

Streams Processing

Supervised learning

Supervised learning

Regression and classification

Regression and classification for streams

Regression

- Given a numeric class attribute, a regression algorithm builds a model that accurately predicts for every unlabelled instance attribute x a numeric value.
- Examples
 - Stock-market price prediction
 - Airplane delays

What is regression?

Given a large, noisy dataset $A = [X \ y]$

1. Try to find a **simple** model $M(X)$
2. Such that the data closely **follows** the model $M(x_i) \approx y_i$

The model allows to **predict** y from x

Q: Why not use PCA?

A: We want more than being close to all data: given an x we want to predict y

Evaluation

1. Error estimation: *Hold-out or Prequential*
2. Evaluation performance measures: *MSE or MAE*
3. Statistical significance validation: *Nemenyi test*

How to measure performance?

Regression **mean** measures

- ▶ Mean square error:

$$MSE = \sum (f(x_i) - y_i)^2 / N$$

- ▶ Root mean square error:

$$RMSE = \sqrt{MSE} = \sqrt{\sum (f(x_i) - y_i)^2 / N}$$

Forgetting mechanism for estimating measures

Sliding window of size w with the most recent observations

How to measure performance?

Regression **absolute** measures

- ▶ Mean absolute error:

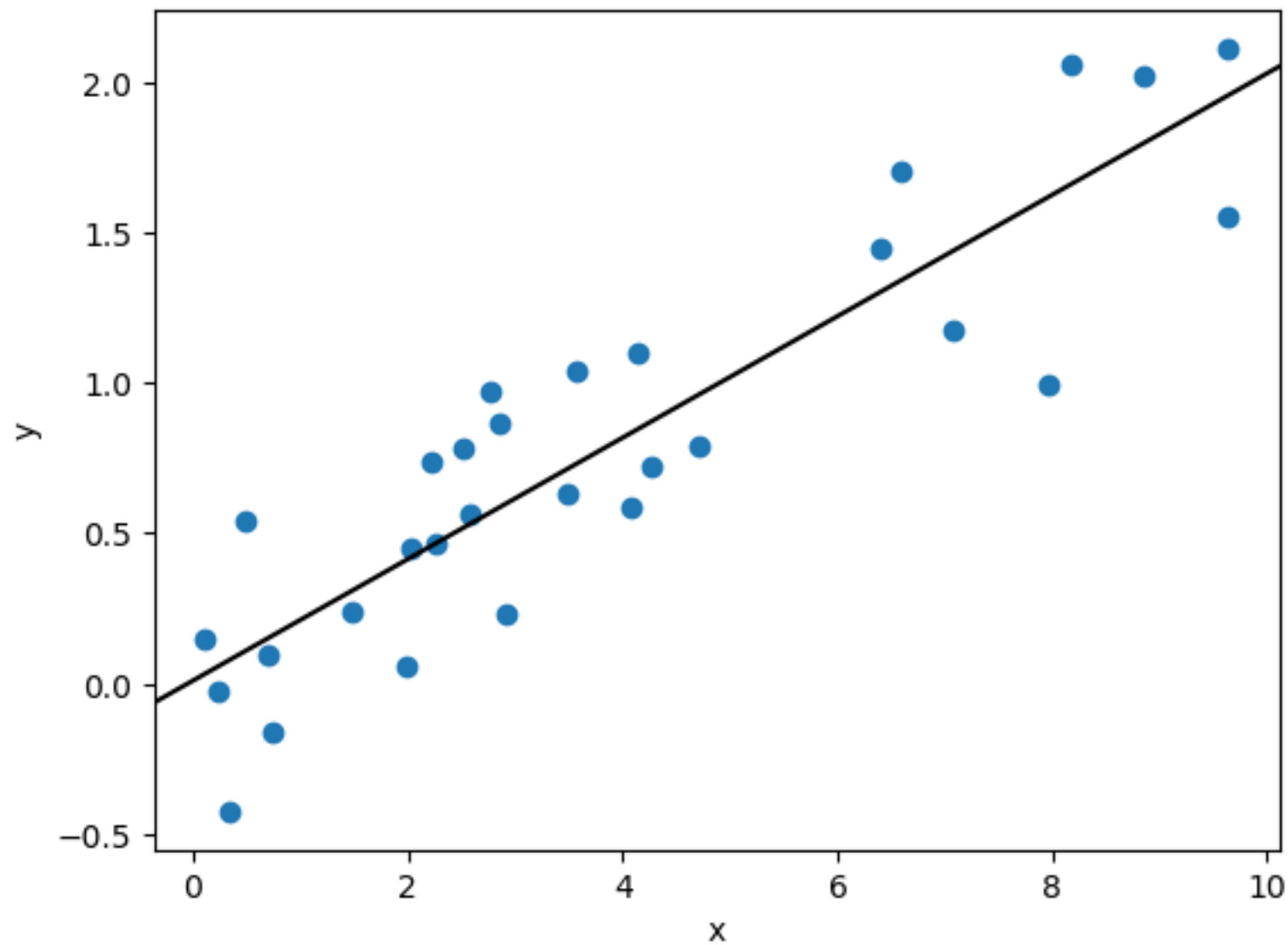
$$MAE = \sum (|f(x_i) - y_i|) / N$$

- ▶ Relative absolute error:

$$RAE = \sum (|f(x_i) - y_i|) / \sum (|\hat{y}_i - y_i|)$$

Linear regression

$$M(x) = ax + b$$



Minimizing the residual

We want to use the explanatory variable x to predict the dependent variable y

$$M(x) = \hat{y}$$

Residual:

$$\begin{aligned} r_i &= |y_i - \hat{y}_i| \\ &= |y_i - M(x_i)| \end{aligned}$$

Measuring fit error
on a dataset:

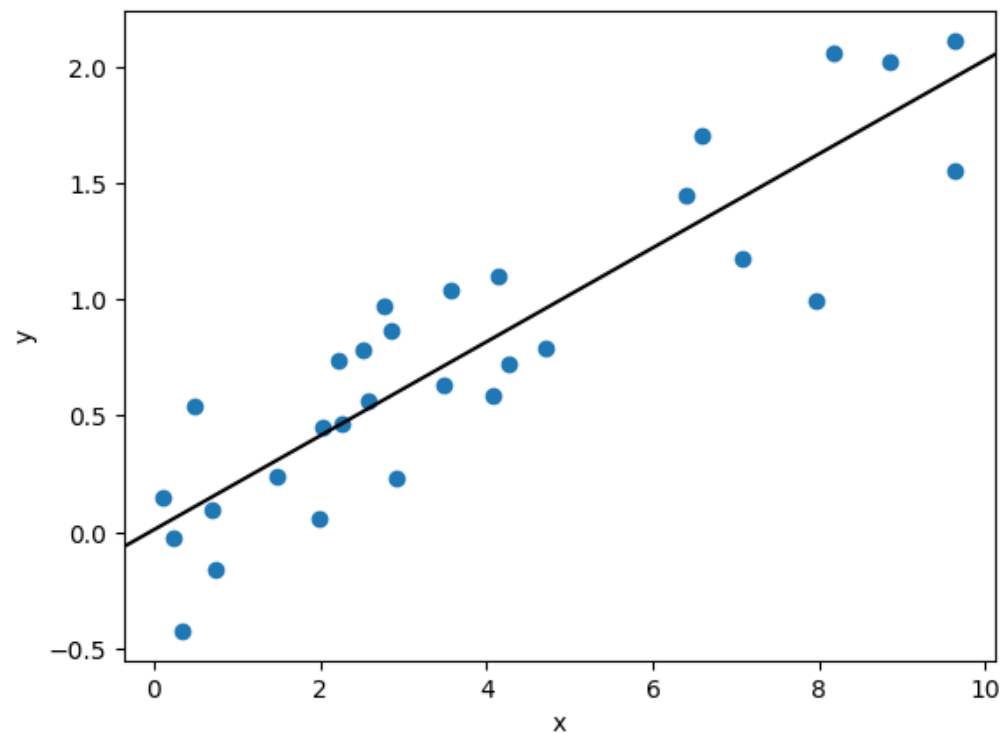
$$\sum_{i=1}^n (M(x_i) - y_i)^2$$

In 2D

$$\underset{a,b}{\text{minimize}} \sum_{i=1}^n (a^T x_i - b - y_i)^2$$

$$\alpha = [b; a]$$

$$X = [1 \ X_1]$$



Q: Is the residual the distance to the line?

A: No, it is the error of the value of the model evaluated at x and the true y

Closed form solution

$$\alpha = [\alpha_0; \alpha_1; \dots; \alpha_d]$$

$$X = [\mathbf{1}, X_1, X_2, \dots, X_d]$$

$$\underset{\alpha}{\text{minimize}} \quad \|X\alpha - y\|^2$$

$$\alpha = (X^T X)^{-1} X^T y$$

Matrices too large!

Classification

Given k different classes, a classifier algorithm builds a model that accurately predicts for every unlabelled instance x the class C to which the instance belongs.

Examples:

Spam filter

Twitter sentiment analysis

What is classification?

Given a large, noisy dataset $A = [X \ y]$

Where y takes values in a finite, unordered set of labels

1. Try to find a **simple** model $M(X)$
2. Such that the data closely **reproduces** the model $M(x_i) \approx y_i$

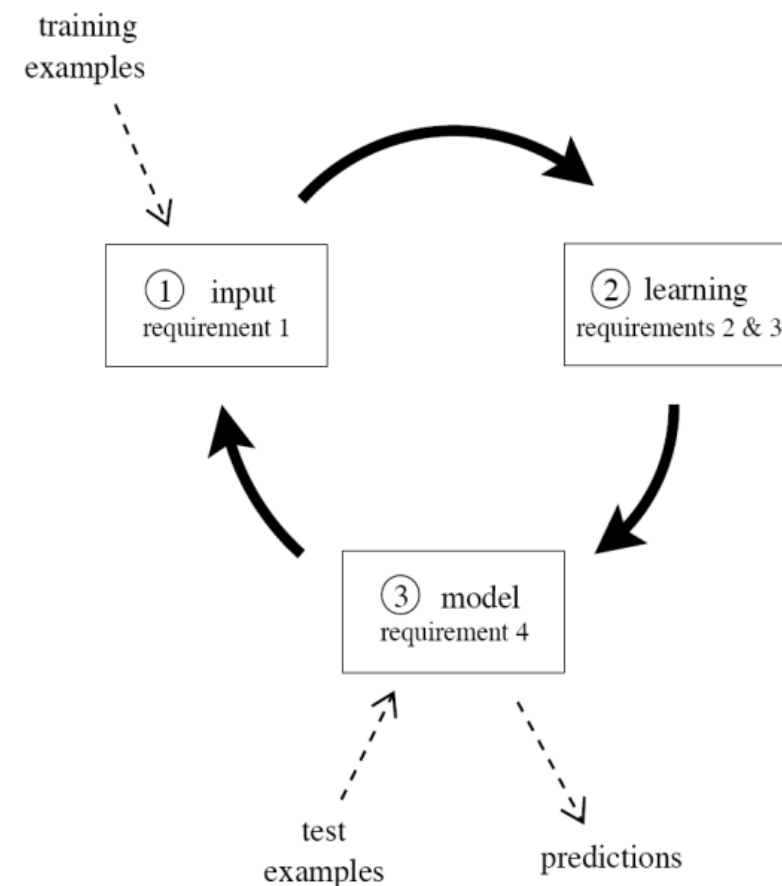
The model allows to **predict** y from x

Q: Why not use regression?

A: We do not wish to follow **ordered continuous** values

Classification for streams

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



Logistic regression

Training the logistic regression classifier is finding a solution to

$$\underset{\alpha}{\text{minimize}} \sum_{n=1}^N \log(1 + \exp(y_n X_n^T \alpha))$$

Fitting a model to data

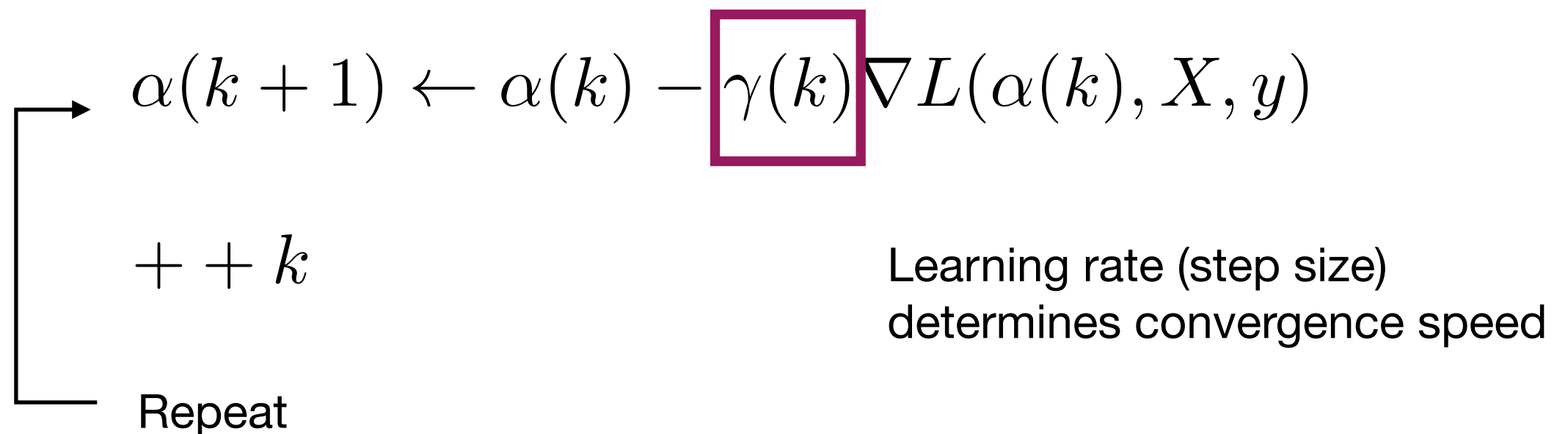
We are minimizing a loss over the model parameters

$$\underset{\alpha \in \mathcal{A}}{\text{minimize}} L(\alpha, X, y)$$

$$L(\alpha, X, y) = \sum_{n=1}^N L(\alpha, x_n, y_n)$$

In river: optim class

Gradient descent



The diagram illustrates a loop structure for gradient descent. A vertical line on the left has an arrow pointing to the right, entering a loop. The loop body contains the update equation, an increment of the iteration counter, and a 'Repeat' label. The learning rate $\gamma(k)$ is highlighted with a red box.

$$\alpha(k+1) \leftarrow \alpha(k) - \gamma(k) \nabla L(\alpha(k), X, y)$$

$++k$

Repeat

Learning rate (step size)
determines convergence speed

N passes over the data just to compute **one** GD iteration!

Gradient descent convergence

Strong assumptions:

Convex loss, for which the infimum can be attained

With Lipschitz continuous gradient

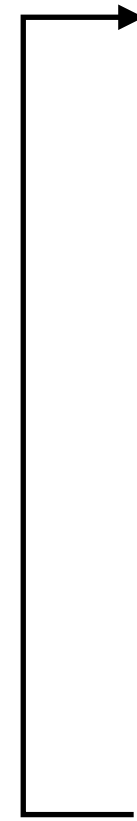
Lipschitz constant

$$\|\nabla L(\alpha) - \nabla L(\beta)\| \leq D \|\alpha - \beta\|$$

$$\forall \alpha, \beta \in \text{dom}(\nabla L)$$

Largest possible $\gamma(k) = \frac{1}{D}$

Accelerated gradient method


$$\beta = \alpha(k-1) + \frac{k-2}{k+1}(\alpha(k-1) - \alpha(k-2))$$
$$\alpha(k) = \beta - \gamma(k)\nabla L(\beta, X, y)$$

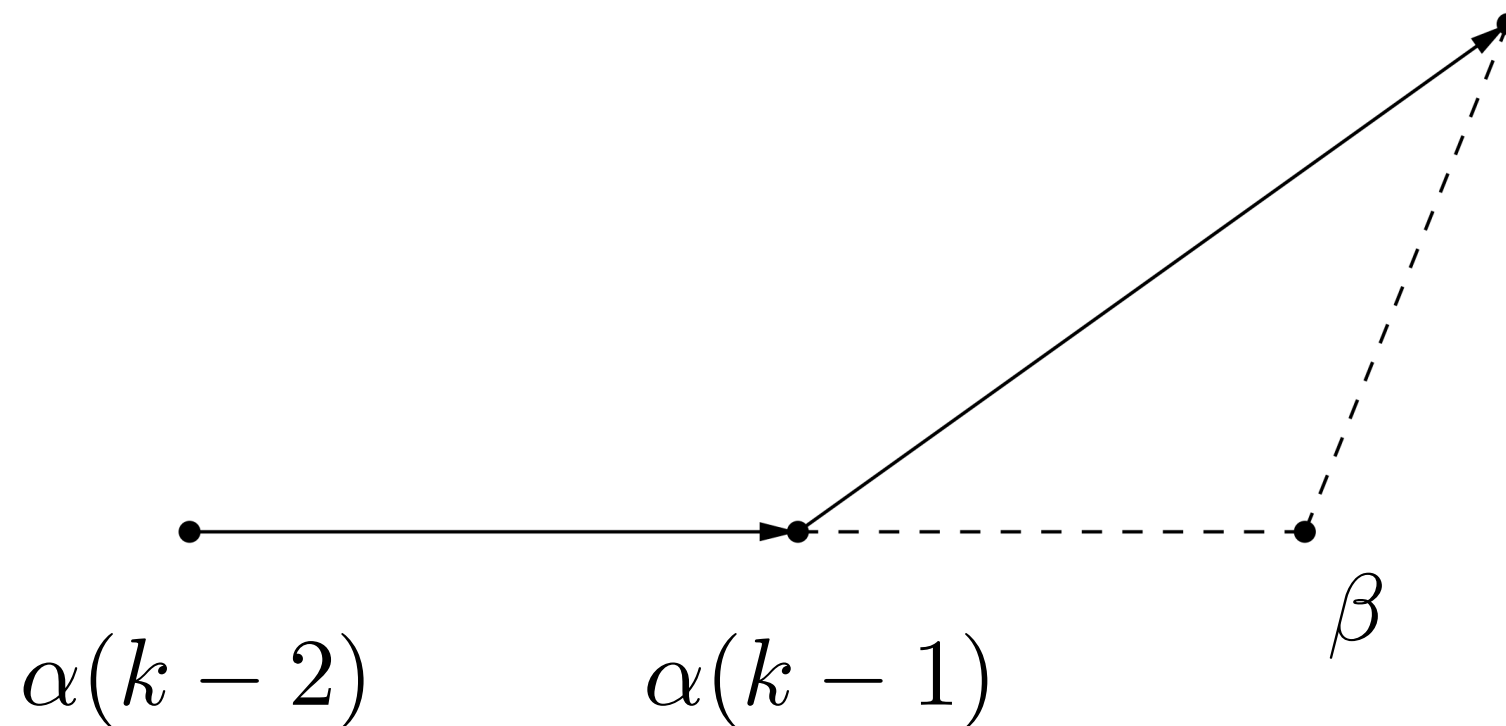
++ k

Repeat

Extrapolated point

$$\beta = \alpha(k-1) + \frac{k-2}{k+1}(\alpha(k-1) - \alpha(k-2))$$

$$\alpha(k) = \beta - \gamma(k)\nabla L(\beta)$$



Incremental batch gradient method

Approximate

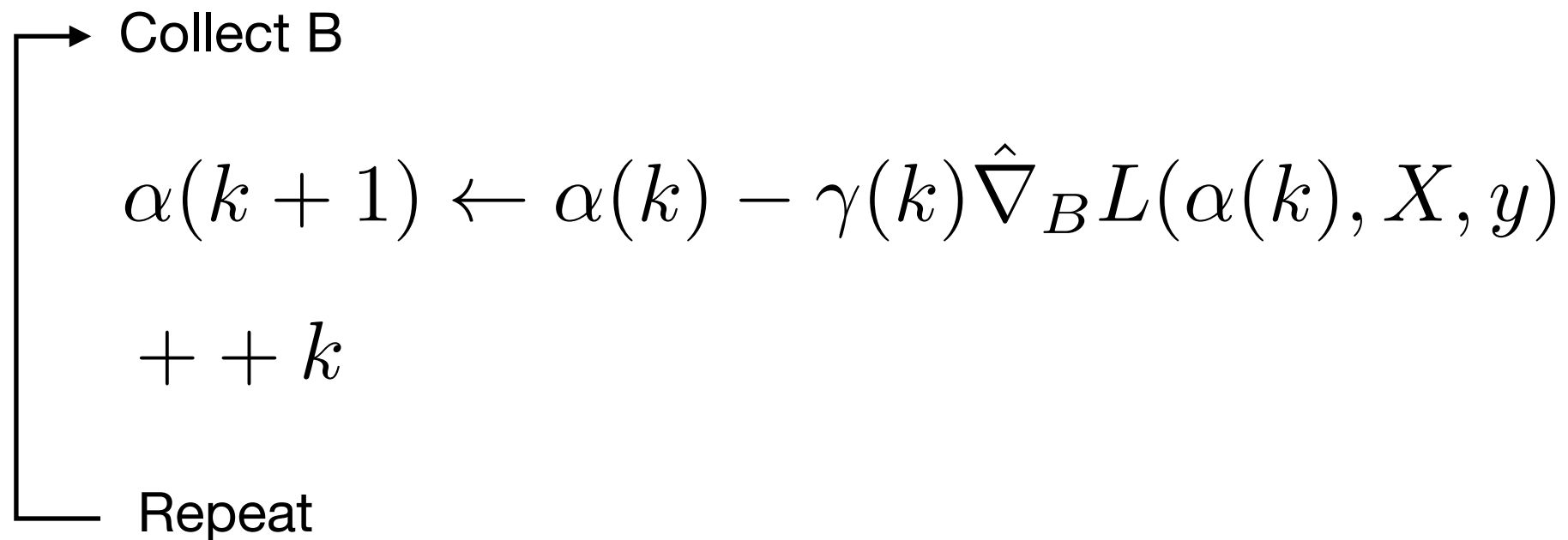
$$\nabla L(\alpha, X, y) = \sum_{n=1}^N \nabla L(\alpha, X_n, y_n)$$

By a noisy gradient

$$\hat{\nabla}_B L(\alpha, X, y) = \sum_{n \in B} \nabla L(\alpha, X_n, y_n)$$

Where B is a batch of data, in sequence. Allows for learning **data streams**.

Incremental batch gradient method



Why should it work?

In the iteration perspective, the minimized function is always changing!

Main assumption:

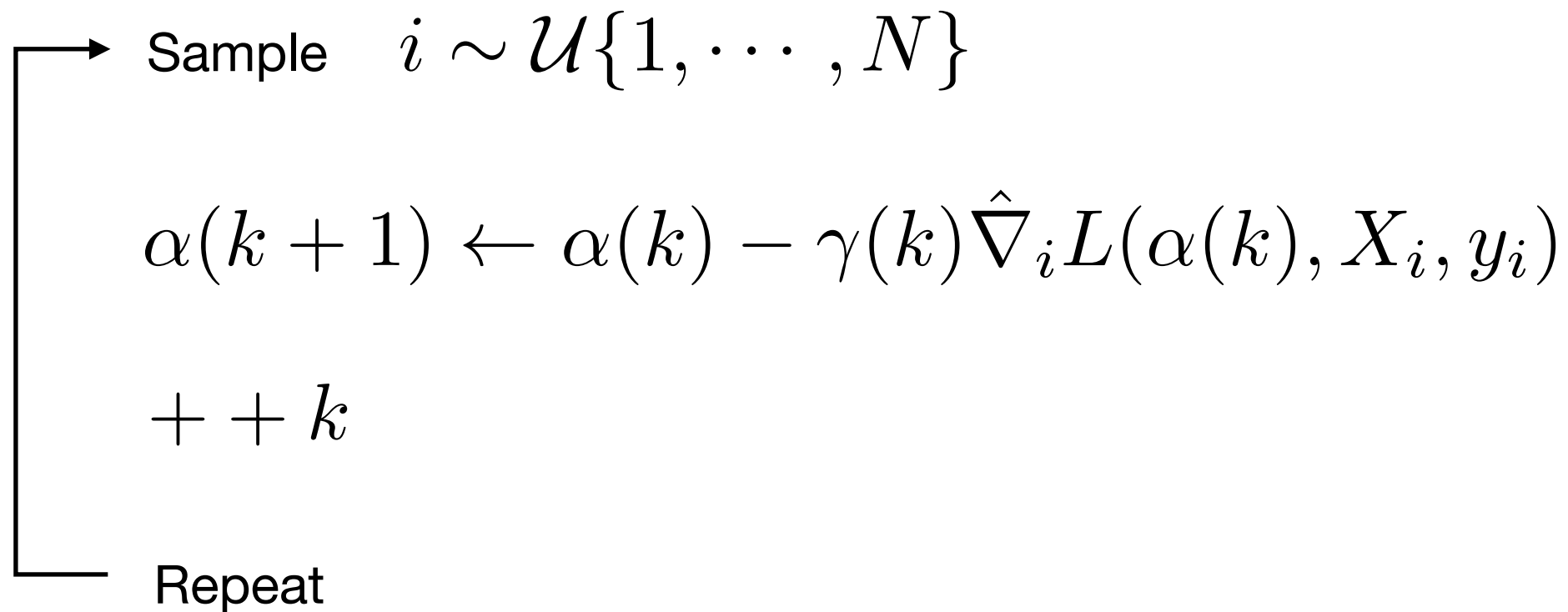
The data is produced by the same underlying process

The model loss will behave similarly for all data points

Assumption valid on learning problems,
not extensible to general optimization problems with the form:

$$f(x) = \sum_i f_i(x_i)$$

Stochastic gradient method



Choice of step-size: $\sum_{k=0}^{\infty} \gamma(k) = \infty \quad \sum_{k=0}^{\infty} (\gamma(k))^2 < \infty$

Acceleration

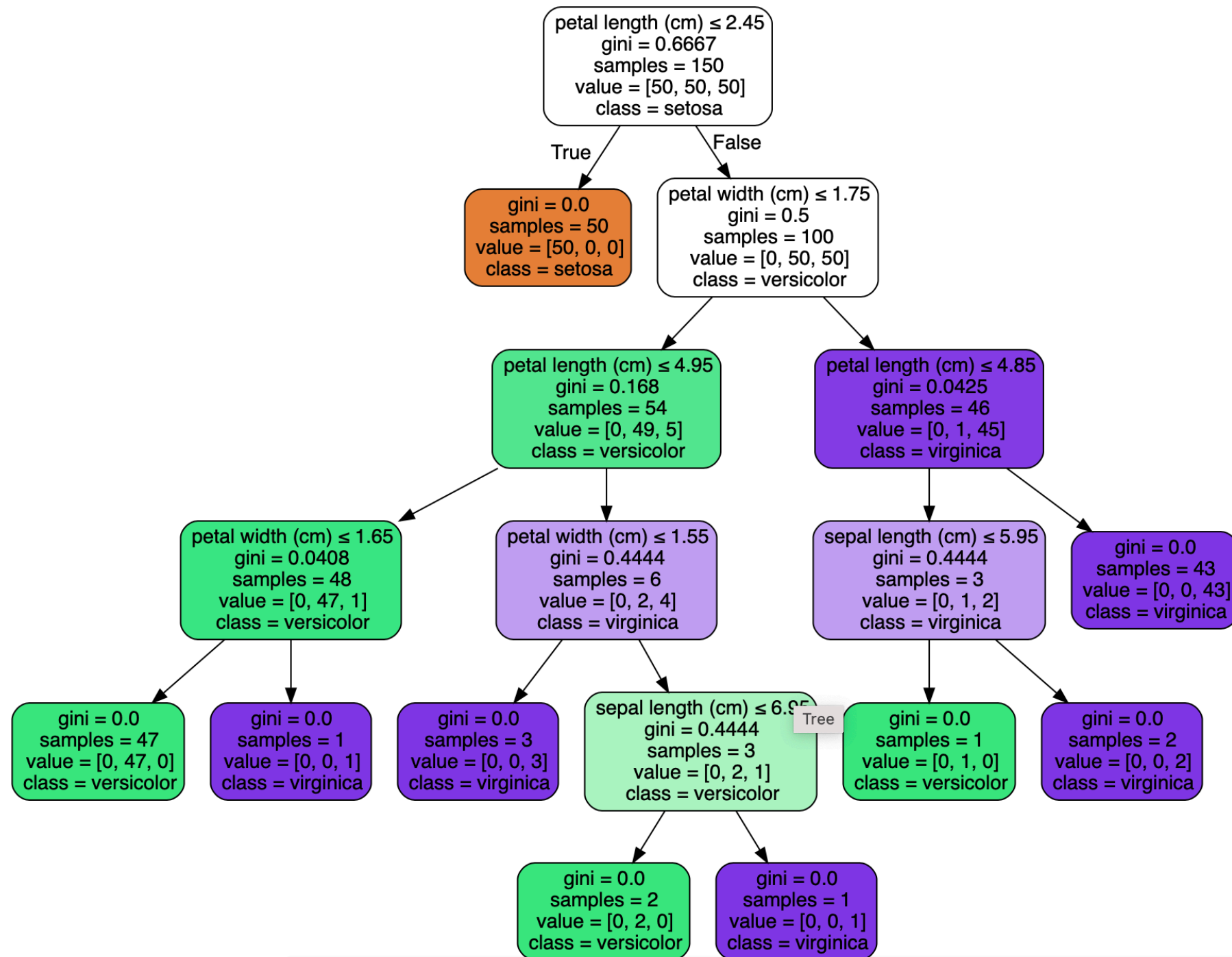
Used in training of Big Data models
and other specific fields like NLP, CV, etc

No guarantees

Decision trees

- A very popular classifier technique
- Tree models are very easy to interpret and visualize
- In a decision tree, each internal node corresponds to an attribute that splits into a branch for each attribute value, and leaves correspond to classification predictors
- Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed.
- Can be unstable. Small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.

Example: iris dataset



Hoeffding trees: decision trees for streams

- Pedro Domingos and Geoff Hulten (VFDT)
- Only sees a training data point once
- With high probability, constructs an identical model that a traditional (greedy) method would learn
- Has theoretical guarantees on the error rate
- Remember Hoeffding inequality? (Lecture 1 of the module)
- In river: <https://riverml.xyz/latest/api/tree/HoeffdingTreeClassifier/>

Gini index

- Popularized by the CART (classification and regression tree) algorithm for classification trees
- A measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset
- For J classes it can be computed as $G = \sum_{j=1}^J p_j (1 - p_j)$ where p_j is the fraction of items labeled as class j

VFDT main algorithm

$HT(Stream, \delta)$

- 1 ▷ Let HT be a tree with a single leaf(root)
- 2 ▷ Init counts n_{ijk} at root
- 3 **for** each example (x, y) in Stream
- 4 **do** HTGROW($(x, y), HT, \delta$)

VFDT grow tree

HTGROW((x, y) , HT , δ)

- 1 ▷ Sort (x, y) to leaf l using HT
- 2 ▷ Update counts n_{ijk} at leaf l
- 3 **if** examples seen so far at l are not all of the same class
- 4 **then** ▷ Compute G for each attribute
- 5 **if** $G(\text{Best Attr.}) - G(\text{2nd best}) > \sqrt{\frac{R^2 \ln 1/\delta}{2n}}$
- 6 **then** ▷ Split leaf on best attribute
- 7 **for** each branch
- 8 **do** ▷ Start new leaf and initialize counts

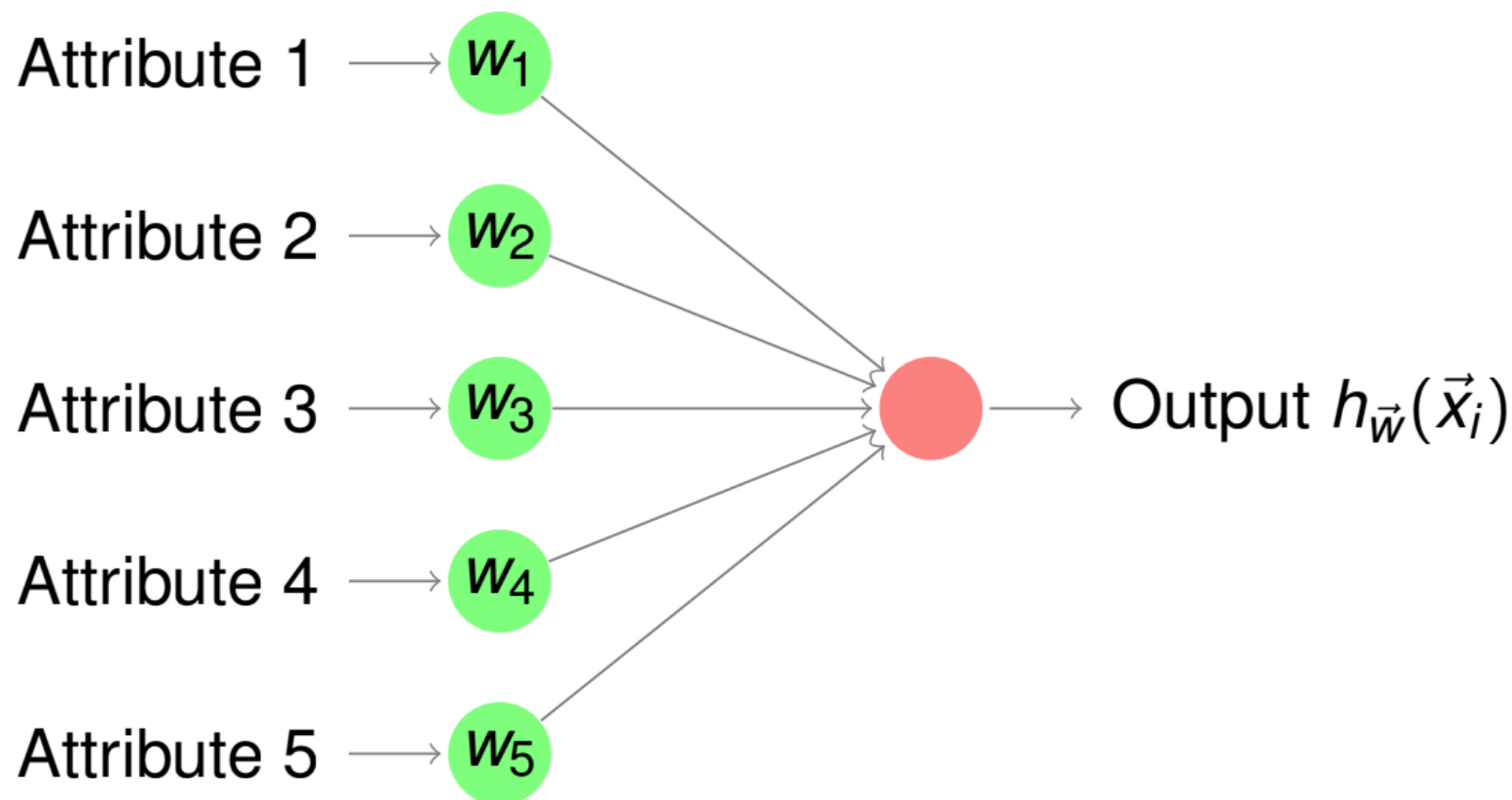
Dealing with drift: Hoeffding Adaptive Tree

Hoeffding Adaptive Tree:

- ▶ replace frequency statistics counters by estimators
 - ▶ don't need a window to store examples, due to the fact that we maintain the statistics data needed with estimators
- ▶ change the way of checking the substitution of alternate subtrees, using a change detector with theoretical guarantees (ADWIN)

In river: <https://riverml.xyz/latest/api/tree/HoeffdingAdaptiveTreeClassifier/>

Perceptron (Neural Network)



- ▶ Data stream: $\langle \vec{x}_i, y_i \rangle$
- ▶ Classical perceptron: $h_{\vec{w}}(\vec{x}_i) = \text{sgn}(\vec{w}^T \vec{x}_i)$,
- ▶ Minimize Mean-square error $J(w) = \frac{1}{N} \sum_{n=1}^N (h_w(x_n) - y_n)^2$

Perceptron (Neural Network)

- As the sign is not differentiable, we use the Sigmoid function

$$h_{\vec{w}} = \sigma(\vec{w}^T \vec{x})$$

- $\sigma(x) = 1/(1 + e^{-x})$

Learning the Perceptron: minimize MSE $J(\mathbf{w})$

- ▶ Stochastic Gradient Descent: $\vec{w} = \vec{w} - \eta \nabla J \vec{x}_i$
- ▶ Gradient of the error function:

$$\nabla J = - \sum_i (y_i - h_{\vec{w}}(\vec{x}_i)) \nabla h_{\vec{w}}(\vec{x}_i)$$

$$\nabla h_{\vec{w}}(\vec{x}_i) = h_{\vec{w}}(\vec{x}_i)(1 - h_{\vec{w}}(\vec{x}_i))$$

- ▶ Weight update rule

$$\vec{w} = \vec{w} + \eta \sum_i (y_i - h_{\vec{w}}(\vec{x}_i)) h_{\vec{w}}(\vec{x}_i)(1 - h_{\vec{w}}(\vec{x}_i)) \vec{x}_i$$

Perceptron for streams

- ▶ Stochastic Gradient Descent: $\vec{w} = \vec{w} - \eta \nabla J \vec{x}_i$
- ▶ Gradient of the error function:

$$\nabla J = - \sum_i (y_i - h_{\vec{w}}(\vec{x}_i)) \nabla h_{\vec{w}}(\vec{x}_i)$$

$$\nabla h_{\vec{w}}(\vec{x}_i) = h_{\vec{w}}(\vec{x}_i)(1 - h_{\vec{w}}(\vec{x}_i))$$

- ▶ Weight update rule

$$\vec{w} = \vec{w} + \eta \sum_i (y_i - h_{\vec{w}}(\vec{x}_i)) h_{\vec{w}}(\vec{x}_i)(1 - h_{\vec{w}}(\vec{x}_i)) \vec{x}_i$$