Maximum Likelihood Modeling

Machine Learning I, Week 3

N. Schraudolph

Bayes' Rule in Machine Learning

$$P(M \mid D) \propto P(D \mid M)\, P(M)$$

Overall goal: maximize **posterior** (= find the most plausible model M to explain data D)

today: how to formulate **likelihood** for different kinds of model and data

↓

next 2 weeks: how to maximize likelihood

in 3-4 weeks: the role of **priors** in machine learning

(**Note**: ML is not always done strictly according to Bayes' Rule)

Maximum Likelihood Approach

$$P(M \mid D) \propto P(D \mid M)\, P(M)$$

- if we don't have any *a priori* reason to prefer one M over another, assume a **flat (uniform) prior**, P(M) = *const*.
- this is also called the (maximally) **uninformative prior,** since it does not give us any prior information about M
- maximizing the posterior then becomes equivalent to maximizing the likelihood, P(D|M)
- for numerical reasons, we always use the **log-likelihood**, log P(D|M). Since log is monotonically increasing, this is equivalent: the maxima remain in the same position.

Data Independence Assumption

- D typically consists of repeated observations of the same natural process. Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots\}$, $\mathbf{x}_k \in R^n$
- key assumption: $\mathbf{x}_k$ are samples from a **Bernoulli process** = they are **i.i.d.** (independent & identically distributed). Allows us to decompose the likelihood:

$$P(D|M) = \prod_k P(\mathbf{x}_k|M)$$

$$\Leftrightarrow \log P(D|M) = \sum_k \log P(\mathbf{x}_k|M)$$

- this makes the likelihood much easier to formulate
- counterexample: in a **Markov process,** the distribution of sample $\mathbf{x}_t$ depends on the previous sample $\mathbf{x}_{t-1}$ (but not $\mathbf{x}_{t-2}$)

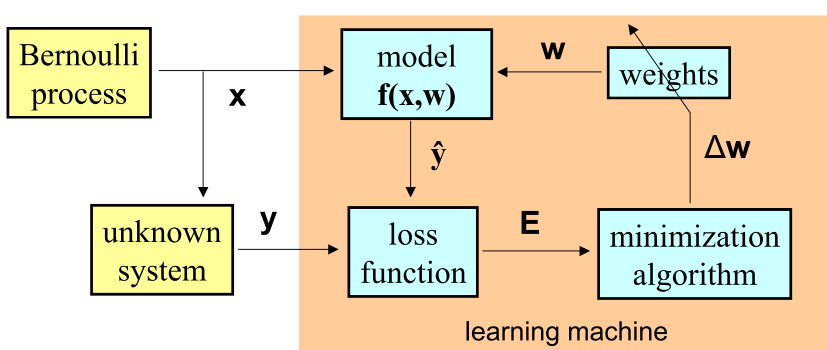
Formalizing the Model

- many useful kinds of model can be written as **parametric function f(x;w)** of inputs **x** and parameters ("weights") **w**
- by adjusting the weights **w**, we can modify how **f** responds to a given input $\mathbf{x}_k$ – *i.e.,* we can adjust the model
- we'll quantify how well the model performs by means of a **loss function** E(**w**). Learning now means **minimizing** the loss, *i.e.,* finding the best weights, arg $\min_{\mathbf{w}}$ E(**w**)
- the maximum likelihood (ML) approach is now implemented by using the negative log-likelihood as loss function:

$$E(\mathbf{w}) = -\log P[D|\mathbf{f}(\mathbf{x};\mathbf{w})] = -\sum_k \log P[\mathbf{x}_k|\mathbf{f}(\mathbf{x}_k;\mathbf{w})]$$

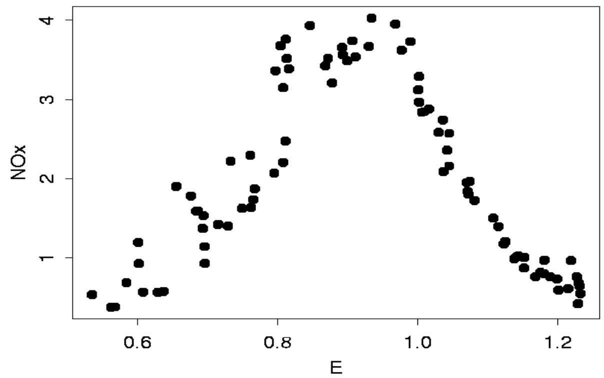
Modeling Tasks: Regression

In regression, we are trying to learn as **target** the response **y** of an unknown system to **input x**. The output $\hat{\mathbf{y}}_k = \mathbf{f}(\mathbf{x}_k;\mathbf{w})$ of our model is a prediction of the system's response to $\mathbf{x}_k$.



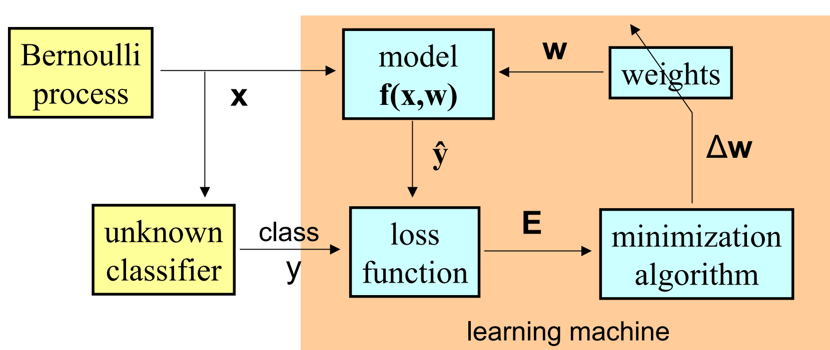
Regression: Simple Example

Relative concentration of NO and NO_2 in exhaust fumes as a function of the richness of the ethanol/air mixture burned in a car engine:



Modeling Tasks: Classification

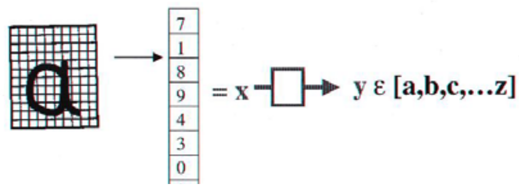
In classification tasks, the response is a discrete **class label** associated with the input. $\hat{\mathbf{y}}_k = \mathbf{f}(\mathbf{x}_k;\mathbf{w})$ predicts the likelihood that point $\mathbf{x}_k$ is in class 1, 2, 3, …



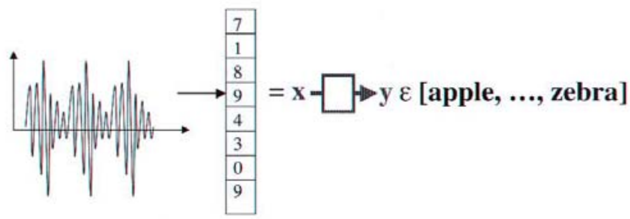
Classification: Examples

email filtering: x ε [a-z]+ → y ε [important, spam]

character recognition:

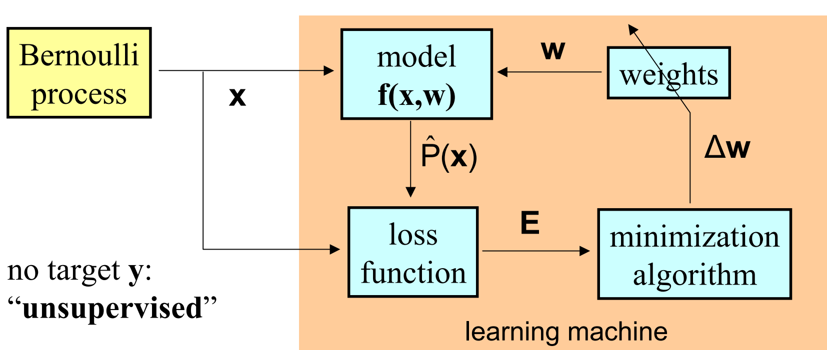


speech recognition:



Modeling Tasks: Density Estimation

In density estimation, there is no target **y**; we want to learn about the Bernoulli process that produces **x**. Our model f($\mathbf{x}_k$;**w**) learns to predict the density P(**x**) at point $\mathbf{x}_k$.



no target **y**:
"**unsupervised**"

ML Loss for Density Estimation

We will now derive **ML loss functions** for the three modeling tasks: regression, classification, and density estimation. Recall that the ML loss is the negative log-likelihood:

$$E(\mathbf{w}) = -\log \prod_k P(\mathbf{x}_k|M) = -\sum_k \log P[\mathbf{x}_k|\mathbf{f}(\mathbf{x}_k;\mathbf{w})]$$

The simplest case is **density estimation**: here by definition the likelihood is just our model's output: $P[x_k|f(\mathbf{x}_k;\mathbf{w})] = f(\mathbf{x}_k;\mathbf{w})$. The ML loss for density estimation is therefore simply

$$E(\mathbf{w}) = -\sum_k \log f(\mathbf{x}_k;\mathbf{w}),$$

known as the **entropy** of $f(\mathbf{x};\mathbf{w})$ under the given density of $\mathbf{x}$.

ML Density Estimation: Example

Fit Gaussian density with parameters $\mathbf{w} = (\mu,\sigma^2)$ to n points x_k.

$$f(x_k;\mathbf{w}) = N(x_k;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x_k-\mu}{\sigma}\right)^2\right]$$

$$E(\mathbf{w}) = -\sum_k \log \mathbf{f}(x_k;\mathbf{w}) = \frac{1}{2}\sum_k \left(\frac{x_k-\mu}{\sigma}\right)^2 + n\log(\sigma) + \text{const.}$$

At optimum $\mathbf{w}^*$, gradient $\partial E(\mathbf{w})/\partial\mu = -\sum_k \left(\frac{x_k-\mu}{\sigma}\right) = 0,$

so $\mu^* = \frac{1}{n}\sum_k x_k$. **Homework**: show that $\sigma^2{}^* = \frac{1}{n}\sum_k (x_k - \mu)^2$.

ML Loss for Classification

For classification, our model produces a vector of class probabilities, so for class c the likelihood is the c^{th} component of $\mathbf{f}$: $P[c|\mathbf{f}(\mathbf{x};\mathbf{w})] = f_c(\mathbf{x};\mathbf{w})$. For a system that only indicates the correct class y_k as target, the loss is then

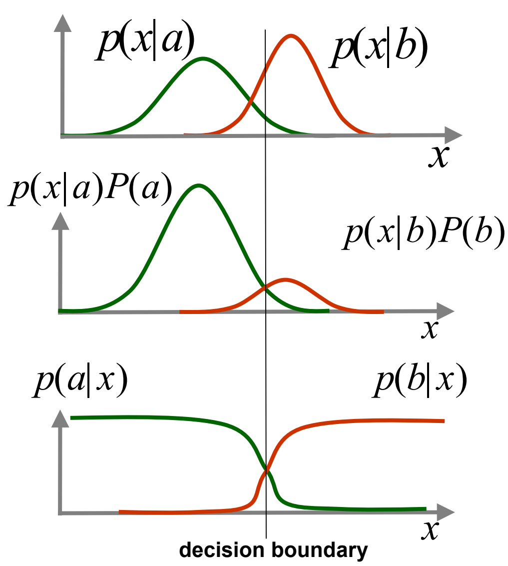
$$E(\mathbf{w}) = -\sum_k \log P[y_k|\mathbf{f}(\mathbf{x}_k;\mathbf{w})] = -\sum_k \log f_{y_k}(\mathbf{x}_k;\mathbf{w}).$$

This can be generalized to systems that provide a full target vector $\mathbf{y}_k$ of posterior class probabilities:

$$E(\mathbf{w}) = -\sum_k \mathbf{y}_k^T \log \mathbf{f}(\mathbf{x}_k;\mathbf{w}),$$

known as the **cross-entropy** between $f(\mathbf{x};\mathbf{w})$ and $\mathbf{y}$.

ML Classification: Example



Bayesian classifier with Gaussian class densities; parameters to learn:

$\mathbf{w} = [\mu_a,\sigma_a,P(a), \mu_b,\sigma_b,P(b)]$

Take care to distinguish the two levels at which Bayes' Rule is being used here:

1. to classify each datum $\mathbf{x}_k$;
2. to learn a good classifier.

ML Loss for Regression

For regression tasks we have a problem: we need the likelihood $P[\mathbf{y}_k|\mathbf{f}(\mathbf{x}_k;\mathbf{w})]$, but our model gives an estimate $\hat{\mathbf{y}}_k = \mathbf{f}(\mathbf{x}_k;\mathbf{w})$ for the target $\mathbf{y}_k$. We can turn it into a likelihood by the expedient of adding zero-mean Gaussian **noise** $\varepsilon \sim N(\mathbf{0}, \sigma^2\mathbf{I})$ to the model's output. This gives

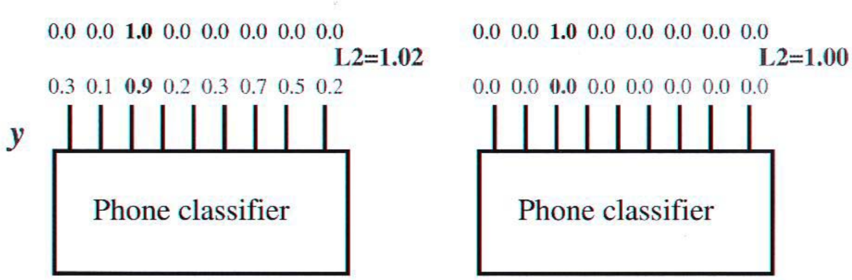
$$P[\mathbf{y}_k|\mathbf{f}(\mathbf{x}_k;\mathbf{w})+\varepsilon] \propto \exp\{-[\mathbf{y}_k - \mathbf{f}(\mathbf{x}_k;\mathbf{w})]^T[\mathbf{y}_k - \mathbf{f}(\mathbf{x}_k;\mathbf{w})]/(2\sigma^2)\}$$

$$E(\mathbf{w}) = -\sum_k \log P[\mathbf{y}_k|\mathbf{f}(\mathbf{x}_k;\mathbf{w})+\varepsilon] \propto \sum_k \| \mathbf{y}_k - \mathbf{f}(\mathbf{x}_k;\mathbf{w})\|^2,$$

known as the **sum-squared error** loss function.

Home-Brewed Loss Functions

It may be tempting to simply minimize some intuitive idea of "distance" between model and target system. This **can cause unexpected problems**. Deriving the loss from a likelihood helps avoid problematic choices of loss function, such as *e.g.* sum-squared loss for a classification problem:



ML Loss Functions: Summary

A maximum likelihood (ML) loss function is the negative log-likelihood of the data assuming i.i.d. sampling. For a parametric function $\mathbf{f}(\mathbf{x};\mathbf{w})$ as our model M, we have

$$E(\mathbf{w}) = -\log P[D|\mathbf{f}(\mathbf{x};\mathbf{w})] = -\sum_k \log P[\mathbf{x}_k|\mathbf{f}(\mathbf{x}_k;\mathbf{w})]$$

Regression: sum-squared loss, $E(\mathbf{w}) = \sum_k \| \mathbf{y}_k - \mathbf{f}(\mathbf{x}_k;\mathbf{w})\|^2$

Classification: cross-entropy loss, $E(\mathbf{w}) = -\sum_k \mathbf{y}_k^T \log \mathbf{f}(\mathbf{x}_k;\mathbf{w})$

Density Estimation: entropic loss, $E(\mathbf{w}) = -\sum_k \log f(\mathbf{x}_k;\mathbf{w})$

where $\mathbf{x}$ = input, $\mathbf{y}$ = target, $\mathbf{w}$ = weights (parameters to learn).