Machine Learning and AI

- Methods and Algorithms -

Personnal Notes
François Bouvier d'Yvoire

CentraleSupélec & Imperial College

Contents

Todo list

Intro

This document will use the following classification for the machine learning algorithms. However their might be some changes. For exemple, some of them will be part of the commons algorithms and not from their real class.

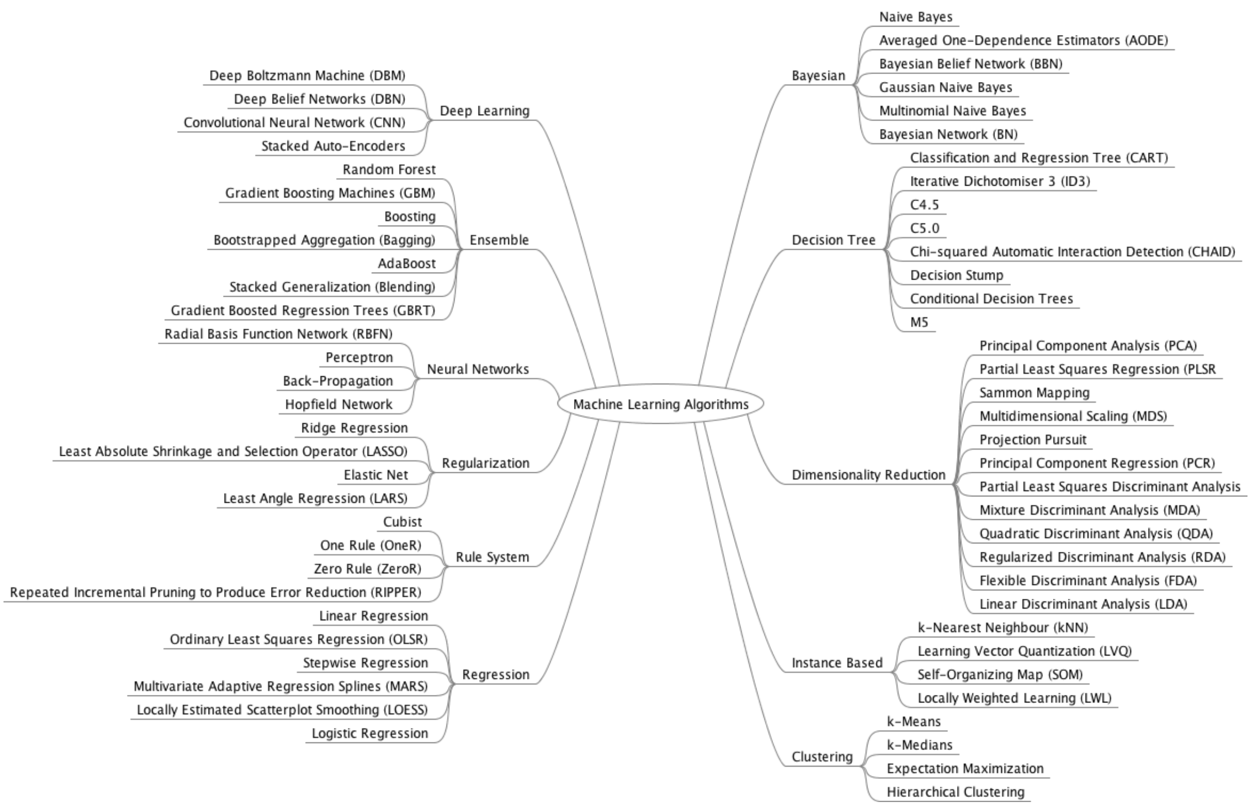

Figure 1 – Simple graph for algorithms classification in ML

Chapter 1

Common Machine Learning algorithms

This chapter is dedicated to the most common ML algorithms, a major part of the notes come from the mml-books.com

Add bibtex reference

Find better paragraph layout

1.1 Linear Regression

1.1.1 Maximum Likelihood Estimation (MLE)

Closed-Form Solution

In some cases, a closed-form solution exist, which make computation easy (but not necesseraly cheap)

Maximum A Posteriori Estimation (MAP)

1.2 Gradient Descent

1.2.1 Simple Gradient Descent

1.2.2 Gradient Descent with Momentum

1.2.3 Stochastic Gradient Descent

1.3 Model Selection and Validation

1.3.1 Cross-Validation

1.3.2 Marginal Likelihood

1.4 Bayesian Linear Regression

1.4.1 Mean and Variance

1.4.2 Sample function

Chapter 2

Reinforcement Learning

2.1 Markov Reward and Decision Process

2.1.1 State Value Function Closed-form

For a Markov Reward Process $(\mathcal{S},\mathcal{P},\mathcal{R},\gamma)$, defining the Return R_t and the State Value Function $\mathsf{v}(s) = \mathbb{E}[\mathsf{R}_t S_t = s]$
Then we have, in a vector form :

$$\mathbf{v} = (\mathbb{1} - \gamma\mathcal{P})^{-1}\mathcal{R}$$

Unfortunately, Matrix inversion in costly, so this is only feasible in small Markov Reward Process

2.1.2 Iterative Policy Evaluation Algorithm

2.2 Dynamic Programming in RL

2.2.1 Policy Iteration Algorithm

2.2.2 Value Iteration Algorithm

2.2.3 Assynchronous Backup in RL

Prioritised Sweeping

Real-time Dynamic Programming

2.2.4 Properties and drawbacks of Dynamic Programming

2.3 Model-Free Learning

2.3.1 Monte-Carlo Algorithms

(First Visit) Monte-Carlo Policy Evaluation

Every Visit Monte-Carlo Policy Evaluation

Add "you cannot backup death" explanations

Batch vs Online Monte-Carlo

Incremental Monte-Carlo Update

Runing Mean for Non-Stationnary World

2.3.2 Monte-Carlo Control Algorithms

Monte-Carlo Policy Improvement

Greedy Policy Improvement over State Value Function

Greed Policy Improvement over State-Action Value Function

Exploring Starts Problem

Don't forget Starting to explore

On Policy Soft Control

On-Policy ϵ-greedy first-visit Monte-Carlo control Algorithm

Monte-Carlo Batch Learning to Control

Monte-Carlo Iterative Learning to Control

2.3.3 Temporal Difference Learning

Temporal Difference Value Function Estimation Algorithm

Add Comparison between MC and TD learning

2.3.4 Temporal Difference Learning Control Algorithm

SARSA - On Policy learning Temporal Difference Control: Here is the Sarsa Algorithm :

Data: State $\mathcal{S}$, Action $\mathcal{A}$, Reward $\mathcal{R}$ and Discount γ
Result: The optimal Q(S, A) State-Action Value Function and a greedy policy w.r.t Q
Initialise Q(s, a) $\forall a, s$ with Q(terminal state, a) = 0 ;
while Convergence condition (number of epoch, $\Delta \leq$ threshold, ...) **do**
    Initialise a state S;
    Choose action A from S with ϵ-greedy policy derived from Q;
    while S is not a terminal State **do**
        Take action A, observe reward R and next state S';
        Choose action A' from S' with ϵ-greedy policy derived from Q;
        Update $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$;
        $S \leftarrow S', A \leftarrow A'$
    end
end
Return Q and π the derived policy

Algorithm 1: SARSA algorithm with ϵ-greedy policy

Theorem 1
Convergence of Sarsa
$Q(s,a) \to Q^{\infty}(s,a)$ under :

- *GLIE (Greedy in the Limite with infinite exploration), which mean every state is visited infinitely many times and that the policy converge toward a greedy-policy (ex: ϵ-greedy with $\epsilon \to 0$).*
- *Robbins-Monroe sequence of step-sizes α_t : which imply $\sum \alpha_t$ diverge and $\sum \alpha_t^2$ converge.*

Remark 1
the ϵ-greedy policy can be replaced by any policy derived from Q. (Because Q is the one updated by the algorithm)

SARSA-Lambda

Hindsight Experience Replay

Q-Learning: Off-Policy Temporal Difference Learning

2.4 Reinforcement Learning with Function Approximation

2.4.1 Exemple of features

Coarse Coding

Tile Coding

Radial-Basis Function

Deep Learning

2.4.2 Monte-Carlo with Value Function Approximation

2.4.3 Temporal Difference Learning with Value Function Approximation

2.4.4 Q-Learning with FA

2.4.5 subsection name

2.5 Deep Learning Reinforcement Learning

2.5.1 Experience Replay

2.5.2 Target Network

2.5.3 Clipping of Rewards

2.5.4 Skipping of Frames

Chapter 3

Dimensionality Reduction and Feature Extraction

3.1 Principal Component Analysis

The objective of PCA is to find a set of features, via linear projections, that maximise the variance of the sample data. We need to decide how many dimension d we want to keep.

3.1.1 Simple algorithm

Data: *Vectors x_i of **centered** data, number F features and n sample. Dimension d of reduction.*
Result: *Y of size $d \times n$*
Compute the product matrix of centered data : $\boldsymbol{XX}^\top$;
Compute the Eigen Analysis $XX^\top = V\Lambda V^\top$;
Order the Eigen Value by descending value, and permute Column of V correspondly;
Compute the eigenvectors : $U = XV\Lambda^{-1/2}$;
Keep specific number of first components : U_d the d first d column of U ;
Compute the new features vectors : $Y = U_d^\top X$

Algorithm 2: Simple PCA Algorithm

3.1.2 Whitening PCA

The feature given by the PCA algorithm are un-correlated, but the variance in each dimension are not the same (in fact this are the eigen value of $XX^\top$). We can whitening the features (or *"sphering" them) by making the covariance matrix equal to Identity.*

Data: *Vectors x_i of **centered** data, number F features and n sample. Dimension d of reduction.*
Result: *Y of size $d \times n$ with Identity Covariance Matrix*
Compute the PCA of X and keep U and Λ;
Compute the Eigen Analysis $XX^\top = V\Lambda V^\top$;
Compute the whitened features vectors : $Y = U_d \Lambda^{-1/2} X = (XV\Lambda^{-1})_d X$

Algorithm 3: Whitened PCA Algorithm

3.1.3 Kernel PCA

Sometimes we want to compute non-linear features extractions. We use the kernel method to compute this. For a dataset $X = (x_i)_{1\dots n}$ we know only the kernel matrix $K = [\phi(x_i)\phi(x_j)^\top]_{(1\dots n)^2} = X^\phi X^{\phi\top}$ with ϕ the non-linear mapping.

Data: *Vectors x_i of **centered** data, number F features and n sample. Dimension d of reduction. K the kernel matrix*
Result: *Y of size $d \times n$*
Compute the Eigen Analysis $K = \mathbf{X}^\phi \mathbf{X}^{\phi\top} = V\Lambda V^\top$;
Order the Eigen Value by descending value, and permute Column of V correspondly;
Keep specific number of first components : V_d the d first d column of V ;
Compute the vector $g(x_t) = [k(x_i, x_t)]_{1..n}]$;
Compute the $E = \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ matrix ;
Compute the new features vectors : $y_t = \Lambda^{-1/2} V^\top (I - E)\left(g(x_t) - \frac{1}{n} K\mathbf{1}\right)$

Algorithm 4: Kernel PCA Algorithm

Chapter 4

Argumentation Framework

This chapter are notes from the Imperial Course Machine Arguing from Francesca Toni. add ref

introduction *Argument Framework are a field in AI which provide way of evaluate any debate problem. It is useful to resolve conflict, to explain decision or to deal with incomplete information.*

4.1 Abstract Argumentation

4.1.1 Simple AA

Definition 1
*an **AA framework** is a set* Args *of arguments and a binary relation* attacks. $(\alpha, \beta) \in$ attacks *means α attacks β.*

Semantics in AA *In order to define a "winning" set of argument, we need to provide semantics over the the framework. This is like recipes which determine good set of arguments.*

Definition 2

- ***conflict-free***
- ***admissible:*** *c-f and attacks each attacking argument.*
- ***preferred:*** *maximally admissible.*
- ***complete:*** *admissible + contains each argument it defends.*
- ***stable:*** *c-f + attacks each argument not in it.*
- ***grounded:*** *minimally complete.*

- ***sceptically preferred:*** *Intersection of all prefered.*
- ***ideal:*** *maximal admissible and containing all prefered.*

Definition 3
Semi-stable extension: *complete such as $A \cup A^+$ is maximal. A^+ is the set of attacked argument by A.*

add ref to ASPARTIX and CONARG

4.1.2 Algorithms for AA

Computing Grounded extensions *Use the same algorithms as grounded labelling, but only output the IN arguments as the grounded extensions.*
the grounded extensions in unique.

Computing the grounded labelling: *Here is an algorithm to compute a grounded labelling*

```
Data: An AA Framework
Result: The grounded Labelling
Label all unatacked argument with IN ;
while The IN and the OUT are not stable do
    Label OUT the arguments attacks by IN ;
    Label IN the arguments only attacked by OUT;
end
Label the still unlabelled UNDEC;
```

Algorithm 5: Computing the grounded labelling

Computing membership in preferred/grounded/ideal extensions; *In order to compute membership, we use Dispute Tree.*

We compute a dispute tree for an argument, and apply the different semantics which are easier to compute on a tree than on a graph.

add algos of computing dispute tree + def of semantics

Computing stable extension: *We use answer set programming with logical program.*

4.1.3 AA with Suppport

Bipolar Abstract Argumentation

*We add a **Support** relation to a classic AA Framework (BAA = $\langle Args, Attacks, Supports\rangle$). There are different semantics :*

Semantics in BAA with deductive support *We can deduce an AA Framework from a BAA with* $Attacks' = Attacks \cup Attacks_{sup} \cup Attacks_{s-med}$ *with*

Definition 4

- $Attacks_{sup}$ *is* ***the supported attacks*** $\implies$ α *attacks every argument that its supports attacks (supports of supports are supports)*

- $Attacks_{s-med}$ *is the super mediated attacks* $\implies$ α *attacks every argument whose supports an argument attacked or* $attacked_{sup}$ *by* α

Then, we apply the AA semantics to $\langle Args, Attacks' \rangle$*. Those semantics are the d-X semantics, where X replace every semantic from AA (grounded, complete, etc.)*

QuAD *We focus here one the QuAD (**Quantitative Argument Debate**) which add a numerical strenght to any argument, and give rule for updating strenght regarding the supporters or attackers.*

Add DF-QuAD rules and algorithm

4.1.4 Argument Mining

4.1.5 AA with Preference Probabilist

4.2 Assumption-Based Argumentation

4.2.1 Simple ABA

4.2.2 ABA more DDs

4.2.3 p-acyclic ABA

4.3 ArgGame

Chapter 5

Useful Computation

5.1 Data Centering using Matrix Multiplication

$$\boldsymbol{X} - \boldsymbol{M} = \boldsymbol{X}\left(\boldsymbol{I}_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top\right)$$