

IFT 6268 Stats refresher

Roland Memisevic

September 10, 2013



Probabilities in AI

- ▶ Probabilities allow us to be explicit about uncertainties:
- ▶ Instead of representing *values*, we can “keep all options open” by defining a distribution over alternatives.
- ▶ Example: Instead of setting ‘ $x = 4$ ’, define all of: $p(x = 1), p(x = 2), p(x = 3), p(x = 4), p(x = 5)$
- ▶ Benefits:
 1. Robustness (let modules tell each other their whole state of knowledge)
 2. Measure of uncertainty (“errorbars”)
 3. Multimodality (keep ambiguities around)
- ▶ We can still express ‘ $x = 4$ ’ as a special case.



Random variable: “not random, not a variable”

- ▶ The only relevant property of a random variable is its distribution.
- ▶ $p(x)$ is a distribution if

$$p(x) \geq 0 \text{ and } \sum_x p(x) = 1$$

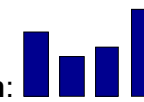
- ▶ Notational quirks:
 - ▶ The symbol p can be heavily overloaded. The argument decides. For example, in “ $p(x, y) = p(x)p(y)$ ” each p means something different!
 - ▶ Sometimes we write X for the RV and x for the values it can take on.
 - ▶ Another common notation is $p(X = x)$.
 - ▶ $\sum_x \dots$ refers to the sum over all values that x can take on.
- ▶ For continuous x , replace \sum by \int (up to some measure theoretic “glitches”, that we usually ignore in practice)
- ▶ Some prefer to use the term “density” or “probability density function (pdf)” to refer to continuous $p(\cdot)$.

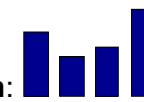


Some useful distributions (1d)


Discrete

- ▶ **Bernoulli:** $p^x(1 - p)^{1-x}$ where x is 0 or 1.



- ▶ **Discrete distribution:**  (also known as “multinoulli”)
- ▶ **Binomial, Multinomial:** Sum over Bernoulli/Discrete. (Sometimes “multinomial” is used to refer to a discrete distribution, too...)
- ▶ **Poisson:** $p(k) = \frac{\lambda^k \exp(-\lambda)}{k!}$

Continuous

- ▶ **Uniform:** 
- ▶ **Gaussian (1d):** $p(x) = \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right)$



How to represent discrete values

- ▶ A very useful way to represent a variable that takes one out of K values:
- ▶ As a K -vector with $(K - 1)$ 0's, and one 1 at position k

$$\mathbf{x} = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

- ▶ This is known as **one-of- K encoding**, **one-hot encoding**, or as **orthogonal encoding**.
- ▶ Note that we can interpret \mathbf{x} itself as a probability distribution.



Some useful distributions (1d)

- ▶ Using a one-hot encoding allows us to write the discrete distribution compactly as

$$p(\mathbf{x}) = \prod_k \mu_k^{x_k}$$

where μ_k is the probability for state k .

- ▶ This can greatly simplify calculations (see below).



Summarizing properties

- ▶ Any relevant properties of RVs are just properties of their distributions.
- ▶ Mean:

$$\mu = \sum_x p(x)x = E[x]$$

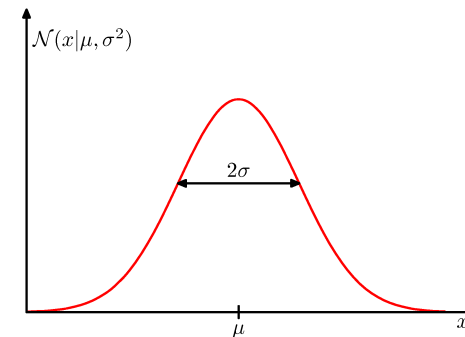
- ▶ Variance:

$$\sigma^2 = \sum_x p(x)(x - \mu)^2 = E[(x - \mu)^2]$$

- ▶ (Standard deviation: $\sigma = \sqrt{\sigma^2}$)



The Gaussian (1d)



$$p(x) = \mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right)$$



Multiple variables

- ▶ The **joint distribution** $p(x, y)$ of two variables x and y also satisfies

$$p(x, y) > 0 \text{ and } \sum_{x,y} p(x, y) = 1,$$

- ▶ Likewise, we can write

$$p(\mathbf{x}) > 0 \text{ and } \sum_{\mathbf{x}} p(\mathbf{x}) = 1,$$

for vector \mathbf{x}

- ▶ For discrete RVs, the joint is a *table* (or a higher dimensional *array*).
- ▶ Everything else stays the same.



Conditionals, marginals

- ▶ Everything one may want to know about a random vector can be derived from the joint distribution.

- ▶ **Marginal distributions:**

$$p(x) = \sum_y p(x, y) \text{ and } p(y) = \sum_x p(x, y)$$

- ▶ Imagine collapsing tables.

- ▶ **Conditional distributions:**

$$p(y|x) = \frac{p(x, y)}{p(x)} \text{ and } p(x|y) = \frac{p(x, y)}{p(y)}$$

- ▶ Think of conditional as a family of distributions, “indexed” by the conditioning variable. (We could write $p(y|x)$ also as $p_x(y)$).



Summarizing properties, correlation

- ▶ Mean:

$$\mu = \sum_{\mathbf{x}} p(\mathbf{x})\mathbf{x} = E[\mathbf{x}]$$

- ▶ Covariance:

$$\text{cov}(x_i, x_j) = E[(x_i - \mu_i)(x_j - \mu_j)]$$

- ▶ Covariance matrix:

$$\Sigma_{ij} = \text{cov}(x_i, x_j) \quad (\Sigma = \sum_{\mathbf{x}} p(\mathbf{x})(\mathbf{x} - \mu)(\mathbf{x} - \mu)^T)$$

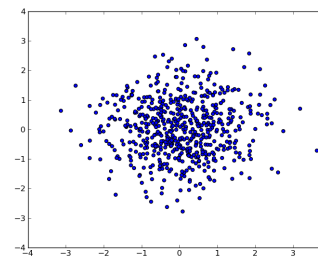
- ▶ The correlation coefficient:

$$\text{corr}(x_i, x_j) = \frac{\text{cov}(x_i, x_j)}{\sqrt{\sigma_i^2 \sigma_j^2}}$$

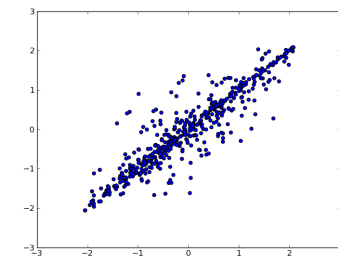
- ▶ Two variables for which the covariance is zero are called **uncorrelated**.



Correlation example



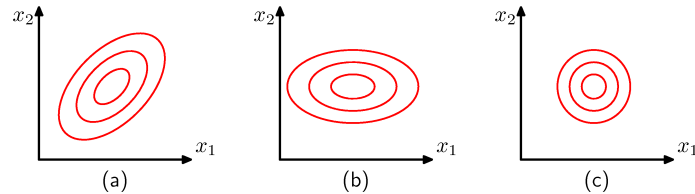
uncorrelated



correlated



The multivariate Gaussian



$$p(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$$



A fundamental formula

$$p(x|y)p(y) = p(x, y) = p(y|x)p(x)$$

- ▶ This can be generalized to more variables (“chainrule of probability”).
- ▶ A special case is **Bayes’ rule**:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$



Independence and conditional independence

- ▶ Two RVs are called **independent**, if

$$p(x, y) = p(x)p(y)$$

- ▶ Captures our intuition of “dependence”. In particular, note that this definition implies

$$p(y|x) = p(y)$$

- ▶ Independence implies uncorrelatedness, but not vice versa!
- ▶ Related: Two RVs are called **conditionally independent**, given a third variable z , if

$$p(x, y|z) = p(x|z)p(y|z)$$

- ▶ (Note that these concepts are just a property of the joint.)



Independence is useful

- ▶ Say, we have some variables, x_1, x_2, \dots, x_K
- ▶ Even just defining their joint (let alone doing computations with it) is hopeless for large K !
- ▶ But what if all the x_i are independent?
- ▶ Then we need to specify just K probabilities, because *the joint is the product*.
- ▶ A more sophisticated version of this idea, using *conditional independence*, is the basis for the area of *graphical models*.



Maximum likelihood

- ▶ Another useful property of independence.
- ▶ Task: Given a set of data points

$$(x_1, \dots, x_N)$$

build a model of the data-generating process.

- ▶ Approach: Fit a parametric distribution $p(x; \mathbf{w})$ with some parameters \mathbf{w} to the data.
- ▶ How? Maximize the probability of “seeing” the data under your model!



Maximum likelihood

- ▶ This is easy if examples are independent and identically distributed (“iid”):

$$p(\mathbf{x}_1, \dots, \mathbf{x}_N; \mathbf{w}) = \prod_i p(\mathbf{x}_i; \mathbf{w})$$

- ▶ Instead of maximizing probability, we may maximize log-probability, because the log function is monotonic.
- ▶ So we may maximize:

$$L(\mathbf{w}) := \log \prod_i p(\mathbf{x}_i; \mathbf{w}) = \sum_i \log p(\mathbf{x}_i; \mathbf{w})$$

- ▶ Thus each example \mathbf{x}_i contributes an additive component to the objective.



Gaussian example

- ▶ What is the ML-estimate of the mean of a Gaussian?
- ▶ We need to maximize

$$L(\mu) = \sum_i \log p(x_i; \mu) = \sum_i \left(-\frac{1}{2\sigma^2} (x_i - \mu)^2 \right) - \text{const.}$$

- ▶ The derivative is:

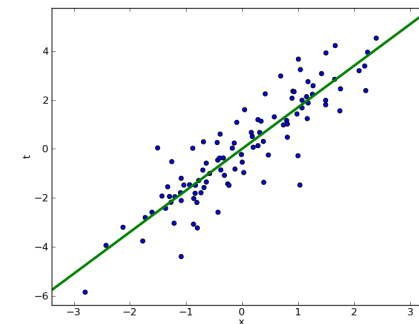
$$\frac{\partial L(\mu)}{\partial \mu} = \frac{1}{\sigma^2} \sum_i (x_i - \mu) = \frac{1}{\sigma^2} \left(\sum_i x_i - N\mu \right)$$

- ▶ By setting to zero, we get:

$$\mu = \frac{1}{N} \sum_i x_i$$



Linear regression



$\mathbf{x} \rightarrow \mathbf{t}$

- ▶ Given two *real-valued* observations \mathbf{x} and \mathbf{t} , learn to predict \mathbf{t} from \mathbf{x} .
- ▶ This is a *supervised learning* problem.



Linear regression

- ▶ We can define linear regression as a probabilistic model, if we make the following assumption:

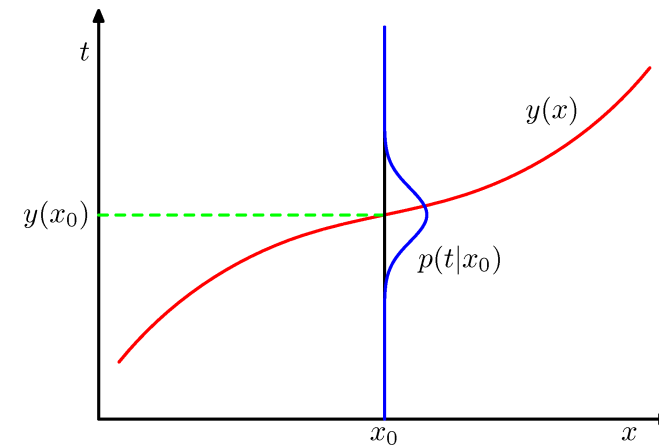
$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon$$

- ▶ In words, we assume there is a true, underlying function $y(\mathbf{x}, \mathbf{w})$, and the function values we observe are corrupted by *additive Gaussian noise*.
- ▶ Thus

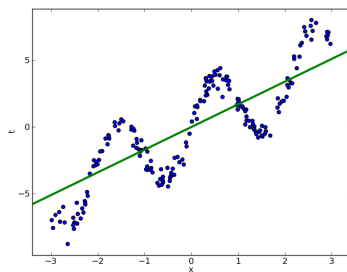
$$p(t|\mathbf{x}; \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \sigma^2)$$



Linear regression



Noise vs. dependencies we don't care about



- ▶ Actually, linear regression can work fine also with highly non-Gaussian noise.



Linear regression

- ▶ To fit the conditional Gaussian, given training data $\mathcal{D} = \{(\mathbf{x}_n, t_n)\}_{n=1}^N$, we make the iid assumption and get:

$$p(\mathcal{D}) = \prod_n \mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \sigma^2)$$

- ▶ Using monotonicity of the log, we may again maximize the log-probability (or minimize its negation):

$$\text{minimize } \sum_{n=1}^N (t_n - \mathbf{w}^T \mathbf{x}_n)^2 + \text{const.}$$



Least squares

- ▶ To optimize with respect to \mathbf{w} , we differentiate:

$$\frac{\partial E}{\partial \mathbf{w}} = - \sum_{n=1}^N (t_n - \mathbf{w}^T \mathbf{x}_n) \mathbf{x}_n^T$$

- ▶ Setting the derivative to zero:

$$0 = - \sum_{n=1}^N t_n \mathbf{x}_n^T + \mathbf{w}^T \left(\sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T \right)$$

yields the solution

$$\mathbf{w} = \left(\sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T \right)^{-1} \left(\sum_{n=1}^N t_n \mathbf{x}_n^T \right)$$

- ▶ (It can be instructional to write down the case for 1-d inputs, if this confuses you)



Least squares

- ▶ We can write this more compactly with the following definitions:

$$\mathbf{t} = \begin{pmatrix} t_1 \\ \vdots \\ t_N \end{pmatrix}$$

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \end{pmatrix}$$

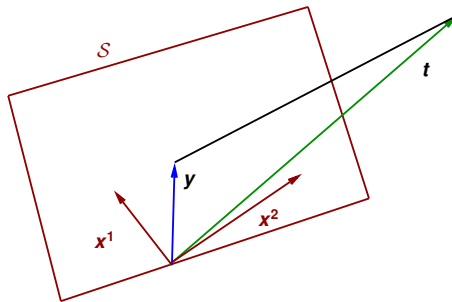
This allows us to write the the solution as

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$$

“The normal equations”.



Geometrical interpretation



- ▶ We can think of the squared error $\sum_n (t_n - \mathbf{w}^T \mathbf{x}_n)^2$ as the squared norm of the difference between two N -dimensional vectors \mathbf{t} and \mathbf{y} , containing all training-outputs and model-predictions.
- ▶ The vector $\mathbf{y} (= \mathbf{X}\mathbf{w})$ has to lie in the space \mathcal{S} spanned by the *columns* \mathbf{x}^i of \mathbf{X} .
- ▶ It is exactly the *orthogonal projection* of \mathbf{t} onto \mathcal{S}



Linear classification

$$\mathbf{x} \rightarrow t$$

- ▶ A prediction task, where the outputs, t , are discrete (that is, they can take on one of K values, $\mathcal{C}_1, \dots, \mathcal{C}_K$), is called *classification*.
- ▶ Like regression, this is a *supervised learning* problem.



(Multi-class) logistic regression

- ▶ Logistic regression defines a probabilistic model over classes given inputs as follows:

$$p(C_k|\mathbf{x}) = \frac{\exp(\mathbf{w}_k^T \mathbf{x}_n)}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \mathbf{x}_n)}$$

where $\mathbf{w}_1, \dots, \mathbf{w}_K$ are parameters.

- ▶ The exp-function ensures positivity, and the normalization that the outputs sum to one.
- ▶ (In practice, one usually adds constant “bias”-terms inside the exp’s)



Multi-class logistic regression

- ▶ Represent discrete one-hot labels row-wise in a matrix \mathbf{T} , like we did before for continuous vectors.
- ▶ The negative log-likelihood cost, assuming iid training data, can then be written

$$\begin{aligned} E(\mathbf{W}; \mathcal{D}) &= -\log \prod_n p(\mathbf{t}_n | \mathbf{x}_n) \\ &= -\log \prod_n \prod_k p(C_k | \mathbf{x}_n)^{t_{nk}} \\ &= -\sum_n \sum_k t_{nk} \log p(C_k | \mathbf{x}_n) \\ &= -\sum_n \sum_k t_{nk} (\mathbf{w}_k^T \mathbf{x}_n - \log \sum_{j=1}^K \exp(\mathbf{w}_j^T \mathbf{x}_n)) \\ &= -\sum_n (\mathbf{w}_{t_n}^T \mathbf{x}_n - \log \sum_{j=1}^K \exp(\mathbf{w}_j^T \mathbf{x}_n)) \end{aligned}$$



Multi-class logistic regression

- ▶ In contrast to linear regression, there is no closed-form solution for \mathbf{W} .
- ▶ But one can use gradient-based optimization to minimize $E(\mathbf{W}; \mathcal{D})$ iteratively.
- ▶ The gradient with respect to each parameter-vector \mathbf{w}_k is

$$\begin{aligned} \frac{\partial E(\mathbf{W}; \mathcal{D})}{\partial \mathbf{w}_k} &= -\sum_n t_{nk} \mathbf{x}_n - \frac{\exp(\mathbf{w}_k^T \mathbf{x}_n)}{\sum_j \exp(\mathbf{w}_j^T \mathbf{x}_n)} \mathbf{x}_n \\ &= \sum_n (p(C_k | \mathbf{x}_n) - t_{nk}) \mathbf{x}_n \end{aligned}$$

- ▶ It can be shown that $E(\mathbf{W}; \mathcal{D})$ is convex, so there are no local minima.



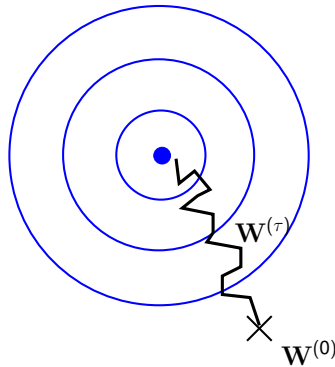
Learning with stochastic gradient descent

$$\mathbf{W}^{(\tau+1)} = \mathbf{W}^{(\tau)} - \eta \frac{\partial E_n}{\partial \mathbf{W}}$$

- ▶ Here, τ denotes the iteration number, and E_n is the cost contributed by the n^{th} training case (one term in the sum over n).
- ▶ Parameters are initialized to some random starting value $\mathbf{W}^{(0)}$.
- ▶ η is called **learning rate**, and it is typically set to a small real value (such as, $\eta = 0.001$). It may be reduced as learning progresses.
- ▶ It is convenient to think of \mathbf{W} as a vector not a matrix when doing learning. (Think of it in “vectorized” form: $\text{vec}(\mathbf{W})$)



Learning with stochastic gradient descent



- ▶ Since the algorithm visits one training-case at a time, it will jitter around an idealized “average path” towards the optimum.
- ▶ That’s why it’s called “stochastic” gradient descent.
- ▶ One could use the gradient of the whole sum instead but that is often slower (because of redundancies in the data).

Random variables and information

- ▶ “Probabilities allow us to be explicit about uncertainty”. So how can we *measure* uncertainty?
- ▶ Idea: Define the *information content*

$$\log\left(\frac{1}{p(x)}\right) = -\log(p(x))$$

contained in a random event.

- ▶ The information content is additive for independent events.
- ▶ So if we use \log_2 and fair coin tosses, then the information content is measured in *bits* and it exactly fits our intuition.

The “logsumexp”-trick

- ▶ Expressions like

$$\frac{\exp(\mathbf{w}_k^T \mathbf{x}_n)}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \mathbf{x}_n)}$$

are very common but *highly unstable*, because the “exp” in the denominator can cause an under- or overflow.

- ▶ Never compute sums $\sum_i \exp(a_i)$ naively.
- ▶ Add a constant A to each argument in all exp’s, so that even the largest argument is small; then undo the operation after computing the sum!
- ▶ Many software packages supply a convenience function “logsumexp” for this purpose:

logsumexp

$$\text{logsumexp}(a_1, \dots, a_K) = \log\left(\sum_i \exp(a_i + A)\right) - A$$

with $A = -(\max_i a_i)$

Entropy

- ▶ To measure uncertainty, we define the entropy

$$H(X) = -\sum_x p(x) \log p(x)$$

which is the average information content.

- ▶ For continuous RV:

$$H(X) = -\int_x p(x) \log p(x) dx$$

- ▶ The more uniform, the more uncertain. The more “peaky”, the more certain.
- ▶ Question: Which probability distribution has maximum entropy, given mean and (co)variance(s)?

KL divergence

- ▶ Closely related to entropy is the Kullback-Leibler divergence (KL divergence) (or “relative entropy”):

$$\text{KL}(p||q) = \sum_x p(x) \log \left(\frac{p(x)}{q(x)} \right)$$

- ▶ The KL divergence measures the dissimilarity between two distributions.
- ▶ It is not symmetric.
- ▶ Maximum likelihood learning amounts to minimizing the KL divergence between the model distribution and the empirical distribution over the observed training data (exercise).



Mutual Information

- ▶ The mutual information between two random variables x, y with joint density $p(x, y)$ is defined as

$$\text{MI}(x, y) = \sum_{x,y} p(x, y) \log \left(\frac{p(x, y)}{p(x)p(y)} \right)$$

- ▶ It is the KL divergence between $p(x, y)$ and the joint of two perfectly independent random variables (with marginals $p(x)$ and $p(y)$).
- ▶ Thus, MI measures the *dependence* between x and y .
- ▶ It is nonnegative, and it is zero iff x and y are independent, in other words iff $p(x, y) = p(x)p(y)$.



Frequentist – Bayesian

- ▶ Probability theory tells us how to *calculate* with probabilities.
- ▶ As scientists, we may ask how to *interpret* a probabilistic expression, such as

$$p(x = 1) = 0.7$$

- ▶ There are two common interpretations:
 1. *Frequentist*: “The *relative frequency* of x in a (possibly infinite) population of trials is 0.7”
 2. *Bayesian*: “*I believe* that x is 1 with certainty 0.7”
- ▶ The Bayesian view used to be contentious because it is less intuitive. But it gives us the freedom to turn model parameters into random variables. And it is now an established view in ML.



Reading

- ▶ A good introduction to most of the concepts discussed in this class can be found in:
Pattern Recognition and Machine Learning. C. Bishop. Springer, 2006.
- ▶ Most illustrations in this presentation are from that book.

