\}^L} \prod_{t=1}^{T} \exp\left\{w_t \cdot f_t(y_t, y_{t-1}, \mathbf{x}_t)\right\} \qquad \triangleright\ e^{a+b} = e^a e^b$$

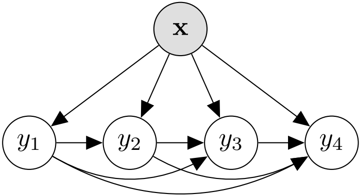
$$= \operatorname*{argmax}_{\mathbf{y}\in\{0,1\}^L} \prod_{t=1}^{T} f_t(y_t, y_{t-1}, \mathbf{x}_t; \mathbf{w}_t) \qquad \triangleright \text{ a generic fn } f$$

$$= \operatorname*{argmax}_{\mathbf{y}\in\{0,1\}^L} f_1(y_1, \mathbf{x}) \prod_{j=2}^{L} f_j(y_1, \ldots, y_{j-1}, \mathbf{x}) \qquad \triangleright \text{ PCC!}$$

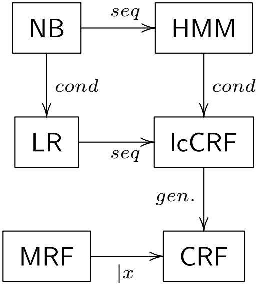
$$= \operatorname*{argmax}_{\mathbf{y}\in\{0,1\}^L} P(y_1|\mathbf{x}) \prod_{j=2}^{L} P(y_j|y_1, \ldots, y_{j-1}, \mathbf{x}) \qquad \triangleright \text{ where } f_j \approx P_j$$







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

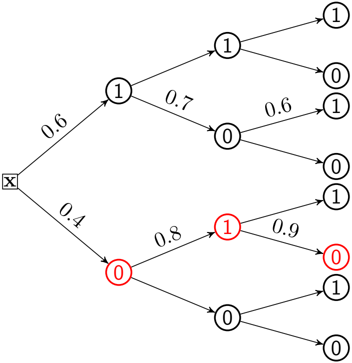


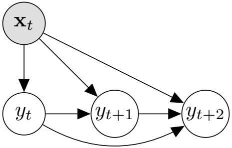
Distribution	Type
$p(y, \mathbf{x})$	NB
$p(\mathbf{y}, \mathbf{x})$	HMM
$p(y\|\mathbf{x})$	LR
$p(\mathbf{y}\|\mathbf{x})$	CRF
$p(\mathbf{y})$	MRF

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

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

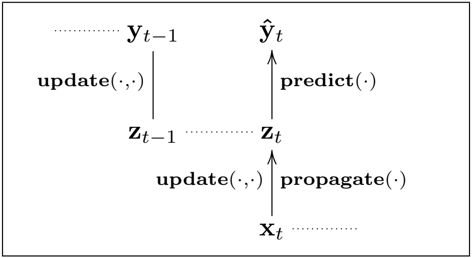
What if we *never* get true labels for some $\mathbf{x}_t$? In many applications it is unrealistic to expect labels for every instance. During learning, we can:

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)

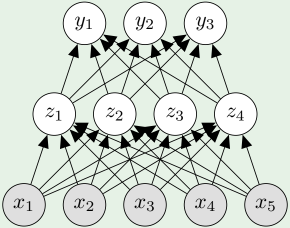


Semi-Supervised Data Streams

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

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