Machine Learning and AI

- Methods and Algorithms -

Personnal Notes

François Bouvier d'Yvoire

CentraleSupélec & Imperial College

Current Branch : `master`
Commit : `30c3d795d0cd2b5f0212653f851e037be0df6f96`

Contents

Todo list

Intro

This document is currently under developement and none of the chapter are finish, nor in a good shape.

This Document is a simple survey of the Machine Learning and AI method that I encounter. Most of it are based on lecture notes from my dual MSc between CentraleSupélec and Imperial College. The first goal is to be used as a personnal reminder of what I have already encounter and understood.

The exemple below is a scheme wich classify the machine learning algorithms in a certain way. This document does not follow this classification. All the definition and concept will not be explains, as a lot of ressources are available on those subjects (Computer Sciences benefits from the most open-sourced field in research). Sometimes ressources might provided, but the reader is invited to look himself for the subjects he does not understand well.

Link figure

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

It is intended to introduce different concept used in most Machine Learning Algorithms before moving to more specific algorithms.

1.1 What is Machine Learning ?

Add short intro that explain the goal, the process, etc of Machine Learning

1.2 Graphical Model for Probabilistic Inference

Based on the Probabilistic Inference Course of Marc Deisenroth (Imperial College)

1.2.1 Probabilistic Pipeline

Here is a simple pipeline of the inference process with a model

1.2.2 Probabilistic graphical Model

In order to deal with complex and big probalistic model, we can use different kind of graphs which represent relationships between random variables. We define the random variables as nodes and the probabiistic or functional relationship between variables as edges in the graphs. With them you can :

- Visualize the structure
- Have insights into properties (such as conditional independance)
- Design or Motivate new models

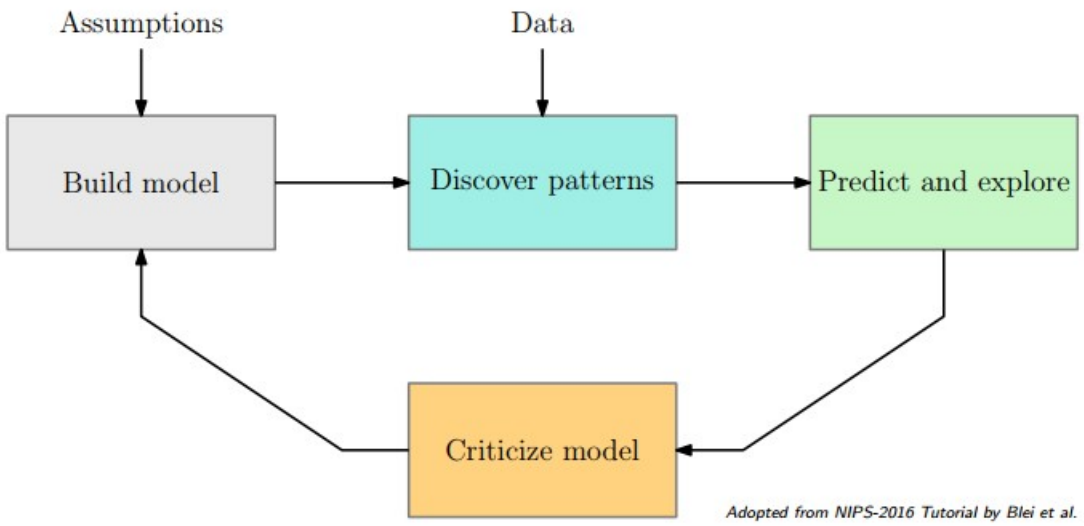

Figure 1.1 – Simple Pipeline of how to build model for inference

- Compute some inference and learning as graphical manipulations

There exists 3 kinds of model : Bayesian networks (directed graphical models), Markov random fields (undirected graphical models) and factor graphs (with nodes which are not random variables)

Bayesian Networks add basic graphs exemple

They are defined by

- Nodes : Random variables
- Shaded nodes: Observed random variables
- Other : Latent Variables
- Edges : (a, b) $\iff$ conditionnal distribution $p(b|a)$

add graphicals model of linear regression

D-Separation: A set A of nodes is d-Separated (conditionnaly independant) from B by C iff all path between A and B are blocked. C is the set of observed variables in the graphical model.

Definition 1
A Path is **Blocked** *if it includes a node such that either :*

- *The arrow on the path meet either head-to-tail or tail-to-tail at the node, and the node is in C*
- *The arrow meet head-to-head at the node, and neither the node nor any of its descendants is in the set C*

Markov Random Fields Defined by : if X and Y are not connected directly, then X and Y are conditionnaly independant given everything else.

Then Joint distributions are the product of cliques in the graphs, and the full joint (of all variables) is the product of the joint of the maximum cliques

$$p(x) = \frac{1}{Z}\Pi_C \psi_C(x_C)$$

Where C describe a maximal clique, x_C all the variable in C, ψ_C the clique potential and Z a normalisation constant.

Check for the conditional independance : Check if when removing the observable variable there is no path between two sets of random variable.

The potentials can be seen as Energy Functions, which mean we can write $\psi_C(x_C) = exp(-E(x_C))$ with E some energy functions, and then $-log(p(x)) = \sum_C E(x_C)$ is the sum of energies of the clique potential.

Add Image Denoising exemple

Factor Graphs (Un)directed graphical models express a global function of several variables as a product of factors over subsets of those variables

Factor graphs make this decomposition explicit by introducing additional nodes for the factors themselves

Add def of factor graph

Converting graphs Directed graph → MRF

- Moralization
- Identiify cliques
- Initialize cliques potential to 1
- Take each conditionnal distribution factor in the directed graph, and multiply it into one of the clique potentials.

MRF ← Factor Graph

- Take variables nodes from MRF
- Create factor nodes corresponding to the Maximal cliques

- the factor are set to be equal to corresponding clique potentials
- Add appropriate link (multiple Factor graphs may correspond to the same MRF)

Removing cycle Conversion make possible to remove cycle in bayesian network.

Lambda message propagation in bayesian network see `https://emtiyaz.github.io/pcml15/belief_propagation.pdf`

Sum-Product Algorithm for Factor Graphs On Factor graphs the inference task is finding the marginal posterior distributions. This can be done easily with Local Message Passing between nodes and factors. Repeated sending message until convergence give use the marginal posteriors.
There is two kind of message :

- Variable to Factor Message $\mu_{x_m \to f_s}(x_m) = \prod_m \mu_{f_l \to x_m}(x_m)$
  This is the product of all incomming message allong all other link.
  A variable node can send message to a factor only when it has received message from all other link
- Factor to variable message : $\sum_{x_1} ... \sum_{x_M} f_s(x, x_1, ..., x_M) \prod_m \mu_{x_m \to f_s}(x_m)$
  Take the product of incomming message
  Multiply by the factor
  Marginalize over all variable of the incomming messages.

Initialisation :

- If leaf is Variable node then $\mu_{x \to f}(x) = 1$
- If leaf is Factor node then $\mu_{f \to x}(x) = f(x)$

1.3 Linear Regression

The Linear regression problem correspond to find a linear mapping $f(x)$ based on noisy observation $y = f(x) + \epsilon$, where ϵ is a noise. Finding the regression function require :

- Choice of parameters (function classes, dimension)
- Choice of probabilistic model (Loss function, ...)
- Avoiding under and overfitting
- Modeling uncertainty on data

Definition 2

- *$p(x,y)$ is te joint distribution*
- *$p(x)$ and $p(y)$ are the marginal distributions*
- *$p(y|x)$ is the conditional distribution of y given x*
- *in the context of regression, $p(y|x)$ is called likelihood, $p(x|y)$ the posterior, $p(x)$ the prior and $p(y)$ the marginal likelihood or evidence.*
- *they are related by the Bayes' Theorem : $p(x|y) = \frac{p(y|x)p(x)}{p(y)}$*

1.3.1 Conjugacy

In order to compute the posterior, we require some calculations which imply the prior, and can be intractable as a closed form. But given a likelihood, it can exist prior which give closed-form solution for the posterior. This is the principle of conjugacy (between the likelihood and the prior)

1.3.2 Maximum Likelihood Estimation (MLE)

In order to obtain the parameters, we need to choose a quantity to optimize. The first and most common idea is to choose the Likelihood as a cost function to optimize. The seems natural because the likelihood represent the probability given the X variables to obtain the observed variables Y and we want this to be maximum on the known data (which are the training set).

$$\theta^* = \arg\max_\theta p(y|X,\theta)$$

To find the desired parameters that maximize the likelihood, we typically do gradient ascent (or gradient descent on the negative likelihood). For numerical reasons, we apply the log-transformation to the probleme and minimize the negative log-likelihood.

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

MLE with Features

1.3.3 Overfitting

1.3.4 Regularization

1.3.5 Maximum A Posteriori Estimation (MAP)

MAP for Linear Regression

1.4 Gradient Descent

1.4.1 Simple Gradient Descent

1.4.2 Gradient Descent with Momentum

1.4.3 Stochastic Gradient Descent

1.5 Model Selection and Validation

1.5.1 Cross-Validation

1.5.2 Bayesian Model Selection

Bayes Factor

Fully Basyesian Treatment

1.5.3 Marginal Likelihood

1.6 Bayesian Linear Regression

1.6.1 Mean and Variance

1.6.2 Sample function

1.7 Features Extraction

Simply explain and redirect to relevant chapter (dim reduc, computer vision, ...)

1.8 Support Vector Machine

1.8.1 Linear Separating Hyperplane

Lagrangian duality

Karush-Kuhn-Tucker Condition for Optimality

1.8.2 SVM dual problem

1.8.3 Support Vector Regression

Chapter 2

Regression

2.1 Logistic Regression

Add Logistic regression with the content of Probabilistic Inference

2.1.1 Logistic Sigmoid

For a binary classification problem – with two class C_1 and C_2 – We want to compute $p(C_1|x)$ (or equivalently $p(C_2|x)$) the posterior in order to classify an input x.
If we write $a = \log \frac{p(C_1|x)}{p(C_2|x)}$, then the sigmoid of a is the desired posterior.

$$\sigma(a) = \frac{1}{1+\exp(-a)} = p(C_1|x)$$

This can be generalized to multiclass thanks to SoftMax function, Boltzmann distribution and/or normalized exponential.

2.1.2 Implicit Modeling Assumptions

In case of multiclass, this is perfectly tractable assuming some conditions.

Here we assume : $p(x|C_k) = N(x|\mu_k, \Sigma)$, every likelihood are gaussian and sharing a common covariance matrix Σ.

Then a can be rewritten with a simple and tractable form (depending on k). For k = 2, we have :

$$p(C_1|x) = \sigma(a) = \sigma(\theta^\top x + \theta_0)$$

with $\theta = \Sigma^{-1}(\mu_1 - \mu_2)$ and $\theta_0 = \frac{1}{2}(\mu_2 - \mu_1)^\top \Sigma^{-1}(\mu_2 - \mu_1) + \log \frac{p(C_1)}{p(C_2)}$
Here the argument of the sigmoid is linear in x which mean the Decision boundary is a linear function of x.

notes : If the Covariance are not shared, then it's still tractable, and we get quadratic decision boundaries.

2.1.3 Model Specification

Sometimes we want to put assumptions on the posterior model. For a binary classifier, with output $y \in \{0,1\}$ we might want a bernoulli posterior $p(y|x,\theta) = \text{Ber}(y|\mu(x))$. Then the bernoulli parameters need to be a parameters of x. The idea is to have a linear model in x such as $\theta^\top x$. Combining this with the logistic function gives :

$$p(y|x,\theta) = \text{Ber}(y|\sigma(\theta^\top x))$$

2.1.4 Model Fitting

Now that we have some model assumption, lets look to the parameters θ estimations. We can do Maximum Likelihood Estimation (MLE) or Maximum-A-Posteriori (MAP). Writting down the likelihood gives :

$$p(y|X,\theta) = \prod_{n=1}^{N} \mu_n^{y_n}(1-\mu)^{1-y_n} \quad \text{with } \mu_n = \sigma(\theta^\top x_n)$$

which lead to the following negative log-likelihood :

$$\text{NLL} = -\sum_{n=1}^{N} y_n \log \mu_n + (1-y_n)\log(1-\mu_n)$$

This can be differentiate (easily with chain rule) :

$$\frac{dNLL}{d\theta} = -\sum_{1}^{N} (\frac{y_n}{\mu_n} - \frac{1-y_n}{1-\mu_n})\frac{d\mu_n}{d\theta}$$

with for all n

$$\frac{d\mu_n}{d\theta} = \sigma(\theta^\top x_n)(1-\sigma(\theta^\top x_n))x_n^\top$$

finally :

$$\frac{dNLL}{d\theta} = (\mu - y)^\top X$$

This is not tractable with a closed-form solution, but Gradient descent methods can be used as it exists a Unique global optimum !

Comments on MLE

- In the case of a linearly separable classes, then the decision boundary between the 2 classes is not unique (any separation hyperplan works) and the likelihood will tend to infinity
- Overfitting is a again a problem when we work with features $\phi(x)$ instead of x
This can be adressed with MAP (to some extents at least)

Notes on MLE : as a recall, the Log-Posterior is $\log p(\theta|X,y) = \log p(y|X,\theta) + \log p(\theta) + const$. There is no closed-form solution for θ_{MAP} but still numerical maximization is possible. With a gaussian prior on the parameters ($p(\theta) \sim \mathcal{N}(\mu,\Sigma)$) the log prior is : $\frac{1}{2}(\theta-\mu)^\top \Sigma^{-1}(\theta-\mu) + \text{cte}$ which lead to the following gradient $\frac{d\log p(\theta)}{d\theta} = \Sigma^{-1}(\theta-\mu)$

Predictive Labels: more than just a predicted label, the output is a parameter of a bernoulli variable, which mean it gives the probability of the input to be in that class.

Add graphical exemples

2.1.5 Bayesian Logistic Regression

An other way to estimates the parameters is to do bayesian inference on the parameters. Now we are given a datased $\mathcal{D}$ of N exemples, and we want to compute a posterior distribution on the parameters θ.
We set Gaussian Prior $p(\theta) \sim \mathcal{N}(\theta|0,\Sigma_0)$ and we get

$$p(\theta|\mathcal{D}) = \frac{p(\theta)p(y|\theta,X)}{p(y|X)} = \frac{\mathcal{N}(\theta|0,\Sigma_0)\prod^N \text{Ber}(\sigma(x_n^\top\theta)}{\int \mathcal{N}(\theta|0,\Sigma_0)\prod^N \text{Ber}(\sigma(x_n^\top\theta))d\theta}$$

This is a nice equation, but there is no analytic solution, therefore approximation are necessary.

2.1.6 Laplace Approximation

One of the simplest approximation is the Laplace approximation. The objective is to approximate an unknown distribution with a Gaussian one.
It does not seem a very good method as most of the distribution that can be encountered are not gaussian, but most of the time a gaussian is a good approximation in the domain of work. That said, the Laplace approximation will give you the best Gaussiann even if the true distribution has a completely different shape.

The idea is to use the Taylor expension of the negative log of the distribution arround the MAP estimate (named x^*).

We write $p(x) \propto \exp(-E(x)) = \tilde{p}(x)$ then, we have $-\log(\tilde{p}) \simeq E(x^*) + \frac{1}{2}(x-x^*)^\top H(x^*)(x-x^*)$ (because Jacobian is 0 as x^* is a stationnary mode.) which lead to the following $\tilde{p}(x) \simeq \exp(-E(x^*))\exp(-\frac{1}{2}(x-x^*)^\top H(x^*)(x-x^*) \propto \mathcal{N}(x|x^*,H^{-1}) = q(x)$

Chapter 3

Dimensionality Reduction and Feature Extraction

The objective of this chapter is to explain dimensionality Reduction, Feature extraction and methods to achieve this. Those method can be then applied to different kind of data and model, in any domain.

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 : $\boldsymbol{X}\boldsymbol{X}^\top$;
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 1: 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 2: 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 $K = \boldsymbol{X}^\phi \boldsymbol{X}^{\phi^\top} = V\Lambda V^\top$;
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 3: Kernel PCA Algorithm

Chapter 4

Bayesian Algorithms

Bayesian Algorithms aims to produce not just function (for classification or regression) but distribution over function, which achieve to get the uncertainty of our model according to the uncertainty of the data.

4.1 Gaussian Process

This section is made with a great inspiration from the Probabilistic Inference from Marc Deisenroth (Imperial College London)

4.1.1 Problem setting

For a set of observations $y_i = f(x_i) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$, we want to fine a distribution over **functions** $p(f)$ that explains the data. It's not exactly the same as a linear regression problem because we do not look for only one function.
This powerfull methods is used in a lot of different problematic thanks to its robustness and known properties (in comparison to deep learning methods).

- Reinforcement Learning and Robotics
- Bayesian optimization
- Geo-statistics
- Sensor networks
- Time-series modelling and forecasting
- High-energy physics
- Medical application

Formally a Gaussian Process is a multivariate Gaussian distribution with infinite variables (countable or even uncountable).

4.1.2 Definition

A Gaussian process is defined by a mean function $m(\dot{)}$ and a covariance function (=kernel) $k(\dot{,}\dot{)}$.

The mean function represents the average of all the function.
the Covariance function allows us to compute covariance between any two functions. Notes that the function are unknown, and only those correlations are fully known.

4.1.3 Gaussian Process Inference

Considering X training inputs and y training target, the Bayes' theorem in the case of Gaussian Process become

$$p(f|X,y) = \frac{p(y|f,X)p(f)}{p(y|X)}$$

Which gives us a Likelihood $p(y|f,X) = \mathcal{N}\left(f(X), \sigma^2 \boldsymbol{I}\right)$,
a Marginal Likelihood $p(y|X) = \int p(y|f,X)p(f|X)df$
and a Posterior $p(f|y,X) = \mathbf{GP}(m_{post}, k_{post})$
How can we manage to work with the distribution over function, and then infinite dimension for calculus, etc. ? This is possible because each time you consider only finite sample, computing the joint distribution boils down to work on finite-dimensional multivariate Gaussian distributions
Then we can use the gaussian properties (including gaussian prior as conjugate, ...) to make prediction :
(We define X_*, f_*, etc as the test data and predicted function)

$$p(f(x_*)|X,y,x_*) = \mathcal{N}(m_{post}(x_*), k_{post}(x_*,x_*))$$

This can also be seen as

$$p(f(x_*)|X,y,x_*) = \mathcal{N}(\mathrm{E}[f_*|X,y,X_*], V[f_*|X,y,X_*])$$

with $\mathrm{E}[f_*|X,y,X_*]$ = prior mean + "Kalman gain" * error $= m(X_*) + k(X_*,X)(\boldsymbol{K} + \sigma_n^2\boldsymbol{I})^{-1}(y - m(X))$ and $V[f_*|X,y,X_*] = k(X_*,X_*) - k(X_*,X)(\boldsymbol{K} + \sigma_n^2\boldsymbol{I})^{-1})k(X,X_*)$

4.1.4 Covariance function / Kernel of Gaussian Process

A Gaussian process is fully specified by mean and kernel. The requirement for the kernel is to be symmetric and positive semi-definite. Therefore sum and product term by terms of Kernel are still valid kernel. This make possible to create various kind of kernel, encoding high-level structural assumptions about the functions, with combining a few known kernel.
The most common kernel are :

- Gaussian Kernel : $k(x_i, x_j) = \sigma_f^2 \exp\left(-\frac{(x_i - x_j)^\top (x_i - x_j)}{l^2}\right)$
- Matérn Kernel : $k(x_i, x_j) = \sigma_f^2 \left(1 + \frac{\sqrt{3}\|x_i - x_j\|}{l}\right) \exp\left(-\frac{\sqrt{3}\|x_i - x_j\|}{l}\right)$
- Periodic Kernel : $k(x_i, x_j) = \sigma_f^2 \exp\left(-\frac{2\sin^2\left(\frac{\kappa(x_i - x_j)}{2\pi}\right)}{l^2}\right)$ which is a gaussian with a conjugate periodic cosinus and sinus.

4.1.5 Hyper-paramaters of a GP

A GP possesses 3 kinds of hyper parameters:

- Mean function
- Covariance
- Likelihood parameters (such as noise variance)

Therefore this imply that hyper-parameters can be optimized and that model selection can be made.

Marginal likelihood and optimization

4.1.6 Model Selection

4.1.7 Limitation and tips

4.2 Bayesian Optimisation

4.2.1 Machine Learning Meta-Challenges

- Models are more and more complicated (more parameters)
- Non Convex and stochastic optimization have hyper parameters hard to optimize (learning rates, momentum, ...)
- Ther

4.2.2 Search for Good Hyper-parameters

-Require objective function for evaluation (Generalization evaluation, cross-validation, ...)
- Standard search procedures:

- Manual Tuning : Costly and mostly ineficient
- Grid Search : Costly but in fine works

- Random search : Simple, and works surprisingly often well
- Black Magic : Take hyper parameters from other papers/ Researcher/similar problem.
- Bayesian Optimization

4.2.3 Description of Bayesian Optimization

The objective is to optimize our model as a black box which is expensive to evaluate.For this a proxy model which is cheap to evaluate is build using the outcomes of experiments of the black box.
The proxy need to be cheap to evaluate and therefore cheap to optimize.
The standard proxy is of course a Gaussian Process.

Key-step :

- Approximate the model with a proxy function
- Find global optimum of the proxy (we can guarranty that this is global and not only local thanks to the properties of the proxy).
- Evalute the true model at this optimum
- use the new value to get a better proxy model and restart until satisfied.

Add link to detailed explanation

4.2.4 Use Uncertainty in global Optimization

The previous pseudo-algorithm are very focused on the global optimum of the mean function of the Gaussian Process. Howerver we need to take into account the Variance if we want to explore the parameters space. Let's imagine a GP with only one datapoint as training set, with value lesser than the mean value of the GP prior. Then the algorithm is stuck at this point and never explore.
That's why using high variance is interresting, with the followong trade-off:

- Exploration : Seek places with high variance/uncertainty
- Exploitation : Seek places with low mean

Write BO as a proper algorithm

Probability of Improvement The idea is to determine the point with the higher probability of improvement (in term of min value of the black-box).

slack variable Avoir continous exploration arround a best value, and force more exploration

Expected Improvement Here we use the expectation of the improvement instead of probability, this tend to have a better exploration.

GP-lower confidence Bound We can simply use the lower confidence bound as the function for next point location, using a factor between mean and variance for the trade-off between Exploration and Exploitation.

4.2.5 Limitation

- Problem of function model for proxy (Kernel choice).
- Limited Scalibility : Curse of dimensionnality

Chapter 5

Ensemble Algorithms

Idea : Aggregate the predictions of a group of predictors (either classifiers or regressors)

In many case the aggregated answer is better than best individual prediction.

add illustration from ML for imaging course

Even weak learner can achieve high accuracy, provided there are un sufficient number and sufficently diverse.

Voting toy exemple ?

Definition 3

- *Homogenous Ensemble Learning Use the same class of ML model (which are Weak Learner)*
- *Heterogenous Ensemble Learning Use different ML model*
- *Sequential : Base Learners are added one at a time, mislabelled examples are up-weighted each time(Ex : Boosting)*
- *Parallel : many independent base learners are trained simultaneously and then combined (Ex : Voting, Bagging, Random Forest)*

Definition 4

Weak Learner *: is defined to be a classifier that is only slightly correlated with the true classification (0.5 for bi-classifier)*

5.1 Bagging : Boostrap Aggregating

Reduce model variance through averaging

Add Bootstrapping 3 steps and Aggregating

5.1.1 Outof-bag error

5.2 Boosting

5.2.1 Principle

Rather than building independent weak learners in parallel and aggregating at end: – build weak learners in serial – but adaptively reweight training data prior to training each new weak learner – in order to give a higher weight to previously misclassified examples

5.2.2 Exemple ?

5.2.3 Several variants

• Adaboost (adaptive boosting) – Originally for classification – Can be used for regression too • Gradient Tree Boosting • XGBoost

Adaboost

Adaboost = Adaptive Boosting

Optimizing Adaboost with early termination

5.3 Decision Trees

Chapter 6

Deep Learning

6.1 Intro

6.1.1 Representation Learning

Representation Learning is the field where you try to learn the representation (features) of input. Then you can do classification or regression on the representative space.

6.1.2 Deep Learning vrsus classical Representation Learning

Add image with where Deep Learning is relative to ML

6.1.3 Supervised vs Unsupervised Learning

Supervised : Learn to map $x \rightarrow y$ with y labels. Ex : Classification, regression, object detection, semantic segmentation, image captionning, etc. Unsupervised : Inferring a function that describes the hidden structured of unlabelled data. EX : Density Estimation (for continuous probability), Clustering, Feature Learning/Representation Learning (as Embedding vectors), Dimensionality Reduction, etc.

6.2 Optimisation : Gradient Descent

6.2.1 Stochastic GD versus Mini-Batch

Classicaly the learning is an optimization problem :

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{1}^{n} l(f_\theta(x_i), y_i)$$

Then we can calculate a gradient

$$\nabla_\theta \hat{L}(\theta) = \frac{1}{n}\sum_{1}^{n} \nabla_\theta l(f_\theta(x_i), y_i)$$

There is different possibility to update the parameters, the stochastic way (SGD) which learn only one by one exemple:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla_\theta l(f_\theta(x_t), y_t)$$

And the mini-batch which average the learning on a mini-batch (b < n the data set size) of examples.

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \frac{1}{b}\sum_{1}^{b} \nabla_\theta l(f_\theta(x_{tb+i}), y_{tb+i})$$

6.2.2 Convergence rate and computational complexity

$$\mathbb{E}(\hat{L}(\theta^{(k)} - \hat{L}(\theta^*))) \leq \frac{\alpha\sigma^2}{2m} + (1-\alpha m)^k (\hat{L}(\theta^{(0)} - \hat{L}(\theta^*)))$$

With conclusion :

- Small step size $\Longrightarrow$ **Slower** initial convergence and Stalls at **more** accurate result
- Large step size $\Longrightarrow$ **Faster** initial convergence and Stalls at **less** accurate result
- Small batch size $\Longrightarrow$ Stalls at **less** accurate result and **Lower** iteration cost
- Small batch size $\Longrightarrow$ Stalls at **more** accurate result and **Higher** iteration cost

6.2.3 Deep Learning is not convex

which mean you don't know if local extrema is global extrema, and this is problematic for gradient descent method, which find local extrema.

Add ref to convexity

6.3 Maximum Likelihood estimation

Kullback-Leibler divergence

cross entropy

Log-likelihood

L_2 as MLE

Logistic Regression

6.4 Regularization

In order to avoid certain problem (overfiting, saturation...) regularization can be added to the loss function, (can be named weight decay). This works in restricting the hypothesis class, and find explanation in different way, such as Ockahm Razor.

6.4.1 Generality

General form :

$$\min_{\theta} \hat{L}(\theta) + \beta R(\theta)$$

- Soft constraint
- Equivalent to hard constraint (Lagrangian)
- equivalent to the Maximum a posteriori interpretation (see Common ML)

The regularization reduce the capacity of the model, so a trade-off exist between large capacity but under-regularisation (overfitting or not convergence) and small capacity with over-regularization

Bayesian view todofill MAP equation

6.4.2 Geometric interpretation

6.4.3 L1 Regularization

6.4.4 Noisy input

We can show that add an i.i.d noise $\epsilon \sim N(0, \beta \boldsymbol{I})$ is equivalent to weight decay.

6.4.5 Data Augmentation

6.4.6 Early stopping

Dropout

6.4.7 Covariate shift

6.4.8 Batch normalization

6.4.9 Input pre-processing

- Zero-centered data
- normalized data
- Decorrelated data
- Whitened data

6.5 Deep Feed-forward network

6.5.1 Perceptron

Activation function

- tanh
- sigmoid
- ReLU -> Logistic regression

Multi-layer perceptron Single layer is not enough

XOR Problem

Theorem 1
A perceptron with one hidden layer of finite width can arbitrarily accurately approximate any continuous function on a compact subset of $\mathbb{R}^n$ *(under mild assumptions on the activation function)*

What is better: Depth or Width ?

Popular output layer types

- 1D Regression
- Binary classification
- nD Regression
- Softmax (multi-class classification)

Backpropagation Use chain rule to calculate all the layer gradient relative to the output gradient.

Differentiable programming

6.6 Deep convolutionnal Neural network

See Imaging chapter first

6.7 Generative Models

Generative models are generally in the unsupervised learning category, as their objectif is to find a model such as $p_{model}(x) \simeq p_{data}(x)$.

Then, with a random noise z as input of our model, which is equivalent to sample from the model distribution, we can generate new data which are similar to the original data.

6.7.1 Sampling from PCA

PCA is a dimensionality reduction algorithm (see eponymous chapter), therefore for a given data distribution we reduce the dimension of features. In the reducted space, approximating the distribution is easier, and using the reverse PCA after sampling from the reducted distribution approximation can draw sample from an approximation of the distribution in the original space.

Chapter 7

Imaging

7.1 Image Classification

7.1.1 Features and Classifier

7.1.2 Feature Extraction - without ML

- Why features ?
- What features should be :
  – Distinctive and discriminative – Local (to enable establishing correspondences) – Invariant to viewpoint changes or transformations (translations/rotations) – Invariant to illumination changes – Efficient to compute – Robust to noise and blurring – Should be hierarchical
- Many options possible : – Intensities – Gradient – Histogram – SIFT – SURF – BRIEF
- Pixel level bad : Not discriminative, Localiser and does not represent local pattern, not invariant to transformation.
- Patch Level : More discriminative, Not necessarily rotation invariant, semi-localised. Use Convolution for patch.
- Image Level : discriminative, Not localised, not necessarily rotation invariant. Get a feature map where each pixel correspond to local pattern

How to make rotation invariant ? → Use Histogram.

Filtering

- Use filter to identify features such as gradient, edge detection.
- Gradient are invariant to absolute illumination.

SIFT

- Scale-space Extrema Detection : For all scale, compute convolution with gaussian (blur at different level) and compute the difference of each scaled guassian, then Detect local extrema of Differences
- Keypoint Localisation :
- Orientation Assignment : Take a small window around the keypoint location.An orientation histogram with 36 bins covering 360 degrees is created. Each pixel votes for an orientation bin, weighted by the gradient magnitude after applying a Gaussian filter with he keypoint scale !). The keypoint will be assigned an orientation, which is the mode of the distribution
- Keypont descriptors.

OHoG (Histogram of Gradient)

SURF (Speeded-Up Robust Features)

BRIEF (Binary Robust Independant Elementary Features)

Local binary Patterns

Haar features

Convolution ineficient → Faster computation via integral images S

7.1.3 with ML

Classification or regression model, many options : Logistic regression, Naives Bayes, K-Nearest neighbors, Support vector Machines (does not deal good with computer vision , Boosting, Decision/Random forsest, Neural network.

Boosting and Decision trees are Ensemble forests.

add bias-variance error explanation in common ML

Add Classical ML pipeline for computer vision (images, features, classification)

why ?

7.2 Segmentation

- Quantify analysis
- Location and extent of object
- Generating 3D models for simulation

7.2.1 Semantic Segmentation

Chapter 8

NLP

Thanks to Lucia Specia Course at Imperial Collage and several ressources :
https://github.com/nyu-dl/NLP_DL_Lecture_Note/blob/master/lecture_note.pdf
https://web.stanford.edu/~jurafsky/slp3/

8.1 What is NLP

Natural language Processing (or Natural Language Understanding sometimes) is the field which try to make machine understand natural human language. Indeed we have developped really complex way of communication thanks to vocaculary, grammar, syntax and semanticDomain a the intersection between AI, Computer Sciences, Linguistics and Cognitive Science.

The goal are to undersand NL, execute requested task and produce back NL as answer. The field generally focus on written language as speech and handwritten require only speech-to-text and OCR to be computed with NLP models.

8.1.1 Linguistic required

Working on NLP require some knowledge of linguistics in order to understand the different problematics and the possible solutions

- a word
- Word Segmentation (tokenization and decompounding)
- Lemmatization
- Morphological analysis (POS Tagging, Suffixes (unhappy-happy), gender, number and tense)
- Grammar (phrase-structure)

- Semantics (Lexical meaning and compositionnal meaning)
- References and relationship

8.1.2 NLP Application

This are different kind of applications of NLP models :

- Sentiment analysis
- Fake News/Deception detection
- Offensive Language detection and categorisation
- Question answering
- Machine translation
- Text summarisation
- Dialog System (Phone assistant)

8.2 How to represent word

In order to use maths to infer anything on word, the first questions that raised is "How to represent words in a mathematical way ?". This can be rephrased as a question about encoding the meaning of a word. The definitions of the meaning of a word have been studied for a long time by philosophy and linguistic, and this can inspire maths a lot.

The notions of lexical semantics (meaning of a word) cover those following subjects : Lemma and Senses, Relationship between words or senses (Propositionnal meaning, Principle of Contrast, Antonyms/reversive), Word similarity and Word relatedness (Co-participating in shared events), Taxonomic relations.

Thanks to this notion, we have a better insight of the meaning of words, however, this does not come with any computational model. The current best models are **Vector semantics** models.

The following paragraphs show different methods or general concept to encode words (on a sentence level or word level). Usually the objective is to find a vector representation of the entities we want to describe.

Bag of Word Count the number of word of in a sentence, this gives us a One Hot Vector (can be normalized as a frequency vector) as input.
Problem :

- Very Sparse (if each word of the dictionnary is a dimension, this make more than 170 000 dimension for english)
- This does not give any intel about the relationship of words - of common subject (e.g. Cat and kitten) or as grammar (subject, verb, etc.)

Mapping with concept Use logical concept to classify words :
-> Cat and kitten are feline mammal
-> Mouse and rat are rodent mammal
The WordNet (https://en.wikipedia.org/wiki/WordNet) synset (dictionnary of synonymes and themes) group words in such a way and is a reference for AI and ML application and research. Problem :

- Misses Nuances between words as concept are based on similiraty and not on differences.
- Misses new meanings, languages evolutions, etc.
- Classify word as concept without those issues is a problem in itself called Word Sense Disambiguation (WSD).
- Ressources does not exist for most languages or even for specialised vocabulary (Medical, any technical subject).

Features extraction An other solution is to extract only some features from words. Those may have to be designed depending on the application (for better accuracy), but this can be powerful and efficient. For exemple counting the % of positive and negative words for sentiment analysis. Problem :

- Require domain knowledge
- The feature are really task-specific and the extraction can be difficult.

Corpora and distributional hypothesis All those representations forget (partially or totally) a significant aspect of Natural Language, the context. A way to explain and describe context is to say that a word meaning is given by the words arround it, then the precise meaning of a word can be 'capture' in the word itself plus probability of surrounding word beeing in a specific corpora.

With a Fixed size of window context and a many corpora with different context available this is computable.
The use of probabilities and frequencies help us to move from discrete to continous representations, which makes possible to use a wider variety of model, which achieve generally better results.

This way (Describe by Firth in 1957) gives some explanations for why further methods work.

It can be done with different manner, for example with Co-occurence matrix. This are also One Hot vector, but here each word is a vector and not each sentences. Different meaning of the same word can be encoded as different vector, it is easy to remove stop words (common word use in natural languages, but with a low ammount of meaning), Terms can be weighted by TF-IDF (term frequency–inverse document frequency) and some notion of similarity ca be incorporated (distance between vector for example). All of this tend to escape the discrete representation, however this is still very large space and therefore very sparse representations.

TF-IDF : Weighing terms in the vector Co-occurence matrix take the raw frequency of co-occurent words, but this is not the best measure of association between words. The raw frequency is not enough discriminative as a lot of really frequent word but meaningless can be shared by two words.

In order to avoid this paradox, the TF-IDF (Term frequency - Inverse document frequency) use the product of TF $= 1 + \log_{10}(count(t,d)$ (or 0) and the IDF $= \log_{10} \frac{N}{df_t}$ where df_t is the number of document where t occures and N the number of all document.

$$\text{tf-idf} = tf_{t,d} \times iff_t$$

The tf-idf is commonly use as a baseline for weighting a co-occurence Matrix.

Vector Space models In order to use known Inference/Machine Learning models, we want to represent words as vector in mathematical space. We embed words in a continuous vector space where semantically similar words are mapped to nearby points. The previous description (based on Co-Occurence matrix and diverse weighting) is not enough, but the use of Dimension reduction can help us to get more dense representation. For example the use of SVD (Singular Value Decomposition) can easily reduce the dimension of 170 000 words to 1 000 dimensions.
Problems:

- SVD is computationnally expensive ($O(mn^2)$)
- Incorporating new word require to re-compute the SVD

The idea to solve those two problems have been found in representation learning, and their efficiency are the explanation of most of the progress in NLP in the recent years.

8.3 Learning word representation

The solution to most of the word representation problems is the learning of those representations. This work as many machine learning problem, the model test how well it performs on a task and the error is used to enhance the representation.
Here the task is the prediction of context words in context. For a given word as input the model gives a predicted context (words) as output, the score is given by a distance between words (using corpora as training set). Of course each word are vector. This is both computationnaly efficient (compared to SVD) and scalable with the corpus size (to some extent).
All deep learning Idea can then be applied here, with more or less efficiency.

8.3.1 Word embeddings : word2vec

The most known word embeddings modelx (which have been a revolution in the NLP field) are word2vec (by Mikolov et al., 2013). Those papers introduce two different models :

Add ref to Mikolov papers

- Continous Bag-of-Words model (CBOW):
  - The objective is to predict a target word w_t given context words $w_{t-j},...,w_{t-1},w_{t+1},...,w_{t+j}$.
- Skip-Gram model:
  - The objective is to predicts context words $w_{t-j},...,w_{t-1},w_{t+1},...,w_{t+j}$ given a target word w_t.

But actually we are not really interested in the prediction made by the algorithms, but only in the weights of the models, which give us a dense representation of the words. Therefore it can be rephrased as a classification problem : given two words, how likely they can occure together. This can be done with simple Logistic Regression classifier, and the weight of the logistic regression will be the embedding matrix.

Skip-gram model

The Goal of the Skip-Gram model is to find word representations that are useful for predicting context word (w_{t+j}) given a word w_t. For this we maximise the following equation :

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c}\log p(w_{t+j}|w_t,\theta)$$

which is the average log probabilities for context window c given the parameters θ. T is the size of the corpus.

The probability can be defined using the softmax function :

$$p(w_O|w_I) = \frac{\exp(v_{w_O}^\top v_{w_I})}{\sum_{w \in W} \exp(v_w^\top v_{w_I})}$$

where v_w are the embedding vector of the word w

- 1. Treat the

Neural Net implementation The neural network for the Skip gram model is simple, the input vectors are one-hot vector of the size of the language (Ex: 10 000), then there is a hidden layer of significantly fewer dimension (Ex: 300), which are fully connected to an output layer of the size of the language (here 10 000) with a softmax activation. Each dimension $i$of the output learn the probability of having the word w_i as a context word for the input w_I. After learning, we can remove the output layer, and getting the embedding of the words as the output of the previous layer. This can be seen a little bit as encoder neural net for computer vision, with encoder and decoder part.

As a result, the hidden layer is a embedding matrix, working as a look-up table.

The training can be done with Gradient Descent and Neural net algorithm for learning. How ever, there are a lot of parameters in naive Skip-Gram (size of corpus time embedding dimension) which are prohibitively expensive for large dataset.

Negative Sampling However there is no need for a full probabilistic model and some trick can help to reduce the computation. A binary classification objectif can be used for training the model, using logistic regression to discriminate real target words from other noise words. This is called Negative Sampling, which can be written as maximising :

$$\log Q_\theta(D = 1|w_t, c) + k\mathbb{E}_{w \sim P_{noise}}[log Q_\theta(D = 0|w, c)]$$

Where the first term is the binary logistic regression probability of seeing the word w_t in the context c in the dataset D, calculated in terms of the learned embedding vectors. The expectation is approximated bu using k contrastive words from a noise distribution.

As for intuition, this can be seen as the following :
- To predict a word Context from a word Input, first select k noisy (i.e. contrastive) words which are not in the context of the word Input and then compute the loss for observed and contrastive pairs. at a step t, the objective is $\log Q_\theta(D = 1|Context, Input) + \log Q_\theta(D = 0|Noise_k, Input)$

How to select the Negative Sampling ? The probability for selecting a word need to be related to its frequency. The number of sample required need to be larger for small dataset and smaller for large datasets. Arround 5-20 words against 2-5 words for larger dataset.

Exemple of vector representation Add explanation about what vector capture (king, queen, man, woman exemple)

Other tricks

- Treating common words pairs as single "words"
- Subsampling frequent words to decrease the number of training examples
- Using hierachical softmax

Other kind of embeddings word CBOW, Glove, ELMo and BERT Contextualised word embeddings seems really powerful, but really costly

8.3.2 Discussion on Embedddings

Bias and Embeddings One of the problem with embeddings is that the learning process relies on a dataset (the corpus) which is by nature imperfect and biaised. One Exemple is the gender biais : As the corpus contains data about job, it may decide that man and woman have different similarity for job and reproduce that nurse is associated with women and doctor to man as this is the bias of the corpus (Bolukbasi et al. 2016).

Evaluating Vector Models

- Test performance on similarity
- WordSim-353
- SimLex-999
- Toefl Dataset
- Stanford Contextual Word Similarity (SWCS)
- Analogy task

8.4 ML Algorithms for NLP

NLP use classifier for different tasks, such as sentiment analysis, fake news, authorship attribution, detection of suicide risk, patient risk assessment, etc.

8.4.1 Naives Bayes Classifier

This is the most simple classifier which work well in some NLP application, such as text categorization, it is used a lot in anti-spam algorithm for exemple.

Work with a training corpus, use feature vector of word input $x = [x_1, ..., x_I]$ and we are looking for the output class $\hat{y} = \arg\max_y P(y|x) = \arg\max_y P(x|y)P(y)$

We assume in Naive Bayes that the features of x are independant (in particular the order and the position in the text is not taken into account). Which makes possible to write :

$$\hat{y} = \arg\max_y P(y)\Pi_1^I P(x_i|y)$$

which lead in practise to

$$P(y) = \frac{N_y}{N_x} \text{ and } P(x_i|y) = \frac{\text{count}(x_i, y) + 1}{\Sigma_{x \in V}(\text{count(x, y)+1})}$$

with V the vocabulary across classes

In order to ensure the hypothesis, usualy a simple Bag-of-Word representation is used.

Problems

- All featurees are equally important (this is not the case generally)
- There is a strong conditonal independence assumption
- The context is not taken into account
- The MLE training have problem to compute probabilities of word which never appear in some classes. The solution might be to add a Laplace Smoothing.
- Unknown words are not tractable (Require to be ignored for predictions)

Data: Corpus D and Classes C
Result: $\log P(C) and \log P(w|c)$
forall $c \in C$ **do**
  N_{doc} = Number of document in D
  N_c = Number of document within c
  logprior[c] $\leftarrow \log \frac{N_c}{N_{doc}}$
  $V \leftarrow$ Vocabulary of D
  bigdoc $\leftarrow$ append(d) for d in D with class c
  forall $w \in V$ **do**
    count(w,c) $\leftarrow$ number of occurences of w in bigdoc[c]
    loglikelihood[w, c] $\leftarrow \log \frac{count(w,c)+1}{\sum(count(w',c)+1)}$
  end
end
return logprior, loglikelihood, V

Algorithm 4: Naive Bayes Algorithm training with Laplace Smoothing

8.4.2 Variant of Bayes Naive

- Binary multinomial naive Bayes.
- Use of sentiment lexicons for unlabelled corpus (General Inquirer, LIWC, MPQA, etc.)

8.4.3 Logistic Regression

See Chapter Regression for details, basically we are adding weight to each features x_i of x
example : The Movie is Bad → Bad might be more important than movie in sentiment analysis.

The logistic Regression is particularly suitable for discovering link between features and some outcome. This is an other baseline for classification tasks.

Difference with Naive Bayes and advantages The main difference with naive Bayes is that Logistic Regression is a discriminative classifier instead of a Generative Classifier.

The strong independance assumption of naive Bayes implies that strong correlatio will artificially augment the importance of some features (imagine what happen if you duplicate one feature). Logistic Regression is much more robust to correlated features and will be the preffered task for larger documents and datasets.

However, on very small datasets or short document Naive Bayes work extremely well and is much more faster to implement (there is no optimization step).

8.4.4 Neurals Networks

See DeepLearning for details of what is a neural networks.

The imput of neural networks can be one-hot representation of words, automatically learn dense feature representations (word embedding within the NN) or pre-trained dense representation (from another models)

NN are used because of the automatic learning of features which are hard to describe humanly (because of grammar, semantics, etc) and the non-linearity, their flexibility to fit highly complex data.

We talk about Neural Language Models for model which use neural nets to compute the probability of the next word given n previous word

Neural Language model can use pre-trained word embeddings, making possible to get dense embedding even with small corpus, and to learn faster.

However Neural Nets require usually lot of data for training correctly (with overfitting, etc).

8.4.5 CNNs in NLP

Context approach

Character-Level Approach

8.4.6 Recurrent Neural Networks (RNN)

Use the sequence structure of language. Very efficient but none paralelizable.

8.5 Preprocessing in NLP

Require consistency, truecase (instead of lowercase), removal of stopwords (espacially important for smaller datasets)

8.6 Evaluation of NLP model

3 different possible evaluations :

- Precision : $\frac{TP}{TP+FP}$
- Recall : $\frac{TP}{TP+FN}$
- Accuracy $\frac{TP+TN}{TP+FP+TN+FN}$
- F-Mesure: $F = k \times \frac{precision \times recall}{precision+recall}$

With TP = True positives, FP = False positives, TN = True negatives, FN= False negatives

8.7 Language Model

Language model task = LM's tasks. Language models are models that assign probabilities to each possible next word given a history of words. This can be generalised for entire sentences or operate at character level (or any symbol).

Why are Language Models useful ?

- Speech/Handwritten recognition
- Spelling correction
- Machine Translation
- Augmentative Communication
- and even more

In all this task, probability of the next words (or entities) can drastically remove the space of search.

Warning In the following, n-gram can design either a sequence of n-words or the Language model based on n-gram.

8.7.1 Basic N-gram language models

A N-grams in simply a sequence of n words, the LM's task here is to evaluate $P(w|h)$ where w is next word and h the history. P can then be estimated from frequency counts, but require large corpus.
Longer history or even for full sentences require very large corpus, and is not feasible for long sentences.
However, we can decompose the n-gram as n different word. This is translate in term of probablility by $P(h) = P(w_1...w_n) = P(w_1)P(w_2|w_1)...P(w_n|w_1...w_{n-1})$ using a conditional chain rule. This help us to see this as Markov Chain, and making Markov Assumption :

$$P(w_n|w_1^{n-1}) \simeq P(w_n|w_{n-N+1}^{n-1})$$

where $w_1^{n-1} = w_1...w_{n-1}$. This mean that for a n-gram, the next word probability depend only the last N few words (or are approximate so). This approximation is the core of n-grams. Bigram use only the last word to estimate the next, trigram the two lasts, and n-gram n-1 words.

The kind of linguistic captured by bigram can be strictly syntactic, but also can be various kind of things, like cultural association, gender, etc.

MLE as relative frequencies : The estimation of the n-gram probabilities can be done using MLE and the **Relative frequency** which is the ratio between observed frequence and the observed frequency of a prefix.

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1} w_n)}{C(w_{n-N+1}^{n-1})}$$

Use log space : For computing the probability multiplication, log space is much more stable.

How to decide on n (of the n-gram) : In practice bigram are never used, trigram are more common, but this can be larger with a consequent training dataset (larger n-gram require obviously larger dataset). More over it can be required context data (outside sentences data) for correct inference on the begining and ending of a sentence. The context data can be replaced with pseudo word as padding.

Google exemple ressource https://ai.googleblog.com/2006/08/all-our-n-gram-are-be html

8.7.2 Evaluation of language models

The best way to evaluate a model is to test how much it performs on a application regarding some baseline models. This is called **extrinsic evaluation**. However running some complete systems can be expensive, so **Intrinsic evaluation** can be done.

The question that can be ask is "Does it prefer real sentences to ungrammatical ones ?"
The measure the perplexity of an unseen corpus can give this answer
The **Perplexity** is the inverse probability on the test set, normalized with the number of words.
for a word W : .

Add perplexity formulas

The lower the perplexity is, the better our model is on the test set. The perplexity can be understand as the **Weighted average branching factor** of a language, i.e the number of possible next words that can follow any word.

8.7.3 Sparsity

N-gram rely on dataset, which have some various problems.

- Bias : Specific vocabulary, old corpus, etc
- Missing sequences, leading to 0 probabilities.
- Unable to contains new words

This lead to sparsity (lot of 0 probabilities) in the Probabilities given by the n-grams. This means that for a lot of sequences, there is not a lot of different possibility for the next words. A generative n-gram will often then directly copy the corpus sentences, This can be seen as a kind of "overfitting" in this tasks. Furthermore, if the corpus is not adapted to the tasks then there is fewer chance to get good results.
Moreover, if any sequence in a test set has a 0 probabilities then the full perplexity of the set is untractable.
Some techniques exists

Unknown words : Open vocabulary, Out of vocabulary (OOV), OOV rate. <UNK> pseudo word can be added, using vocabulary restriction, or frequency.

Smoothing What can be done for known word which appear in an unknown context ? Smoothing (discounting) to avoid 0 probabilities.

- Add-1, juste Add 1 to the count for each sequence. This is named the **Laplace Smoothing**. It does not perform well, but useful as baseline.
- Add-K, optimizing k on a devset. Usefull for classification, but not for language modeling.

Back-off Simply, when n-gram as probabilties 0, try with lesser n until finding none 0 proba. The proba can still be discounted when n is going lower. Using for exemple the Katz Backoff for discounting. Often the Katz is used with a Good-Turing Smoothing.

Interpolation Same idea as Back-Off, but mix the probabilities of different size of n-gram with different weight, instead of only the largest one.

All these methods may have hyperparameters, such as the weighing of discounting. Those can be optimized with a dedicated separate dataset, named **held-out** corpus.

8.8 Structured prediction

All the models that have been seen before forget about some very important points of the language : his inherent structure. Of course they don't need it theory, but models are always difficult to train, and using the language structure can only be benefitial. The formalism of this structure is called **Part-Of-Speach**

8.8.1 Part-Of-Speech tagging (POS)

POS tagging is the assignement to each word of a sentence of a classes/syntactic categories. This is really useful because POS reveal a lot about the role of a word and his neighbours. POS make possible to extract the subject of an action, which information is the main and which one is only background, etc. POS is really important in information extraction. But text never come with POS labels, which explain the need for tagging algorithms.

Tagset : The tagset is the different syntactic categories that we want to use for a task, there is no unicity as it might depend of the task, the language, and it can be extend to any language, even theoritical language, if they have the good properties. However, in NLP task, as most research is about english, the following tags are the most common : Adjective, adverb, interjection, noun, proper noun, verb, punctuation, symbol (like math symbol), adposition, auxiliary verb, coordination conjonction, determinant, numeral, particule, pronoun, subordination conjonction and other. Actually this is a little more complicated than this, for exemple have a look to the Penn Treebank Part-Of-Speech tagset with 45 categories for English.

Baseline method : Of course it is impossible to set for each word in a dictionnary a unique tag but the baseline method is simply to assign to each words its most frequent POS tag according to a dataset and to set all unknown word to NOUN. The results with the baseline is quite impressive, arround 90% for any test set which are similar to the training set.

POS ambiguities However the exeptions are the devil in this case. Indeed, the ambiguities are the most difficult to get, and the ones whith the highest probabilities to deeply change the meaning of a sentences...
POS is really a disambiguation task. And if ambiguous word (wrt POS) are 15% of the vocabulary, they represent more than half of the words use in text.

8.8.2 Probabilistic POS tagging

The concept of probabilistic POS tagging is the following : given $W = w_1, w_2, ..., w_n$, we want to compute $P(T|W)$ with $T = t_1, t_2, ..., t_n$ a sequence of POS tags.
Therefore, A generative approach can be made, with

$$P(T|W) \simeq P(W|T)P(T)$$

and $P(T)$ can be decompose with the chain rule and simplify with markov assumption (The tag t_i depends only on a few previous tags)

$$P(T) \simeq P(t_1)P(t_2|t_1)...P(t_n|t_{n-1})$$

and $P(W|T)$ can be compute assuming that each word depend only on his tag (and not on any other words/tags) :

$$P(W|T) \simeq P(w_1|t_1)P(w_2|t_2)...P(w_n|t_n)$$

and finally $P(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})}$ and $P(w_i|t_i) = \frac{C(w_i,t_i)}{C(t_i)}$
Then we can tag new sentences, going left to right, using at each step the highest $P(T|W)$ for tagging the next word. At each step it require to compute only $P(t_i|t_{i-1})P(w_i|t_i)$.
However, using the max probabilities greedily make the algoritms to ignore potentially goods paths. These approach can't guarranty that we find $\hat{T} = \arg\max_T P(T|W)$.

8.8.3 Hidden Markov Model (HMM) tagger

HMM are sequence model (or classifier), which mean there are model which give an output for each element of the input sequence.

Add reference to a good definition and explanations of hidden markov model

In the POS Tagging task, the observable event are the words of the sentence, and the hidden event are the TAG for each words. The HMM has 2 probability matrices, one for the likelihood $P(t_i|t_{i-1})$ the transition probability and one for the likelihood $P(w_i|t_i)$ the emission probability. .

Add graph of HMM example

The transition Matrices are estimate with the equation as before.

Tagging as decoding : Here is the important part, now that the HMM is set up with the same information (probabilities) of the simple probabilistic tagging the difference is the Tagging. Using a HMM, the tagging is actually the decoding task

and with the Viterbi Algorithm it is possible to find the best sequence.

Data: Sequence of word and HNN state graph
Result: Best-path (i.e best sequence) and the according probability
N = len(state graph)
T = len(Sequence)
Create a path probability matrix viterbi[N, T] **forall** sate s **do**
  viterbi[s, 1] $\leftarrow \pi_s * b_s(o_1) = \pi_s * B_{s,1}$ = Initial probability distribution times Emission probability.
  backpointer[s, 1] $\leftarrow 0$
end
forall t in T **do**
  forall state s **do**
    viterbi[s, t] $\leftarrow \arg\max_{s'}$ viterbi[s', t-1] $*A_{s',s} * B_{s,t}$
    backpointer[s, t] $\leftarrow \max_{s'}$ viterbi[s', t-1] $*A_{s',s} * B_{s,t}$
  end
end
bestpathprob = $\max_s viterbi[s,T]$
bestpathpointer = $\arg\max_s viterbi[s,T]$
bestpath = path starting at best pointer
return bestpath, bestpathprobb

Algorithm 5: Viterbi Algorithm for finding optimal sequence of Tag

8.8.4 Maximum Entropy Markov Model (MEMM) for POS tagging

8.8.5 Other approachs

CRF

Performance MEMM get 97% against 98% for human

Tools Spacy, NLTK

8.9 Sequence Models

Sequence Tagging

8.9.1 Types of Sequence Models

8.9.2 Neural Sequence Modelling

Non-Markovian Assumption

8.9.3 Language Modelling with Recursion

8.10 Recurrent Neural Network

8.10.1 Structure of LM

8.10.2 Training of RNNLM

Effective Back-Prop Strategies -Back-Prop Through Time
-Truncated Back-Prop Through Time

Vanishing Gradient

8.10.3 Gated Recurrent Neural Network

8.10.4 Other Variant

- Long Short-Term Memory networks (LSTM) - Similar functions

8.11 Bi-directional RNN

8.12 Transformer Based Models

Chapter 9

Reinforcement Learning

9.1 Markov Reward and Decision Process

9.1.1 State Value Function Closed-form

For a Markov Reward Process $(\mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma)$, with

- $\mathcal{S}$ the States
- $\mathcal{P}$ The transition matrix
- $\mathcal{R}$ the reward matrix
- $\gamma$the discount

We define then 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

9.1.2 Iterative Policy Evaluation Algorithm

In RL, we need not only the value of different state, but also a policy to define which action to take in any state. Then, we had an action space $\mathcal{A}$ and a policy π to a MRP to obtaine what is called a Markov Decision Process (MDP).

The first algorithm is a method to evaluate a policy. We don't need to store all the previous value of V while updating, and furthermore, it converges faster this

way.

Data: a MDP $(\mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma)$ and π the policy to be evaluate
Result: The value function V^{π}
Initialise V(s) =0 $\forall s$;
while **Convergence condition (number of epoch, $\Delta \leq$ threshold, ...)** **do**
  forall $s \in S$ **do**
    $v \leftarrow V(s)$;
    $V(s) \leftarrow \sum_{\alpha} \pi(s, \alpha) \sum_{s'} \mathcal{P}^{a}_{ss'} [\mathcal{R}^{a}_{ss'} + \gamma V(s')]$;
    $\Delta \leftarrow \max(\Delta, \mid v - V(s) \mid)$;
  end
end
Return V

Algorithm 6: Iterative Policy Evaluation Algorithm

9.2 Dynamic Programming in RL

In order to compute an optimal policy, we can do better than trying every policy and evaluating their V^{π}, the first class of algorithm enter in the Dynamic programming family. Convergence are assured by theorem relative to the Belmann equation (Policy Improvement theorem, Belmann principle of optimality, ...).

9.2.1 Policy Iteration Algorithm

First algorithm simply apply a greedy **deterministic** policy relative to the Value function, and recalculate the new Value function, until the policy is stable (and according to theorem, optimal).

Data: a MDP $(\mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma)$
Result: The optimal policy $\pi*$ and his corresponding value function $V^{\pi*}$
Initialise V(s)=0 and $\pi(s) \in \mathcal{A}, \forall s$;
while **Convergence condition (policy is stable, or number of epoch)** **do**
  Do a Policy Evaluation for π (with the Iterative Policy Evaluation for exemple) **forall** $s \in S$ **do**
    $b \leftarrow \pi(s)$;
    $\pi(s) \leftarrow \arg\max_{a} \sum_{s'} \mathcal{P}^{a}_{ss'} [\mathcal{R}^{a}_{ss'} + \gamma V(s')]$;
    policy-not-stable $\leftarrow (b \neq \pi(s)) \vee$ policy-not-stable
  end
end
Return V

Algorithm 7: Iterative Policy Evaluation Algorithm

This can be seen as the following scheme :

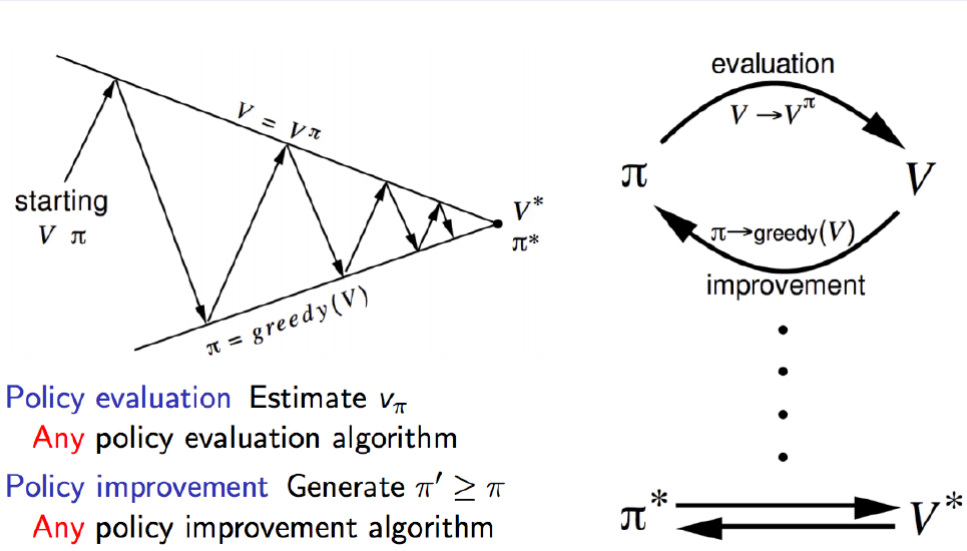

Figure 9.1 – Generialised Policy Iteration Algorithm

9.2.2 Value Iteration Algorithm

It can be shown that we don't need to wait for the convergence of the Value Function before iterating the policy in the previous algorithm. We can shorten some steps, and even get rid of explicity updating a policy (we are always using the greedy policy wrt. $V(s)$). This lead to the following algorithm :

Data: a MDP $(\mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma)$
Result: The optimal policy $\pi*$ and his corresponding value function $V^{\pi*}$
Initialise V(s)=0 $\forall s$;
while **Convergence condition (Value function is stable, or number of epoch)** **do**
  $\Delta = 0$;
  forall $s \in S$ **do**
    $v \leftarrow V(s)$;
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;
    $\Delta \leftarrow \max(\Delta, | v - V(s) |)$;
  end
end
Return $\forall s, \pi*(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;

Algorithm 8: Iterative Policy Evaluation Algorithm

9.2.3 Assynchronous Backup in RL

Definition 5

- *The **Backup** in RL is the fact to update the value of a state using values of futures state states. This can be understood as you check what's going on in the future, and backup this information to present time to update your knowledge.*
- *A **synchronous backup** is a backup made simultaneously for all states (can be done in parallel).*
- *An **assynchronous backup** apply a backup to only one selected state. Those methods can be much more efficient in reducing computation (only deal we required backup) and are still guaranteed to converge.*

For exemple the policy iteration algorithm can be compute both way, but the assynchronous way converge faster.

Prioritised Sweeping

Definition 6

*A **sweep** consists of applying a backup operation to each state. A synchronous backup compute a sweep at every iteration whereas a assynchronous backup may never compute a complete sweep (depending on the strategy).*

*A **prioritised swweping** use a criteria and a priority queue in order to decide which value will be backup next.*

An implementation of a prioritised sweeping can be made using the **Bellman Error**

$$\| v(s) - \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s'} P[s'|s,a] v(s') \right) \|$$

as priority criteria. It requires the knowledge of the **reverse dynamics** $P[s'|s,a]$.

Data: a MDP $(\mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma)$
Result: The optimal policy $\pi*$ and his corresponding value function $V^{\pi*}$
Initialise V(s)=0 and the Bellman Error $B(s), \forall s$;
Initialise the priority queue in ordering the s with $B(s)$;
while **Convergence condition (Value function is stable, or number of epoch)** **do**
  select s the head of the queue $v \leftarrow V(s)$;
  $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$;
  Update $B(s)$ and the priority queue ;
end
Return $\forall s, \pi*(s) = \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$;

Algorithm 9: Prioritised sweeping with Bellman error

Real-time Dynamic Programming The idea here is to use the **agent** experience to guide the selection of states. Each iteration, you backup the state was just visited.

Data: a MDP $(\mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R}, \gamma)$
Result: The optimal policy $\pi*$ and his corresponding value function $V^{\pi*}$
Initialise V(s)=0 $\forall s$;
choose a starting state $s = s_0$;
while **Convergence condition (Value function is stable, or number of epoch)** **do**
  $v \leftarrow V(s)$;
  $V(s) \leftarrow \max_a(\mathcal{R}^a_s + \gamma \sum_{s'} P[s'|s,a]V(s'))$;
  choose the next step from s (greedy policy);
end
Return $\forall s, \pi*(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;

Algorithm 10: Prioritised sweeping with Bellman error

9.2.4 Properties and drawbacks of Dynamic Programming

- Dynamic Programming uses **full-width** backup $\Longrightarrow$ faster convergence but requires lot of ressource
- Require **complete** knowledge of the MDP $\Longrightarrow$ this is not really Machine Learning, but more optisation (all parameters are actually known).
- DP is effective for **medium size problems** (millions of state, backup not to expensive)

9.3 Model-Free Learning

The dynamic programing requires lot of information to be compute, especially what is called the **model**, the states, the actions, the rewards and the transitions. We might want to learn form model which are difficult to observe, and with partial information, etc. That's what we need **model-free learning** algorithms, which can deal with model without knowing in advance the transition and the rewards.

9.3.1 Monte-Carlo Algorithms

Monte-Algorithms are the class of algorithms which learn from episodes of experiences. Which mean basicaly, you play the story fully with what you know, then analyse what append, learn some knowledge from this, and try again. This is different from DP because here you are learning from "real life" experience.

(First Visit) Monte-Carlo Policy Evaluation The first visit MC only take into account the returns from the first visit (i.e. if a state is visited more than once in a trace, it does not count)

Data: a uncomplete MDP ($\mathcal{S}$, $\mathcal{R}$, traces T wich contain series of state and reward, π a policy, and still γ)
Result: The State Value of the MDP for π
Initialise $\hat{V}(s), \forall s$;
Initialise $Returns(s), \forall s$ with empty list **while** **Convergence condition (Value function is stable, or number of epoch)** **do**
  Get a trace τ from T ;
  forall **s in τ** **do**
    R $\leftarrow$ return from the **first appearance** of s in τ ;
    Append R to Returns(s) ;
    $\hat{V}(s) \leftarrow average(Returns(s))$;
  end
end
Return $\hat{V}$;

Algorithm 11: First Visit Monte-Carlo Policy Evaluation

Every Visit Monte-Carlo Policy Evaluation You can have the same algorithm, with taking into account all the visit from a trace.

Data: a uncomplete MDP ($\mathcal{S}$, $\mathcal{R}$, traces T wich contain series of state and reward, π a policy, and still γ)
Result: The State Value of the MDP for π
Initialise $\hat{V}(s), \forall s$;
Initialise $Returns(s), \forall s$ with empty list **while** **Convergence condition (Value function is stable, or number of epoch)** **do**
  Get a trace τ from T ;
  forall **s in τ** **do**
    R $\leftarrow$ returns from **all appearances** of s in τ ;
    Append R to Returns(s) ;
    $\hat{V}(s) \leftarrow average(Returns(s))$;
  end
end
Return $\hat{V}$;

Algorithm 12: Every Visit Monte-Carlo Policy Evaluation

You cannot backup death : if the trace lead to a death state (with no reward during the trace, or infinite negative reward), then the MC will not be able to backup

any sensitive data.

Batch vs Online Monte-Carlo

Incremental Monte-Carlo Update

Runing Mean for Non-Stationnary World

9.3.2 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

9.3.3 Temporal Difference Learning

Temporal Difference Value Function Estimation Algorithm

Add Comparison between MC and TD learning

9.3.4 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 13: SARSA algorithm with ϵ-greedy policy

Theorem 2
Convergence of Sarsa
$Q(s,a) \rightarrow Q^{\infty}(s,a)$ *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 $\epsilon \rightarrow 0$).*
- *Robbins-Monroe sequence of step-sizes α_t : which imply $\sum \alpha_t$ diverge and $\sum \alpha_t^2$ converge.*

Remarque : 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

9.4 Reinforcement Learning with Function Approximation

9.4.1 Exemple of features

Coarse Coding

Tile Coding

Radial-Basis Function

Deep Learning

9.4.2 Monte-Carlo with Value Function Approximation

9.4.3 Temporal Difference Learning with Value Function Approximation

9.4.4 Q-Learning with FA

9.4.5 subsection name

9.5 Deep Learning Reinforcement Learning

9.5.1 Experience Replay

9.5.2 Target Network

9.5.3 Clipping of Rewards

9.5.4 Skipping of Frames

Chapter 10

Logic-Based Learning

add ref to lecturer

10.1 Inductive Logic Programming (ILP)

10.1.1 Introduction to Concept Learning

In Concept Learning we aim to compute a definition of a concept, expressed in a given langage (called **hypothesis space**) that satisfies positive exemple and none of negatives ones. As an exemple, it can be a regexp which can match all words of a set, and none of an other set (see Regexp Golf)

Machine Learning Task The Inductive Logique Programming is a subset of Machine Learning wgere prior knowledge are expressed in declarative language. The task is then a search problem for an hypothesis that would minimise a loss function.

Given :

- A language of examples L_e
- a language of hypotheses L_h
- an unknown target function $f : L_e \rightarrow Y$
- a set E of training exemples $E = \{(e_i, f(e_i))\}$
- e loss function

We want to find the hypothesis $h \in L_h$ that minimises the loss function ($h = \arg\min_{h_j \in L_h} \text{loss}(h_j, E)$). We want the hypothese h to approximates as much as possible the function h

Different loss can be choosed, such as $l(h, E) = \frac{1}{|E|} \sum (f(e_i) - h(e_i)$ or the squared differenced

Data Mining Task In a Data Mining Task, the objective is also to discover hypothesis, but the loss is replaced by a quality criterion $Q(h,E)$ such that the search is now to find h such as $Q(h,E) = \frac{|c(h,E)|}{|E|} \geq \epsilon$. This expresses a notion of coverage. This is essential for the concept learning, as we refer generaly to the set of exemple covered by an hypothesis h

Predictive ILP Given

- A set of observation in L_e with positive examples E_+ and negative examples E_-
- Background knowledge B
- hypothesis language L_h
- a cover relation (notion of covers and rejects)

we want a the best cover Relation $c(H) = E_+$

When E is noisy, we can add a quality criterion such as before which act as the precision test.

ILP as search of program clauses The ILP machine learning tasks can be seen as search program. Assuming for now that the representing exemple, the hypothesis use the same logical formalism and that we are interested to learn definite clauses, we can look over all L_h the best arg for our criterions.
The naives method are obviously none computable in most cases.

Chapter 11

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.

11.1 Abstract Argumentation

11.1.1 Simple AA

Definition 7
*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 8

- ***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 contain in all prefered (i.e. in the sceptically preferred).*

Definition 9
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

11.1.2 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 14: 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

Data: An AA Framework, and a arg α
Result: Finite or infinite dispute tree corresponding to α
add a root node Label α as proponent ;
while The tree is not stable **do**
  for every $\beta \in$ proponent, and for every (γ, β) add a child-node γ as opponent;
  for every $\beta \in$ opponent, and for every (γ, β) add a node γ as proponent in the limit of one child-node per opponent ;
end

Algorithm 15: Computing dispute Tree

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

11.1.3 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 10

- $Attacks_{sup}$ *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 Framework 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.

Definition 11

- *We define the QuAD Framework as $\langle \mathcal{A}, \mathcal{C}, \mathcal{P}, \mathcal{R}, \mathcal{BS} \rangle$ Where $\mathcal{A}$ is the set of answer arguments, $\mathcal{C}$ is the set of con args, $\mathcal{P}$ are the pro args, which are disjoint, $\mathcal{R}$ the relation (with $\mathcal{R}^+$ the supporters and $\mathcal{R}^-$ the attackers) and $\mathcal{BS}$ the Base Score, which give score to each args.*

- *The **Base funcion** is* $f(v_1, v_2) = v_1 + (1 - v_1)v_2$
- *the **aggregation function** as a recursive definition :*

$$\begin{aligned} n = 0 &\implies \mathcal{F}(S) = 0 \\ n = 1 &\implies \mathcal{F}(S) = v_1 \\ n = 2 &\implies \mathcal{F}(S) = f(v_1, v_2) \\ n > 2 &\implies \mathcal{F}(S) = f(\mathcal{F}(S \backslash \{v_n\}, v_n)) \end{aligned}$$

- *the **Combination function** is defined by:*
  $c(v_0, v_a, v_s) = v_0 - v_0 \cdot |v_s - v_a|$ *if* $v_a \geq v_s$
  $c(v_0, v_a, v_s) = v_0 + (1 - v_0) \cdot |v_s - v_a|$ *if* $v_a < v_s$
- *the **Strength Function** is defined as :* $\mathcal{S}(\alpha) = c(\mathcal{BS}(\alpha), \mathcal{F}(SEQ_S(\mathcal{R}^-(\alpha))), \mathcal{F}(SEQ_S(\mathcal{R}^+(\alpha))))$

The DF-QuAD (for **Discontinuity free**-QuAD) algorithm is simply to apply these semantics to a BAA framework, in setting initial weights to any arguments.

11.1.4 Argument Mining

11.1.5 AA with Preference

When taking into account preference in a AA, (in the form $X < Y = X$ is less prefered to Y) we can still use the classic AA semantics but we can improved it by using the preferences informations :

- By **deleting attacks** (an (x, y) attacks succeed iff $x \not\prec y$)
- By **reversing attacks** (if (x, y) and $x < y$ then we reverse by replacing this attack by (y, x)).
- By **Selecting Amongst extension**, using the preference to select extension with the same extension (ex : differentiate 2 stable extensions). Look at **Rich PAF** for some formal definition.

11.1.6 AA with Probabilities

11.2 Assumption-Based Argumentation

Assumption Based Argumentation introduce arguments as Assumptions $a \in \mathcal{A}$, or deduction σ with premises and conclusions achevied with Rules $R \subseteq \mathcal{R}$(deductive rules, as $q \wedge a \implies p$) defined in a language $\mathcal{L}$

11.2.1 Simple ABA

We can define **attacks** in ABA : an argument $A_1 \vdash \sigma_1$ attacks $A_2 \vdash \sigma_2$ iff σ_1 is the contrary of one of the assumptions in A_2.

11.2.2 ABA more DDs

11.2.3 p-acyclic ABA

Definition 12
*a **positive-acyclic ABA** is a AB framework where the dependancy graph of AB is acyclic.*
*The **dependancy graph** is a graph of all the rules, where assumption ($\mathcal{A}$) are deleted.*

11.3 ArgGame

Chapter 12

Sampling

Based on M. Deisenroth course Probabilistic Inference of imperial College

Motivation

In inference and more generally in probability, there is two main problem :

- **Generate samples** from a given probability distribution, for simulation, representation of distributions, etc.
- **Compute Expectations** of functions under distribution, for evaluating means, variancess, marginal likelihood, predictions in Bayesian models, etc.

The second problem imply most of times to solve/evaluate integrals, and in real life there are not often analytically computable.

12.1 Sample from distribution

12.1.1 Sampling Discrete Values

An easy way is to sample from the uniform distribtion and to define for each discrete value an interval in $[0, 1]$ corresponding to its probability.

12.1.2 Sampling from Continous Variables

Sampling from continuous distribution is more complicated. Sampling from uniform distribution is something that is easily done with random generator, and it is also possible to sample from specific distribution like Gaussian but not for any distributions.
However, here is an intuition :
- Visualize the area under the curve of the distribution, - Sample uniformly from this area.

Rejection Sampling One of the key idea is if sampling from $p(z)$ is difficult, evaluating $Zp(z)$ (even with unknown Z constant) is easy. Therefore it is possible to sample from an area which embrace the $Zp(z)$ curve, and then reject sample which are not under the curve.
Moreover, it is possible to find a better embracing area than a square, with a Gaussian or Laplace for example.

- Find a simpler distribution q(z) that we know how to sample from (Gaussian, Laplace, etc.)
- Find an upper bound $kq(z) \geq \tilde{p}(z) = Zp(z)$
- Then generate samples $z_i \sim q(z)$
- Generate $u_i \sim \mathcal{U}[O, kq(z_i)]$
- if $u_i > \tilde{p}(z_i)$, reject the sample, otherwise keep it
- Then, the paire (z, u) are uniformly distributed under the curve of $\tilde{p}(z)$

Acceptance Rate This is the ratio between the numbers of accepted and rejected samples. highly depends on k and k is tricky to optimize

Problem with high dimension k is probably huge, therefore low acceptance rate, which mean not efficient sampling.

Importance Sampling Key Idea : Do not throw away all rejected samples.

12.2 Different method of Approximate Integration

12.2.1 Numerical integration

Faisible for low-dimensionnal problems.

12.2.2 Bayesian quadrature

12.2.3 Variational Bayes

12.2.4 Expectation Propagation

12.3 Monte-Carlo Methods

12.3.1 Monte Carlo Estimation

Computing Expectation

$$E[f(x)] = \int f(x)p(x)dx \simeq \frac{1}{S}\sum_{s}^{S} f(x^s) \text{ with } x^s \sim p(x)$$

This Estimator is assymptotically consistent, the error is gaussian with a variance in 1/S independant of the dimension, and this Estimator is unbiased.

Making Prediction **Problem**: Require to generate Sample. See Sampling Section

12.3.2 Markov Chains MonteCarlo (MCMC)

The idea is to generate sample as a sequence, i.e. sample are not independant, and use a proposal density that depends on the previous sample.
This is a Markov property, therefore Markov Chain seems to be usefull for this problem.

Behaviour of Markov Chains Markov Chain can have 4 different behaviour, depending of the state and transition probability, etc.

- Diverge
- Converge to an absorbing state
- Converge to a deterministic cycle
- Converge to an equilibrium distribution, i.e the MC remains in a region, going arround in a random way.

Of course the behaviour we are interesting in is the fourth. The goal will be to define a Markov Chain with an equilibrium distribution equals to the distribution we want to sample for.
Even if the sample are not independant, it can be approximated if there are enough space between to choosen sample.

Conditions for Converging to an Equilibrium Distribution

- Invariance/stationnarity :If you are in p^* then the next step will stay in p^*.
  It is suffisant to have the following property : **Detailed balance** $p^*(x)T(x'|x) = p^*(x')T(x|x')$ with T the transition Matrix.
  This also ensure that the MC is reversible
- Ergodicity : any state are reachable from any state, which lead to consistency of the equilibrium independantly of the starting state.

Correelated Samples

12.3.3 Metropolis-Hastings

12.3.4 Gibbs Sampling

12.3.5 Slice

Chapter 13

Useful Computation

13.1 Data Centering using Matrix Multiplication

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