Machine Learning II
Week #5

Jan Nagler

Deep Dynamics Group
Centre for Human and Machine Intelligence (HMI)
Frankfurt School of Finance & Management

Add on

covid update for mini projects on covid:
https://fivethirtyeight.com/features/coronavirus-case-counts-are-meaningless/

Fit coefficients & Residuals

(OLS Minimization with L1 penalty = LASSO regularization)

λ

	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7	coef_x_8	coef_x_9	coef_x_10	coef_x_11	coe
alpha_1e-15	0.96	0.22	1.1	-0.37	0.00089	0.0016	-0.00012	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.4
alpha_1e-10	0.96	0.22	1.1	-0.37	0.00088	0.0016	-0.00012	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.4
alpha_1e-08	0.96	0.22	1.1	-0.37	0.00077	0.0016	-0.00011	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.3
alpha_1e-05	0.96	0.5	0.6	-0.13	-0.038	-0	0	0	0	7.7e-06	1e-06	7.7e-08	0	0
alpha_0.0001	1	0.9	0.17	-0	-0.048	-0	-0	0	0	9.5e-06	5.1e-07	0	0	0
alpha_0.001	1.7	1.3	-0	-0.13	-0	-0	-0	0	0	0	0	0	1.5e-08	7.5
alpha_0.01	3.6	1.8	-0.55	-0.00056	-0	-0	-0	-0	-0	-0	-0	0	0	0
alpha_1	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0
alpha_5	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0
alpha_10	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0

HIGH SPARSITY

Courtesy: K. Jain

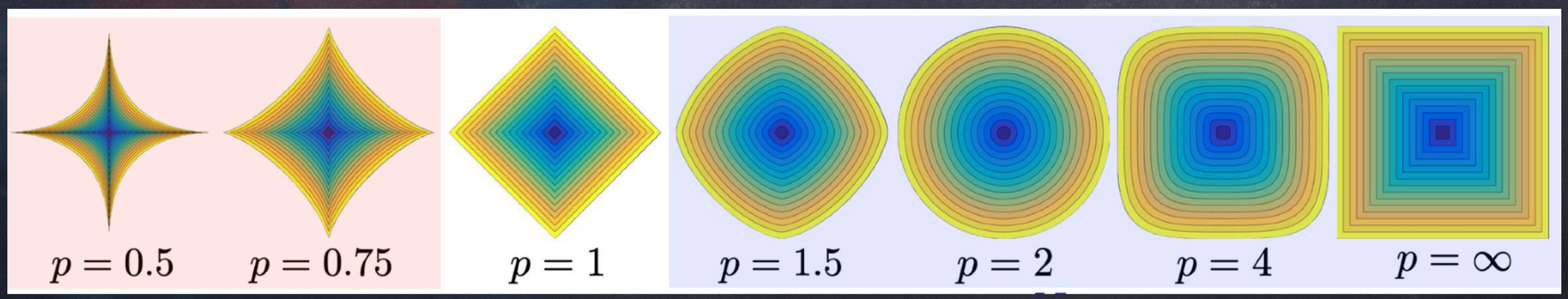
Regularization

LASSO regression is sensitive to correlation,
rather picks randomly few variables from
a larger group of highly correlated features

ElasticNet rather either leaves the whole group of
highly correlated features in (more stable model), or out

ElasticNet tends to select more variables,
typically leads to larger models,
more expensive to train, but more accurate

Lp norms / penalties graphically



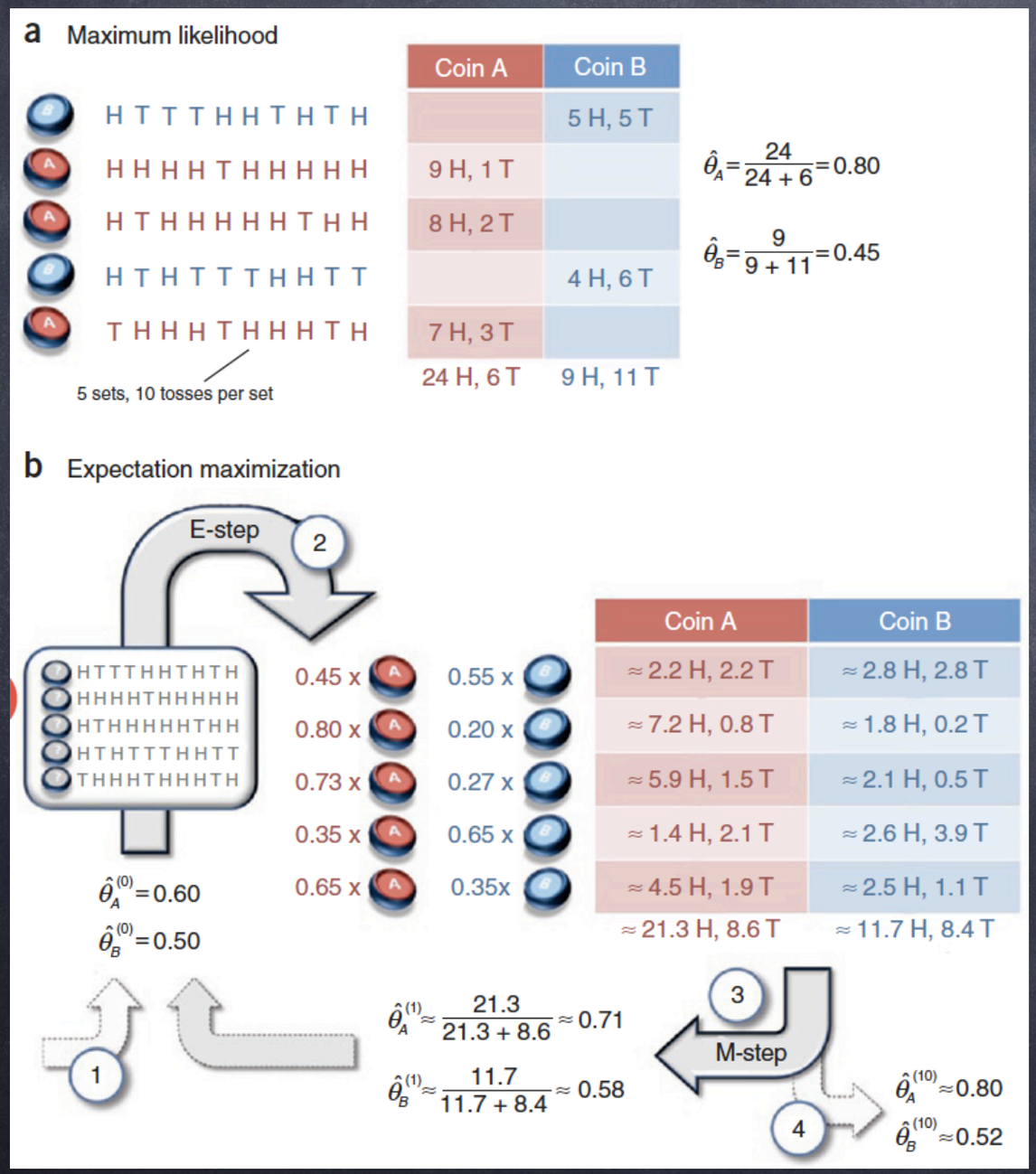
python

Regularization

python

Expectation Maximization: Coin toss

iPad: numbers explained & EM Step in formula



Coin python code

Markov Chain Monte Carlo (MCMC)

MCMC Sampling & State-of-the-art

Marginal-based

Gibbs (sampling in one direction per step, needs marginals)

Non-gradient-based, e.g. discrete variables

Metropolis Metropolis-Hastings (adaptive/elliptic) Slice …

Hamiltonian Monte Carlo (HMC)
e.g., No-U-Turn Sampler (NUTS)

based on a physical energy-conserved system
with well-defined energy landscape & Hamiltonian:=E(nergy)
fast for many continuous variables (Covid python example)
Uses symplectic integrator and other tricks (Intuition, iPad)

PyMC3: https://docs.pymc.io/notebooks/getting_started.html

Murray et al, Elliptical slice sampling, https://arxiv.org/pdf/1001.0175.pdf (2010)

Betancourt, A Conceptual Introduction to Hamiltonian Monte Carlo, https://arxiv.org/pdf/1701.02434.pdf (2018)

Markov Chain Monte Carlo (MCMC)

MCMC methods are instances of statistical Bayesian inference

Methods based on (random but correlated) sampling

Methods contrast deterministic statistical inference algorithms
such as expectation-maximization (EM)

Stationary distribution of the Markov chain is the desired joint distribution

Convergence to the equilibrium=stationary (target) distribution
typically guaranteed (under mild conditions) from
cumbersome (but awesome) theory

Markov Chain Monte Carlo (MCMC)
In a nutshell

(i) Start at current position

(ii) Propose moving to a new position (spinning the roulette wheel)

(iii) Accept/Reject the new position
based on the position's adherence to the data and prior distribution

(iv) If you accept: Move to the new position. Return to Step (i)
Else: Do not move to new position. Return to Step (i)

After a large number of iterations, return all accepted positions
This generates the desired (posterior) distribution

from *SIAM News*, Volume 33, Number 4

The Best of the 20th Century: Editors Name Top 10 Algorithms

By Barry A. Cipra

MCMC (Metropolis-Hastings) is among the top 10 most important algorithms of the 20th century!

Algos is the Greek word for pain. *Algor* is Latin, to be cold. Neither is the root for *algorithm*, which stems instead from al-Khwarizmi, the name of the ninth-century Arab scholar whose book *al-jabr wa'l muqabalah* devolved into today's high school algebra textbooks. Al-Khwarizmi stressed the importance of methodical procedures for solving problems. Were he around today, he'd no doubt be impressed by the advances in his eponymous approach.

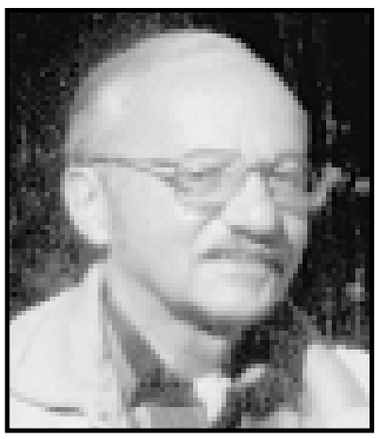
Some of the very best algorithms of the computer age are highlighted in the January/February 2000 issue of *Computing in Science & Engineering*, a joint publication of the American Institute of Physics and the IEEE Computer Society. Guest editors Jack Don-garra of the University of Tennessee and Oak Ridge National Laboratory and Fran-cis Sullivan of the Center for Comput-ing Sciences at the Institute for Defense Analyses put togeth-er a list they call the "Top Ten Algorithms of the Century."

"We tried to assemble the 10 al-gorithms with the greatest influence on the development and practice of science and engineering in the 20th century," Dongarra and Sullivan wrote. As with any top-10 list, their selections—and non-selections—are bound to be controversial, they acknowledge. When it comes to picking the algorithmic best, there seems to be no best algorithm.

Without further ado, here's the CiSE top-10 list in chronological order. (Dates and names associated with the algorithms should be read as first-order approximations. Most algorithms take shape over time, with many contributors.)

1946: John von Neumann, Stan Ulam, and Nick Metropolis, all at the Los Alamos Scientific Laboratory, cook up the Metropolis algorithm, also known as the **Monte Carlo method**.

The Metropolis algorithm aims to obtain approximate solutions to numerical problems with unmanageably many degrees of freedom and to combinatorial problems of factorial size, by mimicking a random process. Given the digital computer's reputation for deterministic calculation, it's fitting that one of its earliest applications was the generation of random numbers.



In terms of wide-

1947: George Dantzig, at the RAND Corporation, creates the **simplex method for linear programming**.

In terms of widespread application, Dantzig's algorithm is one of the most successful of all time: Linear programming dominates the world of industry, where economic survival depends on the ability to optimize within budgetary and other constraints. (Of course, the "real" problems of industry are often nonlinear; the use of linear programming is sometimes dictated by the computational budget.) The simplex method is an elegant way of arriving at optimal answers. Although theoretically susceptible to exponential delays, the algorithm in practice is highly efficient—which in itself says something interesting about the nature of computation.

Gibbs sampling

Gibbs Sampling is a MCMC method to draw samples from a complicated, or computationally expensive high dimensional joint distribution

Example: Complicated normalizing constants in Bayesian inference problems

Gibbs Sampler can draw samples from any distribution, provided all conditional distributions are available analytically

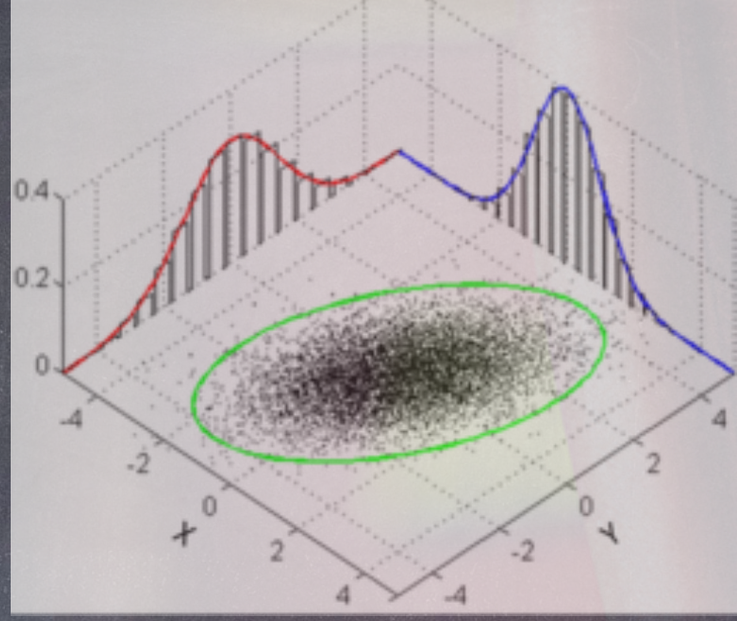
Gibbs sampling generates a Markov chain of samples, each of which is correlated with nearby samples

Generally, samples from the beginning of the chain (the burn-in period) may not accurately represent the desired distribution and are usually discarded

Thinning (i.e. discard every sample, up to the n-th sample) may be applied for high autocorrelation

Gibbs sampling

(Bivariate example)



Given data x

0. Randomly pick $\theta_2^{(i)}$

Repeat

1. Sample $\theta_1^{(i+1)} \sim p(\theta_1 | \theta_2^{(i)}, x)$

2. Sample $\theta_2^{(i+1)} \sim p(\theta_2 | \theta_1^{(i+1)}, x)$

Sampling new variable while holding all others constant
i.e. use the most recent parameter values when sampling

Main advantage of Gibbs sampling over other MCMC methods
(namely Metropolis-Hastings) is that no tuning parameters are required

Disadvantage: Cumbersome derivation of conditional probabilities

Lecture: Theory for bivariate normal distributions and implementation in Python

Gibbs sampling

iPad: Gaussians for MCMC

Python code Gaussian, Gibbs

Conditional distributions of multivariate Gaussians

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad x_1 : \dim(x_1) = p \in \mathbb{N}$$
$$\dim[x_2] = n - p$$

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \qquad \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \quad V \to \dim n$$

$$\det \Sigma = \det(\text{Cov. Matrix}) = |\Sigma| \neq 0$$

Joint distribution

$$f(x) = f(x_1, x_2) = \frac{1}{(2\pi)^{n/2} \cdot |\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

$$Q(x_1, x_2) = (x-\mu)^T \Sigma^{-1} (x-\mu) = \ldots$$

explicit, complicated

Marginal distribution

$\dim x_1 = p$
$\dim x_2 = q$

$$f_1(x_1) = \int f(x_1, x_2)\, dx_2 = \frac{1}{(2\pi)^{p/2}|\Sigma_{11}|^{1/2}} \times$$

$$e^{-\frac{1}{2}(x_1-\mu_1)^T \Sigma_{11}^{-1}(x_1-\mu_1)}$$

Conditional distribution of x_2 given x_1

$$f_{2|1}(x_2|x_1) = \frac{f(x_1,x_2)}{f(x_1)} = \frac{1}{(2\pi)^{q/2}|A|^{1/2}} \times$$

Mean: $b = \mu_2 + \Sigma_{12}^T \Sigma_{11}^{-1}(x_1-\mu_1)$

Cov: $A = \Sigma_{22} - \Sigma_{12}^T \Sigma_{11}^{-1} \Sigma_{12}$

$$\exp\left(-\frac{1}{2}(x_2-b)^T A^{-1}(x_2-b)\right)$$

$\Sigma_{12}^T = \Sigma_{21}$

Metropolis-Hastings sampling

Sampling algorithm for distributions from which it is difficult to sample directly
(due to intractable (conditional pdfs) integrals, normalization constants,
high computational costs for high dimensional distributions)

The sequence (called Monte Carlo chain) can be used to approximate
the distribution (e.g. from the histogram) or to compute integrals
(e.g. an expected value).

The core of the algorithm lies in a distribution that is an
acceptance probability
for the next proposed candidate of the Markov Chain

$$\alpha = A(x \to x') = \min\left(1, \frac{P(x')Q(x' \to x)}{P(x)Q(x \to x')}\right)$$

$P(x)$ Sample distribution (must not be normalized)

x is current state ———- next state is x'

$Q(x \to x')$ is suggested transition probability

Metropolis-Hastings sampling

Transition probability = Suggested transition Q

$$T(x \to x') = Q(x \to x')$$

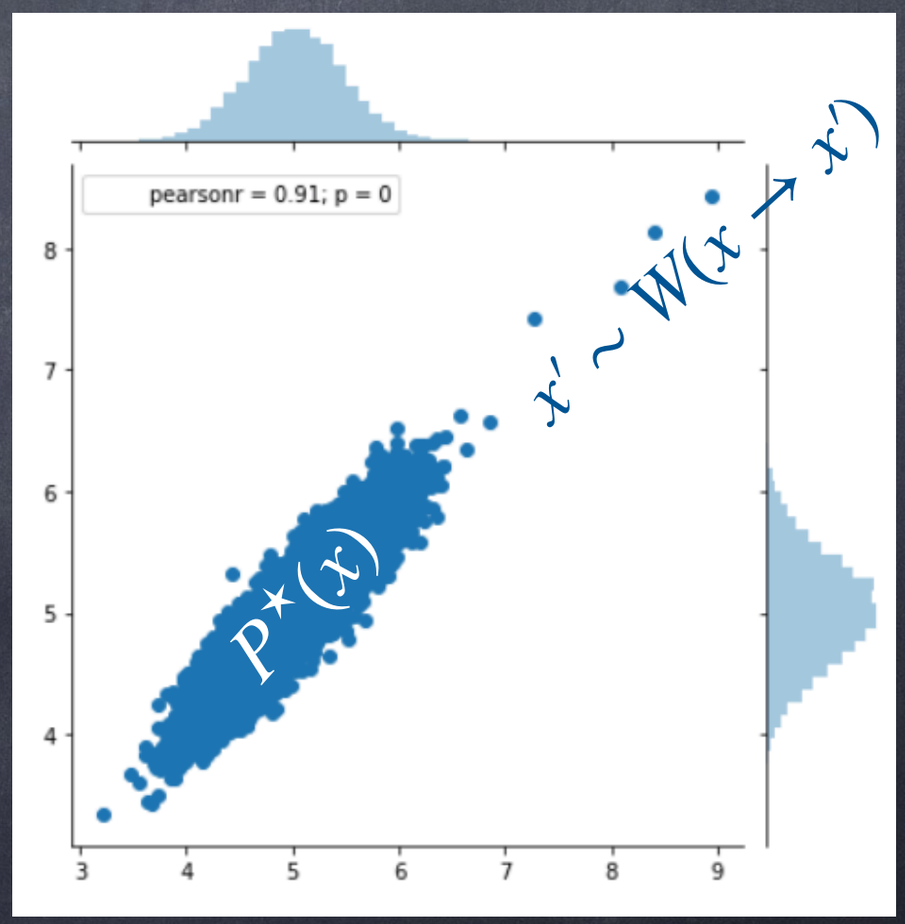
Probability (Wahrscheinlichkeit) of Markov Chain

$$W(x \to x') = Q(x \to x')A(x \to x')$$

Gibbs: W=Q

Target distribution or equilibrium- or stationary distribution

$$P^\star(x)$$



Lecture: Theory and implementation in Python

Markov chains

General

Normalization of Markov Chain

$$\sum_{x'} W(x \to x') = 1$$

Ergodicity of Markov Chain $W(x \to x') > 0$

(any x' must be reached from any other x with non-zero probability in a finite number of steps)

Special

Homogeneity of Markov Chain

$$\sum_{x'} P^\star(x') W(x' \to x) = P^\star(x)$$

A reversible Markov Chain (special property) exhibits

$$Q(x \to x') = Q(x' \to x)$$

Markov chains

Master equation of Markov Chain

$$\frac{dP(x,t)}{dt} = \sum_{x'} P(x')W(x' \to x) - \sum_{x'} P(x)W(x \to x')$$

Stationary distribution of Markov Chain

$P^\star(x)$ is solution of master equation for

$$\frac{dP^\star(x,t)}{dt} = 0 \to P^\star(x)$$

Detailed balance of Markov Chain

$$\to \sum_{x'} P^\star(x')W(x' \to x) = \sum_{x'} P^\star(x)W(x \to x')$$

$$= P^\star(x) \sum_{x'} W(x \to x') = P^\star(x)$$

Markov chains

Detailed balance of Markov Chain

$$\rightarrow \sum_{x'} P^{\star}(x')W(x' \rightarrow x) = \sum_{x'} P^{\star}(x)W(x \rightarrow x')$$

In MCMC detailed balance is enforced by the choosing W such that

$$P(x')W(x' \rightarrow x) = P(x)W(x \rightarrow x')$$

This justifies the Metropolis-Hastings acceptance probability (proof in lecture)

$$\alpha = A(x \rightarrow x') = \min\left(1, \frac{P(x')Q(x' \rightarrow x)}{P(x)Q(x \rightarrow x')}\right)$$

given $W(x \rightarrow x') = Q(x \rightarrow x')A(x \rightarrow x')$

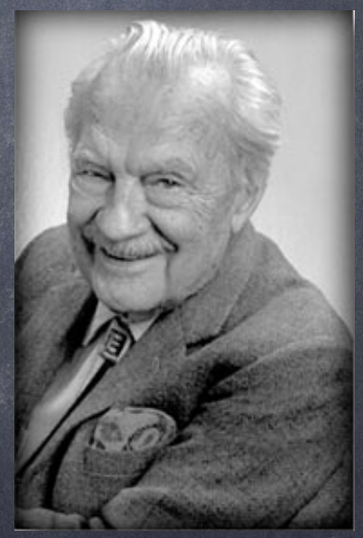
Metropolis and Metropolis-Hastings

Metropolis-Hastings acceptance probability

$$\alpha = A(x \to x') = \min\left(1, \frac{P(x')Q(x' \to x)}{P(x)Q(x \to x')}\right)$$

Metropolis acceptance probability for symmetric Q

$$\alpha = A(x \to x') = \min\left(1, \frac{P(x')}{P(x)}\right)$$



Historically: First Metropolis then Hastings generalised

MCMC Sampling & State-of-the-art

Marginal-based

Gibbs (sampling in one direction per step, needs marginals)

Non-gradient-based, e.g. discrete variables

Metropolis Metropolis-Hastings (adaptive/elliptic) Slice …

Hamiltonian Monte Carlo (HMC)
e.g., No-U-Turn Sampler (NUTS)

based on a physical energy-conserved system
with well-defined energy landscape & Hamiltonian:=E(nergy)
fast for many continuous variables (simple + covid python example)
Uses symplectic integrator and other tricks (Intuition, iPad)

PyMC3: https://docs.pymc.io/notebooks/getting_started.html

Murray et al, Elliptical slice sampling, https://arxiv.org/pdf/1001.0175.pdf (2010)

Betancourt, A Conceptual Introduction to Hamiltonian Monte Carlo, https://arxiv.org/pdf/1701.02434.pdf (2018)

Continuous & Discrete Priors

Continuous¶

`Uniform`([lower, upper])	Continuous uniform log-likelihood.
`Flat`(*args, **kwargs)	Uninformative log-likelihood that returns 0 regardless of the passed value.
`HalfFlat`(*args, **kwargs)	Improper flat prior over the positive reals.
`Normal`([mu, sigma, tau, sd])	Univariate normal log-likelihood.
`TruncatedNormal`([mu, sigma, tau, lower, ...])	Univariate truncated normal log-likelihood.
`HalfNormal`([sigma, tau, sd])	Half-normal log-likelihood.
`SkewNormal`([mu, sigma, tau, alpha, sd])	Univariate skew-normal log-likelihood.
`Beta`([alpha, beta, mu, sigma, sd])	Beta log-likelihood.
`Kumaraswamy`(a, b, *args, **kwargs)	Kumaraswamy log-likelihood.
`Exponential`(lam, *args, **kwargs)	Exponential log-likelihood.
`Laplace`(mu, b, *args, **kwargs)	Laplace log-likelihood.
`StudentT`(nu[, mu, lam, sigma, sd])	Student's T log-likelihood.
`HalfStudentT`([nu, sigma, lam, sd])	Half Student's T log-likelihood
`Cauchy`(alpha, beta, *args, **kwargs)	Cauchy log-likelihood.
`HalfCauchy`(beta, *args, **kwargs)	Half-Cauchy log-likelihood.
`Gamma`([alpha, beta, mu, sigma, sd])	Gamma log-likelihood.
`InverseGamma`([alpha, beta, mu, sigma, sd])	Inverse gamma log-likelihood, the reciprocal of the gamma distribution.
`Weibull`(alpha, beta, *args, **kwargs)	Weibull log-likelihood.
`Lognormal`([mu, sigma, tau, sd])	Log-normal log-likelihood.
`ChiSquared`(nu, *args, **kwargs)	χ^2 log-likelihood.
`Wald`([mu, lam, phi, alpha])	Wald log-likelihood.
`Pareto`(alpha, m[, transform])	Pareto log-likelihood.
`ExGaussian`([mu, sigma, nu, sd])	Exponentially modified Gaussian log-likelihood.
`VonMises`([mu, kappa, transform])	Univariate VonMises log-likelihood.
`Triangular`([lower, upper, c])	Continuous Triangular log-likelihood
`Gumbel`([mu, beta])	Univariate Gumbel log-likelihood
`Rice`([nu, sigma, b, sd])	Rice distribution.
`Logistic`([mu, s])	Logistic log-likelihood.
`LogitNormal`([mu, sigma, tau, sd])	Logit-Normal log-likelihood.
`Interpolated`(x_points, pdf_points, *args, ...)	Univariate probability distribution defined as a linear interpolation of probability density function evaluated on some lattice of points.

A collection of common probability distributions for stochastic nodes in PyMC.

PyMC3: https://docs.pymc.io/api/distributions/continuous.html

Continuous & Discrete Priors

Discrete¶

`Binomial`(n, p, *args, **kwargs)	Binomial log-likelihood.
`ZeroInflatedBinomial`(psi, n, p, *args, **kwargs)	Zero-inflated Binomial log-likelihood.
`BetaBinomial`(alpha, beta, n, *args, **kwargs)	Beta-binomial log-likelihood.
`Bernoulli`([p, logit_p])	Bernoulli log-likelihood
`Poisson`(mu, *args, **kwargs)	Poisson log-likelihood.
`ZeroInflatedPoisson`(psi, theta, *args, **kwargs)	Zero-inflated Poisson log-likelihood.
`NegativeBinomial`(mu, alpha, *args, **kwargs)	Negative binomial log-likelihood.
`ZeroInflatedNegativeBinomial`(psi, mu, alpha, …)	Zero-Inflated Negative binomial log-likelihood.
`DiscreteUniform`(lower, upper, *args, **kwargs)	Discrete uniform distribution.
`Geometric`(p, *args, **kwargs)	Geometric log-likelihood.
`Categorical`(p, *args, **kwargs)	Categorical log-likelihood.
`DiscreteWeibull`(q, beta, *args, **kwargs)	Discrete Weibull log-likelihood
`Constant`(c, *args, **kwargs)	Constant log-likelihood.
`OrderedLogistic`(eta, cutpoints, *args, **kwargs)	Ordered Logistic log-likelihood.

PyMC3: https://docs.pymc.io/api/distributions/discrete.html

Ridge theory in matrix calculus (if time)

Week #6

Final Lecture ML 2
Finishing up & Add ons &
News, Views & Threats in Machine Learning

Jan Nagler

Deep Dynamics Group
Centre for Human and Machine Intelligence (HMI)
Frankfurt School of Finance & Management

Hyperparameters from Bayesian Optimization

Is everybody on the floor?
We put some energy into this place
I want to ask you something
Are you ready for the sound of Scooter?
I want to see you sweat
I said, I want to see you sweat, yeah
Hyper, Hyper
Hyper, Hyper
Hyper, Hyper
We need the bass drum
Come on
Hyper, Hyper
Hyper, Hyper
It's so beautiful to see your hands in the air
Put your hands in the air
Come on
This is Scooter
We want to sing a big shout to U.S. and to all ravers in the world
And to Westbam, Marusha, Stevie Mason, The Mystic Man, DJ Dick
Carl Cox, The Hooligan, Cosmic, Kid Paul, Dag, Mike Van Dyke
Jens Lissat, Lenny D., Sven Väth, Mark Spoon, Marco Zaffarano
Hell, Paul Elstak, Mate Galic, Roland Casper, Sylvie, Miss Djax
Jens Mahlstedt, Tanith, Laurent Garnier, Special, Pascal F.E.O.S.
Gary D., Scotty, Gizmo and to all DJs all over the world
Hyper, Hyper
Sit there
Be good
Bye-bye

Source: LyricFind
Songwriters: H. P. Baxxter / Rick J. Jordan / Jens Thele / Ferris Bueller

Hyperparameters from Bayesian Optimization

Grid Search:
Systematic but stupid

Random Search:
Samples from given distributions

Q: Alternatives to GridSearch and Random Search?

A: State-of-the-art is Bayesian Optimization

Tree Parzen Estimator (TPE)
is based on surrogate objective function

A hyperparameter λ_i can be continuous, integer-valued or categorical

$$\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$$

Hyperparameter domain

$$\Lambda = (\Lambda_1, \Lambda_2, \ldots, \Lambda_n)$$

Lossfunction

$$\mathscr{L}(\lambda, \mathscr{D}_{\texttt{train}}, \mathscr{D}_{\texttt{test}})$$

Data

$\mathscr{D}_{\texttt{train}}$
$\mathscr{D}_{\texttt{test}}$

Objective from, e.g., k-fold cross-validation

$$f(\lambda) = \frac{1}{k} \sum_i \mathscr{L}(\lambda, \mathscr{D}^{(i)}_{\texttt{train}}, \mathscr{D}^{(i)}_{\texttt{test}})$$

Optimal hyperparameters

$$\lambda^\star = \texttt{argmin}_\lambda f(\lambda)$$

Model	**Hyperparameter optimization**	Implementation
Gradient Boosting	Random search: `max_features`, `min_samples_leaf`, `max_depth`, `learning_rate`, `n_estimators`	SCIKIT-LEARN
Random Forest	Random search: `min_samples_split`, `n_estimators`, `max_features`	SCIKIT-LEARN
Gaussian Process	MCMC sampling over hyperparameters	SPEARMINT
SVR	Random search: `C` and `gamma`	SCIKIT-LEARN
NuSVR	Random search: `C`, `gamma` and `nu`	SCIKIT-LEARN
k-nearest-neighbours	Random search: `n_neighbors`	SCIKIT-LEARN
Linear Regression	None	SCIKIT-LEARN
Ridge Regression	Random search: `alpha`	SCIKIT-LEARN

K. Eggensperher et al., *Efficient Benchmarking of Hyperparameter Optimizers via Surrogates* (2015)

Tree Parzen Estimator (TPE)

based on surrogate objective function

Generally: The surrogate function is a probabilistic MCMC that maps hyperparameters to a probability of a score on the objective function

(Bayesian updating of the prior)

Tree Parzen Estimator (TPE) is based on tree-structured Parzen density estimators

TPE is a generalist:
Can handle continuous, categorical, and conditional parameters

`Python example in lecture`

Recommended reading:
`https://sebastianraschka.com/Articles/2014_kernel_density_est.html`