Online Machine Learning

Online Classification Models

Antonio Carta

antonio.carta@unipi.it

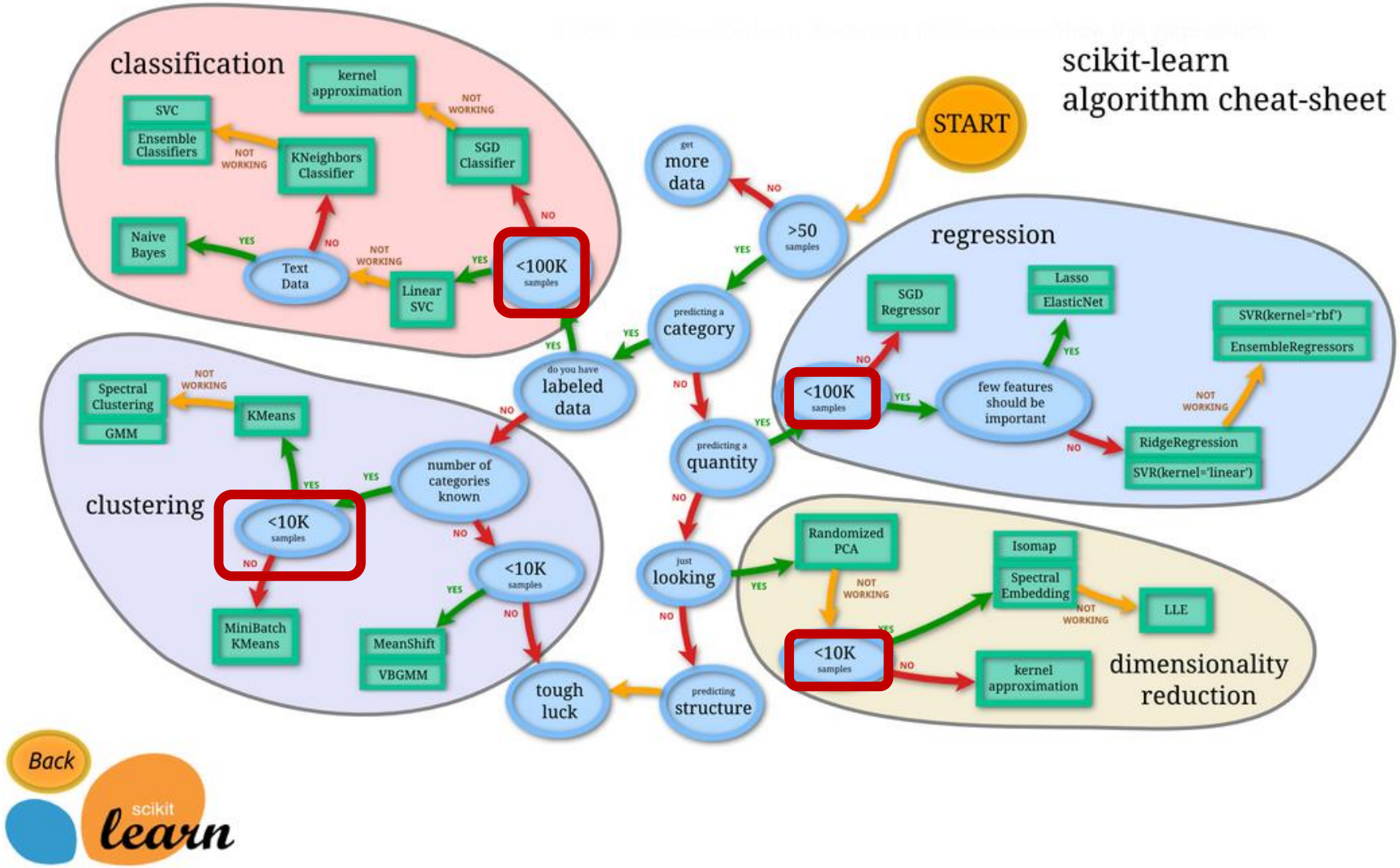
Outline

- **Basics**: what you can and can’t do in an online method
- **Online Methods**: Naive Bayes, SGD
- **From offline to online methods**:
  - kNN -> online kNN
  - Decision Tree -> Hoeffding Tree

What is an Online Classification Method

- **Streaming Classification Models**
  - Models that take as input an infinite stream
  - We can’t save the input data
  - We have latency and computational constraints
- **Out-of-Core Methods**
  - Models that are too expensive to be trained on the entire dataset in one step
  - Only computation constraints
  - We don’t have drifts here, the data is static (e.g. we can shuffle it)

Out-of-core Methods



Source: scikit-learn docs

Training Online – Amount of Data

- **We cannot keep the entire stream in memory**
  - We cannot shuffle the data (except in out-of-core)
- We can keep a **small buffer** but we **cannot retrain from scratch** at each step (latency)
- We can update the model using **small mini-batches**

algorithms need to be able to take the previous model and a small batch of samples as input and return a new model

- $\mathbf{A}: \theta_{t-1}, D \rightarrow \theta_t$

Pretraining and Warm Start

- Many online algorithms are susceptible to larger changes in the first phases of training
  - Others suffer from bad initializations
- We may have an initial model, pretrained on some static data
- We can use it as init for the online model
- Called «Warm start», «finetuning», depending on the field
- This is The Way in the Deep Learning world

Online Computations

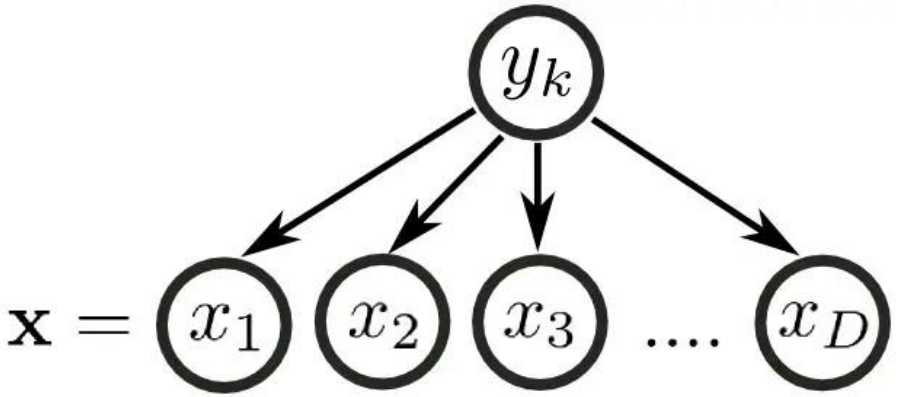
- Some **statistics can be easily computed online**
  - But we need to change the algorithm
- **Approximations**:
  - Can we give correctness bound?
- **Estimate from a buffer**:
  - How much do we wait for data before doing the initial estimate?
- **Sketching algorithms**:
  - Compute approximated statistics over the entire stream
  - Typically with guarantees over memory/time/error
  - We have seen exponential sketching with ADWIN

Online Methods

Naive Bayes and Stochastic Gradient Descent

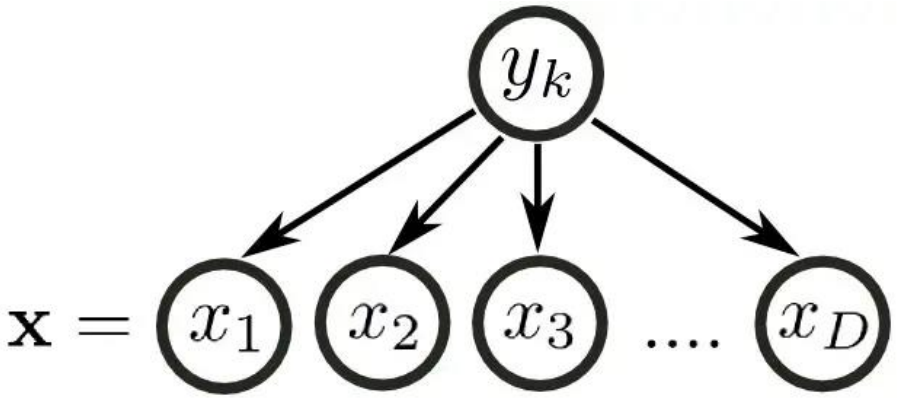
Naive Bayes

- Bayesian method
- Classification with Naive Bayes: $\text{argmax}_i p(x, y_i)$
- **Conditional independence assumptions**: the input features are statistically independent between themselves given the target
- $p(\mathbf{x}, y_k) = p(x_1 \mid y_k)p(x_2 \mid y_k) \ldots p(x_D \mid y_k)p(y_k) = p(y_k)\prod_{i=1}^{D} p(x_i \mid y_k)$



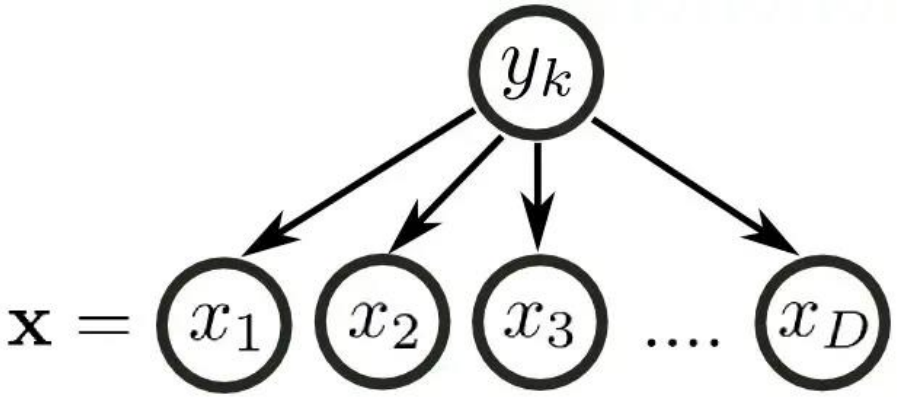
Naive Bayes

- $p(\mathbf{x}, y_k) = p(y_k) \prod_{i=1}^{D} p(x_i \mid y_k)$
- **TRAINING**: estimate conditional probabilities $p(x_i|y_k)$ and priors $p(y_k)$
- **PROBLEM**: online estimate of conditional probabilities



Naive Bayes – Prior

- $p(\mathbf{x}, y_k) = p(y_k) \prod_{i=1}^{D} p(\, x_i \mid y_k \,)$
- N examples in the dataset
  - In streams count how many samples seen up to now
- N_k samples of class k



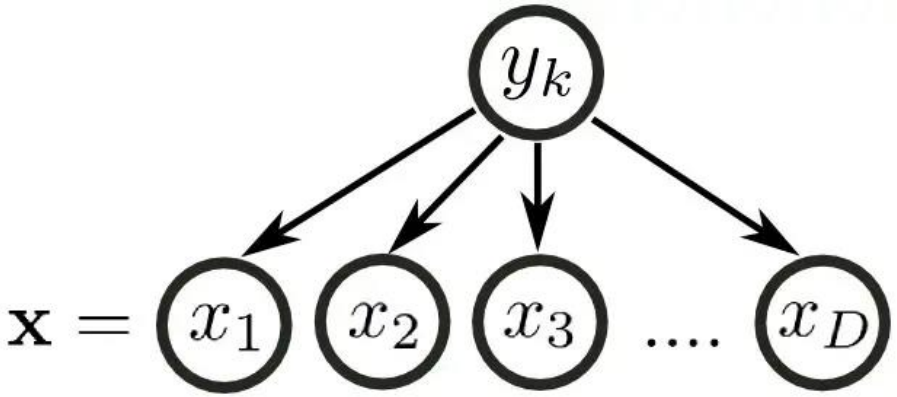
- priors $p(y_k)$
- OFFLINE: N_k/N
- ONLINE: N_k/N
  - Both quantities can be computed online

Naive Bayes – Discrete Conditional Probabilities

- $p(\mathbf{x}, y_k) = p(y_k) \prod_{i=1}^{D} p(x_i \mid y_k)$
- N_k samples of class k

- Conditional probability $p(x_i|y_k)$
  - It's a table of counters
  - For each possible value of x_i counts its occurrence in the training set

- OFFLINE: $M_{j,k}^i / N_k$
- ONLINE: $M_{j,k}^i / N_k$
  - Both quantities can be computed online



Bayesian models are online methods

- Bayes theorem: $posterior = \frac{likelihood \times prior}{evidence}$
- Training for bayesian models = estimating posterior of the parameters:
  - $p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}$
- Online training for bayesian models: the posterior becomes the prior for the next step
  - $p(\theta_t|D) = \frac{p(D_t|\theta_t)p(\theta_{t-1}|D_{t-1})}{p(D_t)}$

Bayesian models are online methods

- the posterior becomes the prior for the next step

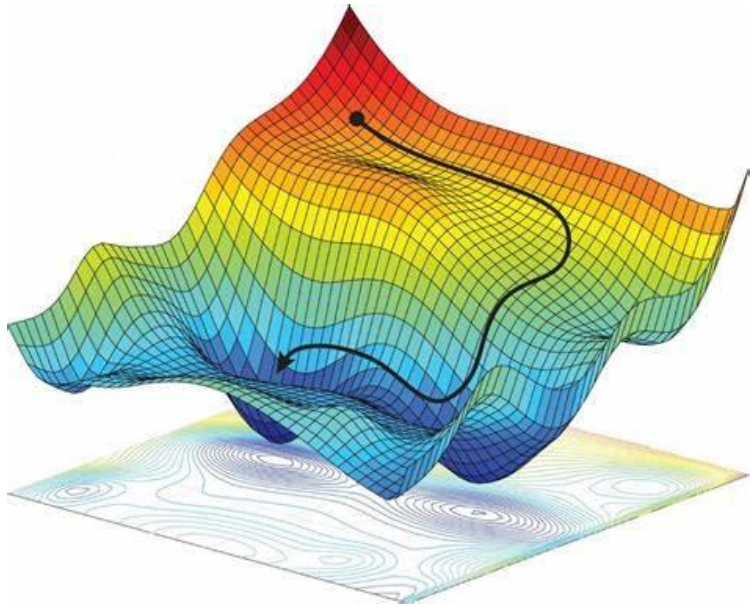
$$p(\theta_t|D) = \frac{p(D_t|\theta_t)p(\theta_{t-1}|D_{t-1})}{p(D_t)}$$

LIMITATIONS

- The posterior is often approximated so online may still have large errors
- Any error is multiplicated, so it can have a large effect over time
- We are also ignoring robustness to drifts and computational limitations

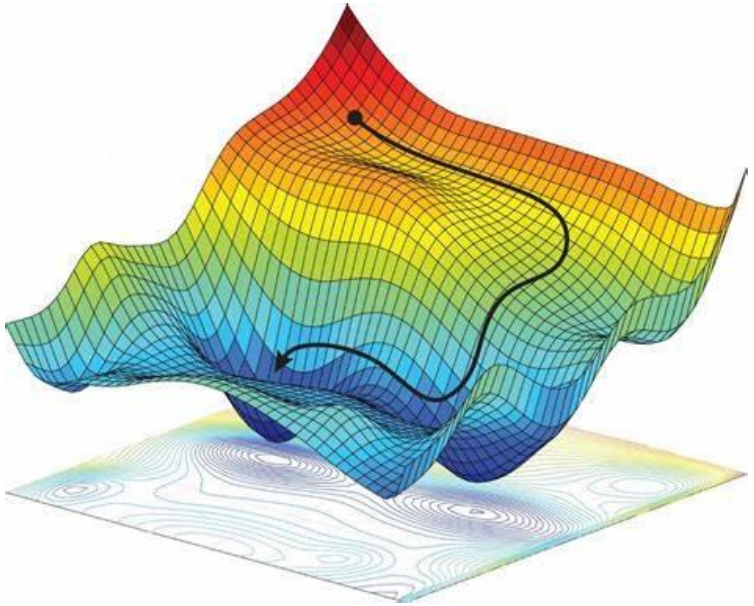
SGD – Stochastic Gradient Descent

- **Classification model**: $y = f_\theta(x)$
  - f is a differentiable function
  - θ are its parameters
  - y are probabilities for each class or normalized to become probabilities
- **Optimization Objective**:
  - Given a loss function $L(\theta, x)$
  - Find a minimum θ^*
  - A local minimum is s.t. $\nabla L(\theta, x) = 0$



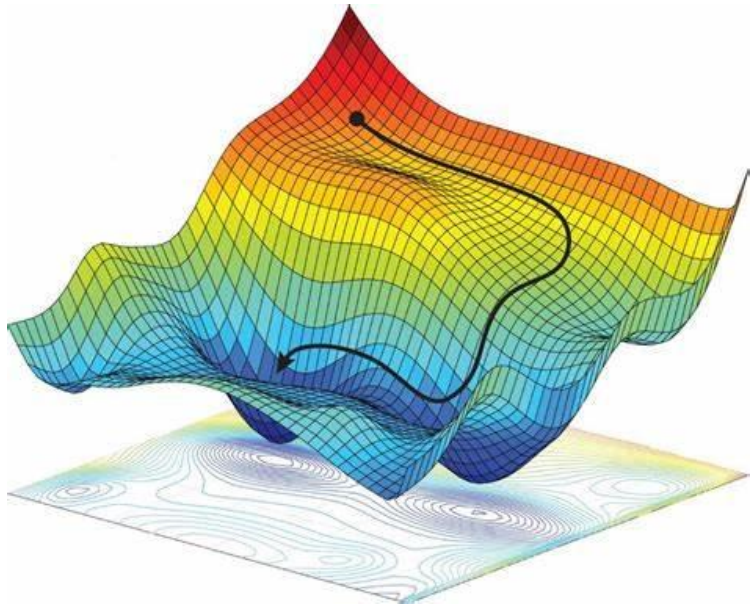
SGD – Algorithm

- **Gradient Descent**:
  - Iterate descent steps until convergence
  - Descent step: $\theta_{i+1} = \theta_i - \lambda \nabla L(\theta_i, x)$
    - λ learning rate
- How do we choose x?
- **Streaming**: x is the current element of the stream x_t
- **Out-of-core**: we sample i.i.d.



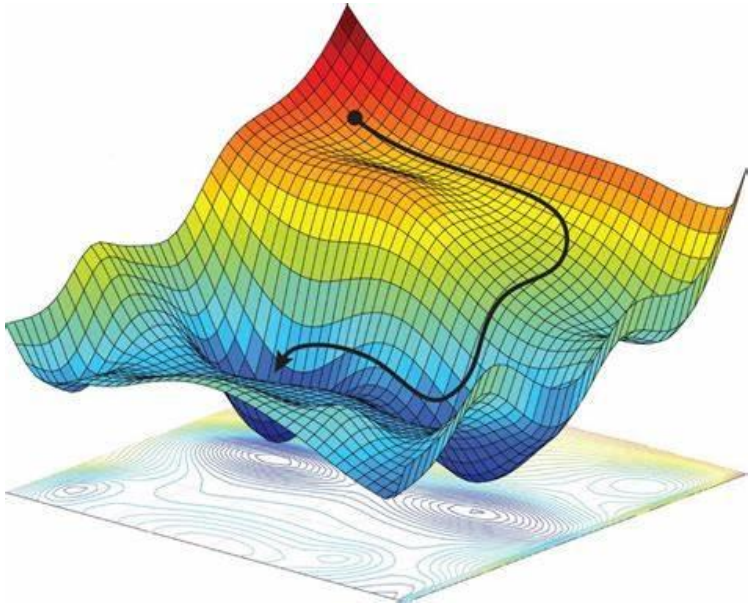
SGD – Online vs Mini-batch

- Often we update with **mini-batches** instead of single examples
- **Noise tradeoff**
  - Smaller batch size -> more noise
  - Noise can help escape local minima
  - Too much noise can slow convergence
- **Computational tradeoff**
  - With GPUs and manycore, parallelization over the batch size is trivial
  - Ideally, you want as many samples as you can fit in memory and compute in parallel
- **Latency tradeoff** (streaming)
  - If we want larger mini-batches we need to wait more data
  - This is more important for inference than training



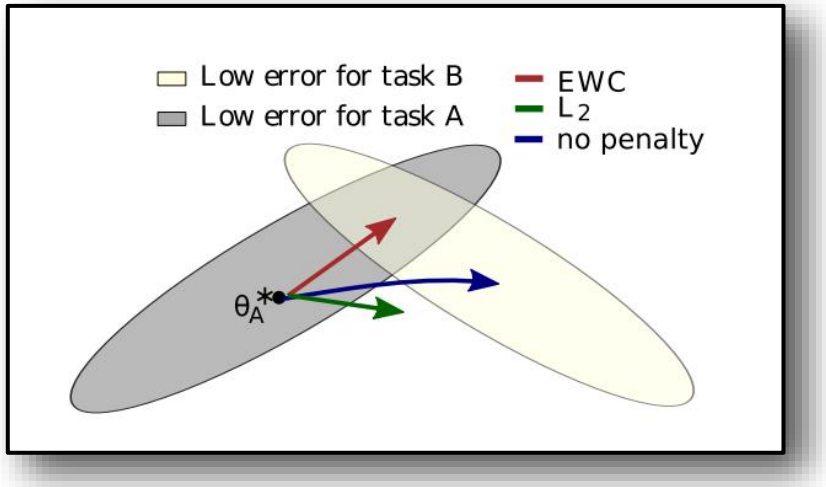
SGD – Convergence and Advantages

- Only local convergence is guaranteed
- Fast method
- Simple to implement with current libraries that perform autodifferentiation
- (out-of-core) Scales to huge datasets



SGD – i.i.d. assumption

- SGD searches a local minimum for $L(\theta, x)$
  - Assumes x are iid
- In presence of drifts, it will soon adapt to the new examples, «forgetting» the previous ones
  - We don't even need a drift detector, SGD will adapt quickly
  - What if we don't want to forget? (DCL module)



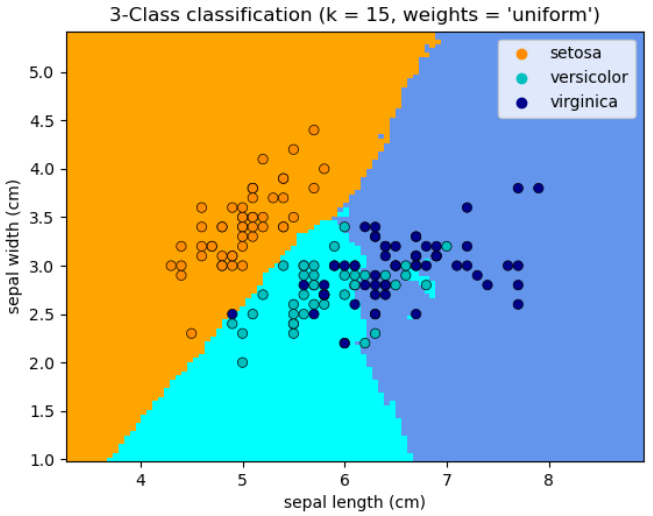
Adapting Offline Methods

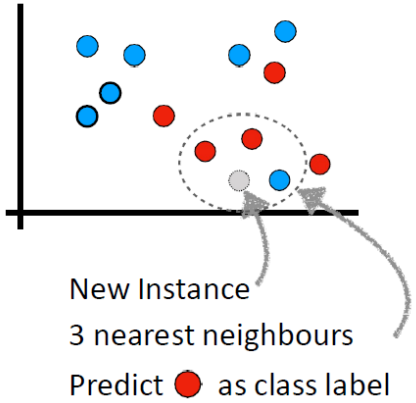
kNN -> online kNN

Decision Tree -> Hoeffding Tree

kNN – k Nearest Neighbors

- Non-parametric distance-based classifier
- **MODEL**:
  - Store samples from the dataset
  - Compute distances between old examples and new example
  - The output is the average of the k closest examples
    - Possibly weighted by distance
    - For classification use a majority voting
- **Hyperparameters**:
  - k: how many neighbors to use
  - Distance metric:
    - $d(a,b) = \sqrt{\sum_{i=1}^{m}(a_i - b_i)^2}$
    - Needs a good distance metric





kNN – offline k Nearest Neighbors

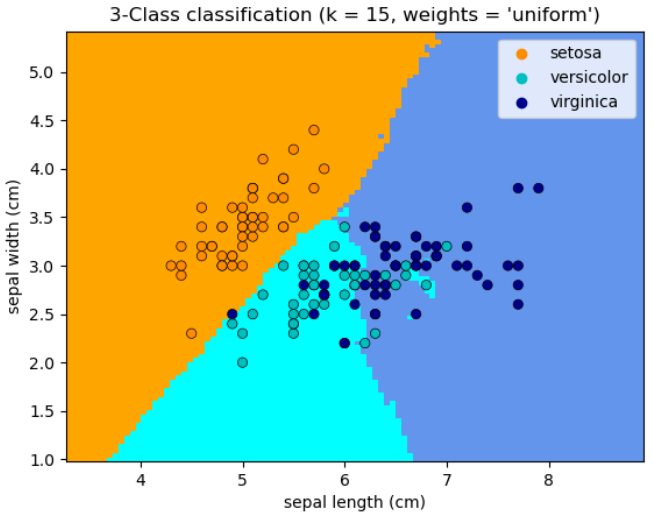
TRAIN:

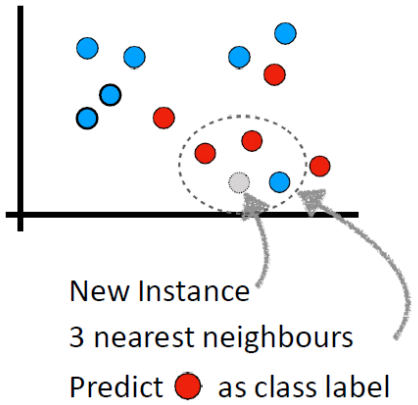
- save the entire dataset

INFERENCE:

- compute distances and find k closest examples
- Use neighbors to compute output

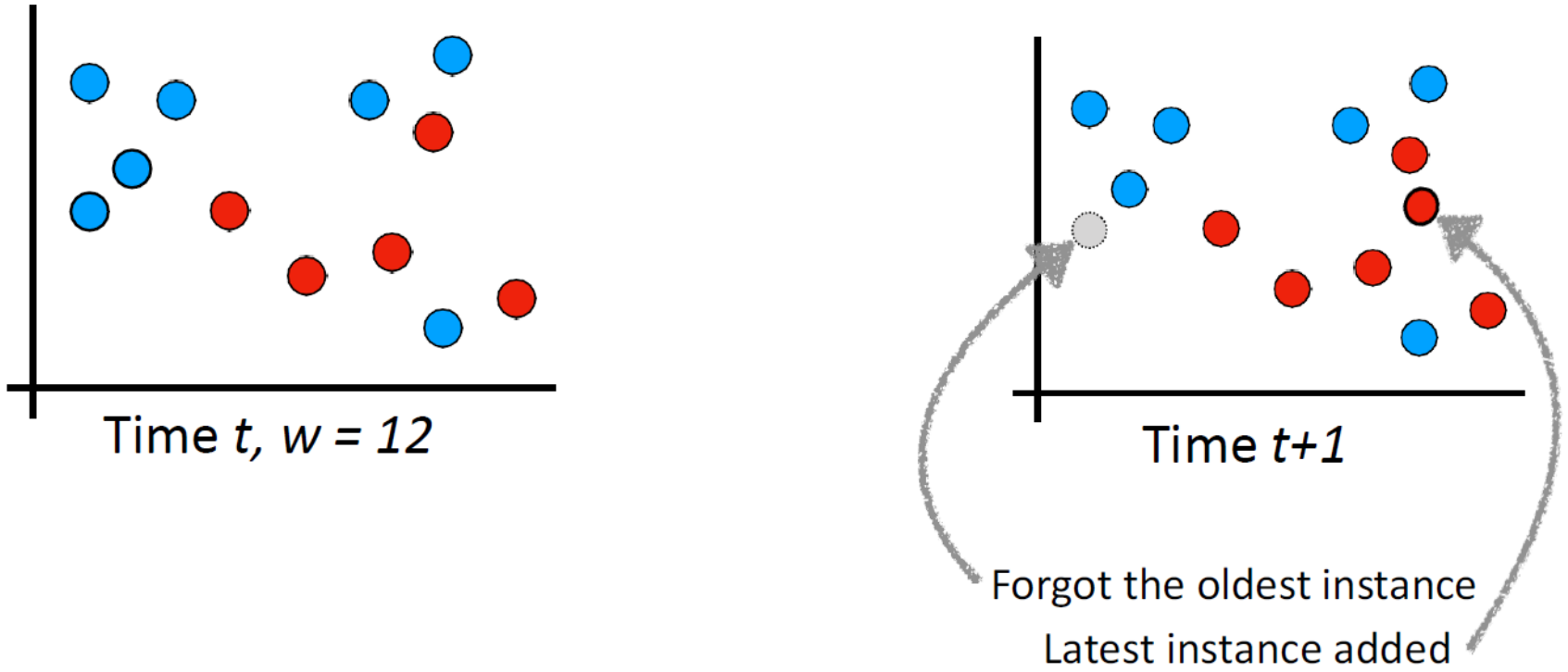
PROBLEM: The algorithm is designed for offline training: we cannot save the entire stream





Online kNN

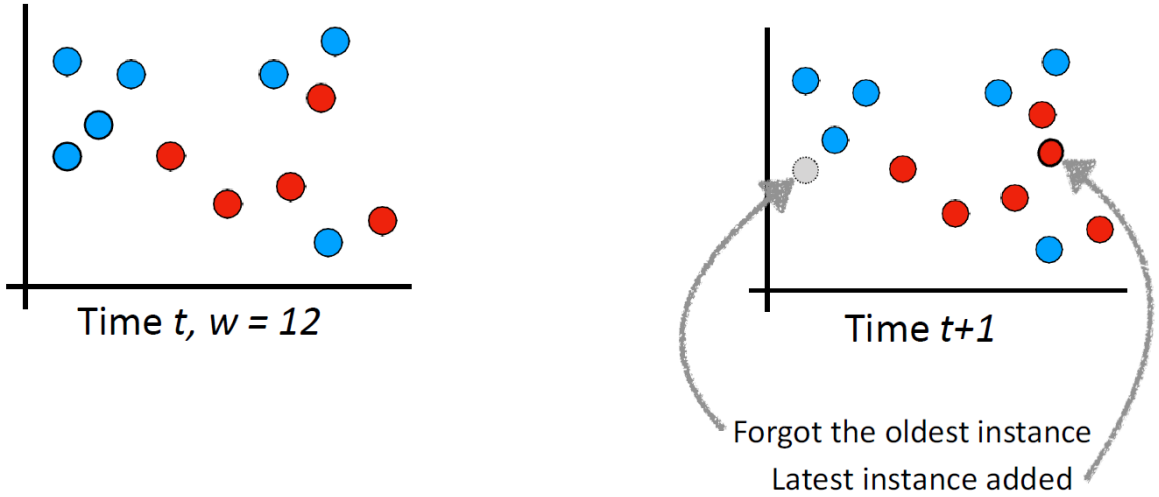
- **SOLUTION**: use a fixed sliding window



Bifet, A., Pfahringer, B., Read, J., & Holmes, G. Efficient data stream classification via probabilistic adaptive windows. 28th ACM symposium on applied computing (2013).

kNN-ADWIN – kNN + Drift Detection

- If a concept drift occurs, with KNN there is the risk that the instances saved into the window belong to the old concept
- Use ADWIN to automatically set the size of the sliding window to save the instances



Bifet, A., Pfahringer, B., Read, J., & Holmes, G. Efficient data stream classification via probabilistic adaptive windows. 28th ACM symposium on applied computing (2013).

Online kNN

- River implementation: https://riverml.xyz/dev/api/neighbors/KNNClassifier/
  - storing a buffer with the `window_size` most recent observations. A brute-force search is used to find the `n_neighbors` nearest observations in the buffer to make a prediction

- You need a good distance metric

Decision Tree

- Fast and interpretable model
- **MODEL**:
  - A tree that represent criteria to split sample
  - Each sample is assigned to one of the **leaves**
  - **Internal nodes** are split criteria
  - A criteria decide which features to use to perform the split and how to split
  - **Classification**: each **leaf** has a corresponding class

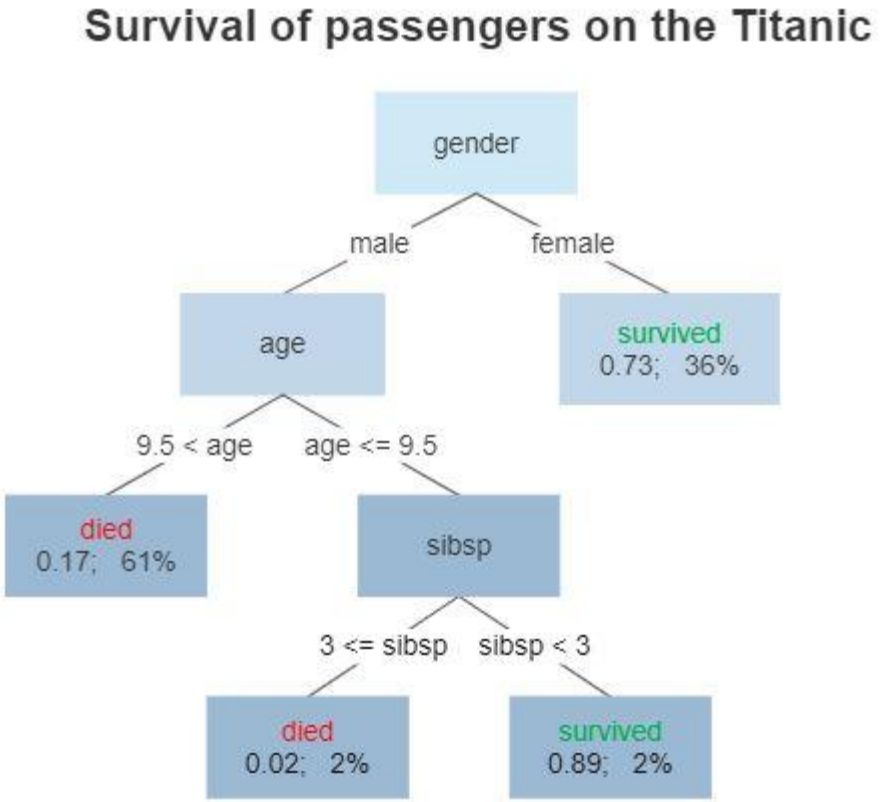


Image: wikimedia

Decision Tree – Inference

INFERENCE

- Until the node is not a leaf
  - Check the feature corresponding to the current internal node
  - Move to the child corresponding to the value of the selected feature
- Return class of the current node (leaf)

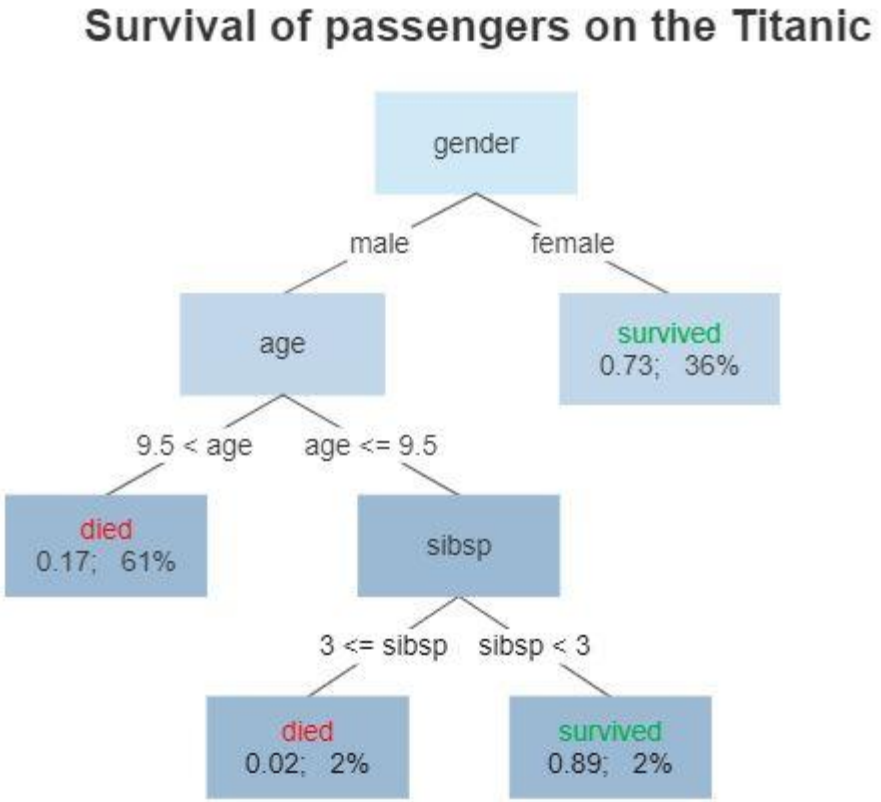


Image: wikimedia

Decision Tree – Offline Training

TRAINING:

- Build the tree
- For each node:
  - Decide if it needs to be split
  - Decide which feature to use for the split
  - Decide how to do the split

- We need to define a split criterion

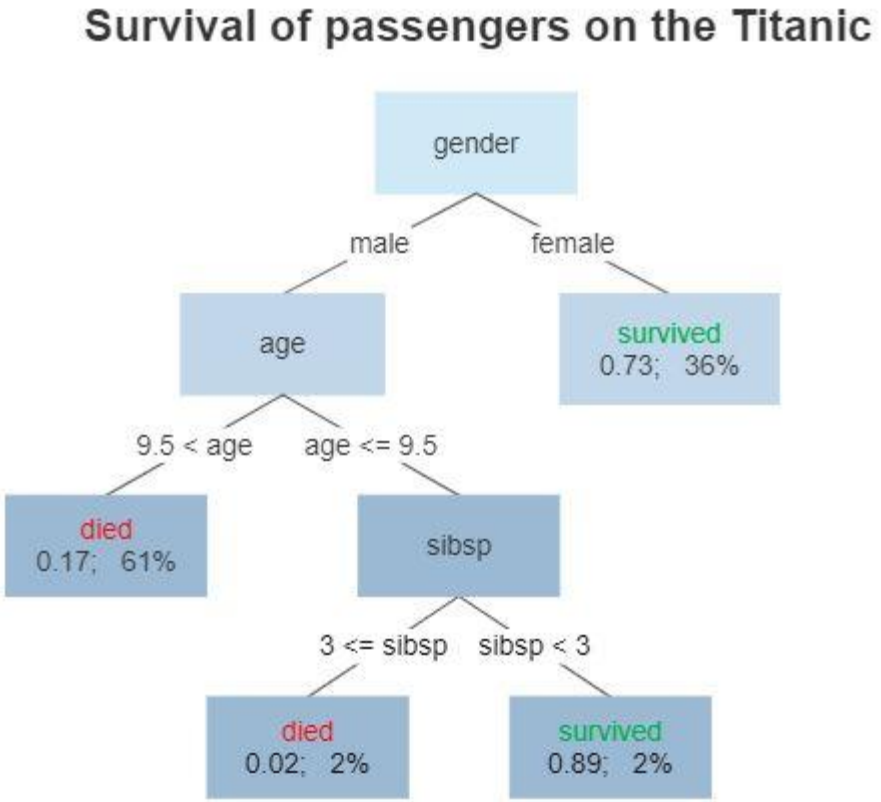


Image: wikimedia

Decision Tree – Offline Training

Greedy recursive algorithm:

- Select only the examples corresponding to the current node
- Find most discriminative attribute Xi
  - Gini index
  - Information Gain (H)
- Split (based on split criterion):
  - Create a new node for each value of Xi
  - Apply the algorithm recursive
- No split:
  - The node is a leaf
  - assigns majority class

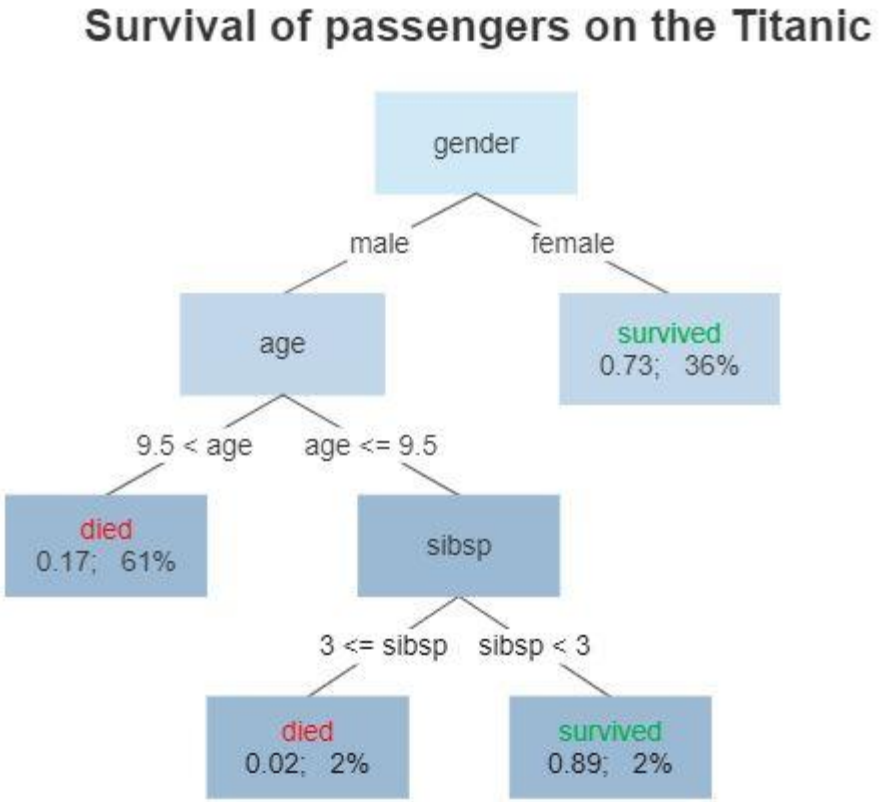


Image: wikimedia

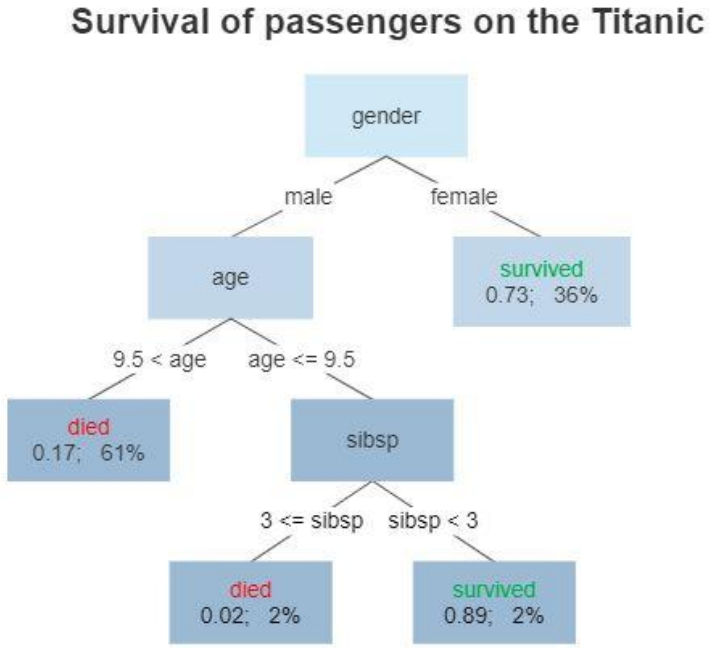
Decision Tree – Information Gain

Information Gain

- $IG(T, a) = H(T) - H(T, a)$
- $H(T, a)$ is the conditional entropy

$$\overbrace{IG(T,a)}^{\text{information gain}} = \overbrace{\mathrm{H}(T)}^{\text{entropy (parent)}} - \overbrace{\mathrm{H}(T \mid a)}^{\text{sum of entropies (children)}}$$

$$H(T, A) = \sum_a H(S_a) \, |S_a| / |S|$$



What is the problem here?

Image: wikimedia

Limitations of the Greedy Choice

$$\overbrace{IG(T,a)}^{\text{information gain}} = \overbrace{\mathrm{H}(T)}^{\text{entropy (parent)}} - \overbrace{\mathrm{H}(T \mid a)}^{\text{sum of entropies (children)}}$$

- **Offline**: we can compute IG and greedily split the node
- **Online**: we can compute the IG of the past data
  - We don't know how much the IG will change with future data
  - If it changes too much, we need to change the tree
  - we don't want to revise the split criterion, so we need to wait for enough data
- **Q**: How much data do we need before deciding the split?

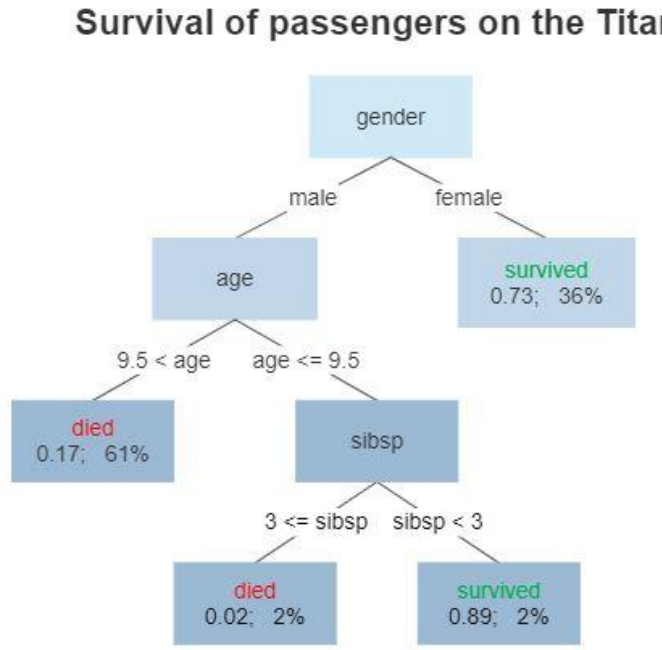


Image: wikimedia

Hoeffding Bound

- Hoeffding's inequality provides an upper bound on the probability that the sum of bounded independent random variables deviates from its expected value by more than a certain amount
- Let $X_1, \ldots, X_n$ be independent random variables such that $a_i \leq X_i \leq b_i$ almost surely. Consider the sum of these random variables $S_n = X_1 + \cdots + X_n$
- The Hoeffding Theorem states that, for all $t > 0$

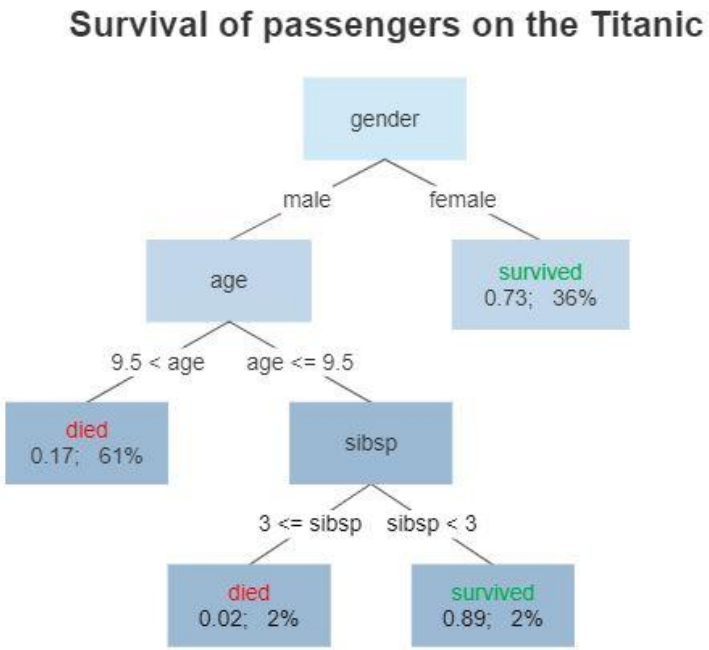
$$\mathrm{P}\left(S_n - \mathrm{E}\left[S_n\right] \geq t\right) \leq \exp\left(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

$$\mathrm{P}\left(\left|S_n - \mathrm{E}\left[S_n\right]\right| \geq t\right) \leq 2\exp\left(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

Given enough samples, we can bound the change in the entropy!
Now we have a criterion to decide when we have enough samples to do the split.

Hoeffding Tree

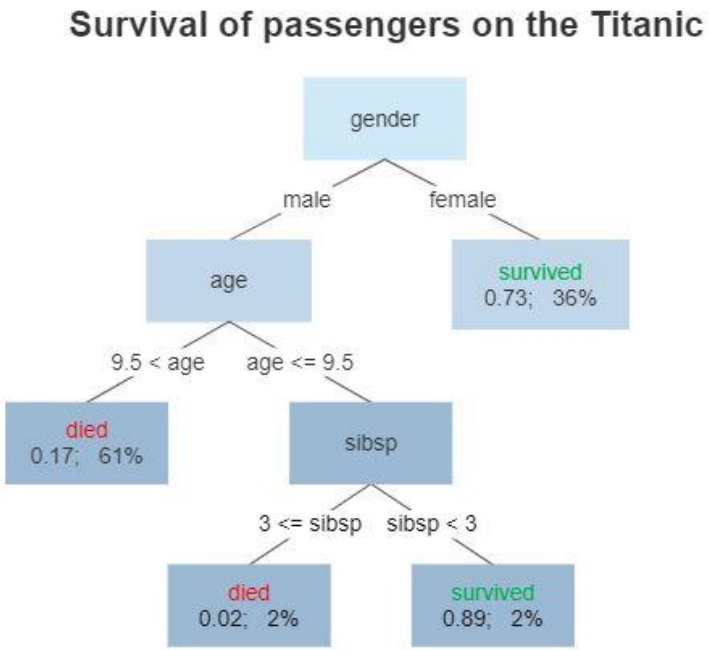
- very fast decision tree algorithm for streaming data
  - Splits decisions based on **Hoeffding bound**
  - wait for enough instances to arrive before splitting
  - with sufficiently large data (and $\delta \to 0$) provably converges to the tree built by a batch learner
- **Confidence interval** for the entropy estimate
  - Confidence interval $\epsilon = \sqrt{\frac{R^2 \ln 1/\delta}{2n}}$
  - R = range of the random variable
  - δ is the desired probability of the estimate not being within ϵ of its expected value,
  - n = number of examples collected at the node



Pedro Domingos and Geoff Hulten. Mining high-speed data streams. 2000

Hoeffding Tree – Online Update

- Every nodes keeps the statistics necessary to compute the split criterion
- For discrete variables a table where:
  - Each row is a triplet $< x_i, v_j, c >$
  - x_i attribute
  - v_j attribute value
  - c counter



Hoeffding Tree – Algorithm

- For each new sample
  - Find its corresponding leaf
  - Update the table
  - Split if the G estimate is close enough (i.e. we have collected enough samples at the node)
- The DT construction is incremental
- Hoeffding bound ensure that the greedy splits must not be revisited

HOEFFDINGTREE($Stream, \delta$)

Input: a stream of labeled examples, confidence parameter δ

let *HT be a tree with a single leaf (root)*
init counts n_{ijk} at root
for each example (x, y) in *Stream*
    do HTGROW($(x, y), HT, \delta$)

HTGROW($(x, y), HT, \delta$)

sort (x, y) to leaf l using HT
update counts n_{ijk} at leaf l
if examples seen so far at l are not all of the same class
  then
    compute G for each attribute
    if G(best attribute) - G(second best) $> \sqrt{\frac{R^2 \ln 1/\delta}{2n}}$
      then
        split leaf on best attribute
        for each branch
          do start new leaf and initialize counts

Algorithm from MOA book

VFDT - Very Fast Decision Tree

Practical implementation of Hoeffding Tree with a few changes

- **Tie breaking**: When two attributes have similar split gain _G_ , VFDT splits if Hoeffding's bound is lower than a certain threshold parameter τ
  - $G(best) - G\ (second\ best)$ is small (< bound) and $\sqrt{\frac{R^2 \ln 1/\delta}{2n}} < \tau$ -> SPLIT
  - The Hoeffding bound tells you that the G estimates are close to the real value and G are similar
    - you can split because the difference between the two is unlikely to change with more data
- **Speed up**: compute G only every k updates
- **Memory improvement**: deactivate least promising nodes:
  - Least promising = low $p_l \times e_l$ product
  - p_l probability to reach leaf l
  - e_l error in node l
- **Warm start**: to mitigate that the performance can be poor at the beginning and slow to converge

CVFDT – Concept-Adapting VFDT

- **Objective**: keep a DT model that is consistent with a sliding windows of w samples
- It needs to add, remove, and forget instances
- **CVFDTGROW:** process an example updating counts of the nodes traversed (same as VFDT)
  - Unlike VFDT, you also need to keep and update the table for internal nodes
- **CHECK SPLIT VALIDITY**
  - check whether the chosen splits are still valid
    - **IF OPTIMAL SPLIT ARE CHANGED: creates an alternate subtree**
  - Periodically, **check whether the alternate branch is performing better** than the original branch tree
    - TRUE -> **replaces the original branch**
    - FALSE -> removes the alternate branch.
- We lose the theoretical guarantees of VFDT
- Lots of additional hyperparameters

HAT – Hoeffding Adaptive Tree

- Hoeffding Tree + ADWIN for concept drift detection

Differences with CVFDT:

- Create a new tree as soon as change is detected
- Switch to the new tree as soon as it becomes better than the old one

- CVFDT requires many hyperparameters related to the expected distance between drifts
- HAT adapts to the scale of time change in the data, rather than relying on the a priori guesses (thanks to ADWIN).
- River: https://riverml.xyz/0.14.0/api/tree/HoeffdingAdaptiveTreeClassifier/

Model Selection – CASH Problem

- **CASH problem**: Combined Algorithm Selection and Hyperparameter.
- **AutoML** aims to automate the data mining pipeline:
- Data cleaning
- Feature engineering
- Algorithm selection
- Hyperparameters tuning

Different implementations with different search spaces and hyperparameter optimizations:

- Auto Weka 2.0
- Autosklearn
- TPOT
- GAMA
- H2O

CASH problem with OML

CASH solution does not consider the adaptation of parameters in an evolving data stream with prequential evaluation

Actual applications to a streaming scenario:

- Train AutoML only the first portion of the data stream
- Retrain AutoML from scratch after a concept drift
- Computational expensive
- Large number of parallel trainings

EvoAutoML

- naturally adapts the population of algorithms and configurations.
- avoids expensive retraining.
- addresses the Online CASH problem by finding the joint algorithm combination and hyperparameter setting that minimizes a predefined loss over a stream of data.

Considers:

- Pipeline structure
- Algorithms
- Configuration space
- Predictions by majority voting

C. Kulbach, J. Montiel, M. Bahri, M. Heyden, & A. Bifet. Evolution-Based Online Automated Machine Learning. PAKDD, 2022

Take-home Messages

- OML methods have a restricted set of available operations
  - Some are naturally online (NB, SGD)
  - Some resort to approximated solutions (kNN, DT)
- You need to be aware of:
  - First phase of learning -> pretrain if possible
  - The limitations of the approximations (if any)
  - Whether the method can deal with concept drift

References

- Streaming Data Analytics Course - Emanuele Della Valle and Alessio Bernardo @ POLIMI
- MOA Book