GHENT UNIVERSITY

Lecture Notes

E016350: Artificial Intelligence

Learning probabilistic models

Aleksandra Pizurica

Spring 2024

Contents

Disclaimer*: These lecture notes were written by Prof. Aleksandra Pizurica to accompany the slides of the course E016350: Artificial Intelligence, facilitating their understanding. The lecture notes are not meant to be self-contained, and do not cover all the study material in the course. They are by no means meant to replace the recommended textbook and do not necessarily cover all the relevant aspects that are presented in the slides and explained in the lectures. Some sections are adapted from the book of S. Russel and P. Norvig: Artificial Intelligence: A Modern Approach.*

1 Bayesian and statistical machine learning

We studied so far machine learning and probabilistic reasoning as two important domains of AI, each with their own motivations, tasks and sets of techniques. Now we bring them together. On the one hand, we put machine learning into probabilistic reasoning in order to *learn* our probabilistic theories and models from experience (i.e., from data). On the other hand, we also bring probabilistic reasoning into machine learning to make the learning process more powerful, less susceptible to various imperfections in the data, less prone to overfit the data or base the results and decisions on a wrong, pre-selected model.

Bayesian machine learning views *learning* as a form of *uncertain reasoning from observations*. We can also say that it devises *models* to represent the *uncertain world* [3]. A key benefit that it brings to machine learning is a sound framework where we do not necessarily choose one *single* hypothesis (one single model) from a given hypothesis space (or model class) but *take each into account with its own probability.* Even if we resort to taking one hypothesis, we will do that by making use of statistical distributions over the various hypotheses and data, which will give us a more robust approach than with deterministic optimization techniques alone. Such a framework provides general solutions to dealing with noise, overfitting and ambiguities regarding what is an optimal prediction. It also copes well with the fact that an AI agent can rarely be certain about which model of the world is correct, yet it must make decisions and actions, and often in real time.

The term **Bayesian machine learning** strictly speaking refers to the class of techniques where we use priors over the models and make the prediction based on all of them. We will use the term **statistical machine learning** to refer to a wider class of approaches where we use probability distributions or statistical distributions of the data, but not necessarily making predictions based on all possible models from a given class and not necessarily using priors for these models. We adopt this terminology from [3].

As in the theory of learning, we deal with *data* and *hypotheses* but now they will be treated as *random variables* (*r.v.*s) or their realizations. The data are **evidence** – instantiations of some (or all) domain r.v.s, and hypotheses are **probabilistic models** (of how the domain "works").

1.1 Bayesian learning

Bayesian learning calculates the probability of each hypothesis, given the data. Predictions are made using *all* the hypotheses weighted by their probabilities rather than using a single "best" hypothesis. Learning becomes probabilistic inference!

Let the random variable $\mathbf{D}$ represent all the data, with observed value $\mathbf{d}$. The (posterior) probability of the hypothesis h_j given data is:

$$\underbrace{P(h_j \mid \mathbf{d})}_{\substack{\text{posterior prob.}\\ \text{of hypotheis}}} = \alpha \underbrace{P(\mathbf{d} \mid h_j)}_{\text{likelihood}} \underbrace{P(h_j)}_{\substack{\text{hypothesis}\\ \text{prior}}} \tag{1}$$

The two key components are the **likelihood** of the data under each hypothesis $P(\mathbf{d} \mid h_j)$ and the **hypothesis prior** $P(h_i)$. Prediction about some unknown quantity X are now made as:

$$\mathbf{P}(X|\mathbf{d}) = \sum_j \mathbf{P}(X|h_j)P(h_j|\mathbf{d}) \tag{2}$$

which is a weighted average of the predictions of the individual hypotheses $\mathbf{P}(X|h_j)$, with weighting factors $P(h_j|\mathbf{d})$.

If the data are *independent identically distributed* (i.i.d), we have that

$$P(\mathbf{d}|h_j) = \prod_i P(d^{(i)}|h_j) \tag{3}$$

Remember that observations are i.i.d. if each example has the same prior probability distribution and is independent of previous examples. In our notation, this means:
$\mathbf{P}(D^{(i)}) = \mathbf{P}(D^{(i+1)}) = \mathbf{P}(D^{(i+2)}) = \ldots$ and
$\mathbf{P}(D^{(i)}|D^{(i-1)}, D^{(i-2)}, \ldots) = \mathbf{P}(D^{(i)})$

1.1.1 "Surprise Candy" use case

.

We illustrate the concepts of statistical learning on the following example from [3]:
Our favorite surprise candy comes in two flavors: cherry (yum) and lime (ugh). The manufacturer has a peculiar sense of humor and wraps each piece of candy in the same opaque wrapper, regardless of flavor. The candy is sold in very large bags, of which there are known to be five kinds—again, indistinguishable from the outside:

h_1: 100% cherry
h_2: 75% cherry + 25% lime
h_3: 50% cherry + 50% lime
h_4: 25% cherry + 75% lime
h_5: 100% lime

Given a new bag of candy, the random variable H (hypothesis) denotes the type of the bag, with possible values $h \in \{h_1, \ldots h_5\}$. As the pieces of candy are opened an inspected, data are revealed $D^{(1)}, D^{(2)}, \ldots D^{(N)}$, where each $D^{(i)}$ is a random variable with possible values cherry *and* lime. *The basic task faced by the AI agent is to predict the flavor of the next piece of candy.*

1.1.2 Bayesian learning example

We will now employ Bayesian learning to predict the flavor of the $(N+1)$-th candy given the N opened ones. Hence, we apply (2), where the random variable to be predicted is now $X = D^{(N+1)}$:

$$\mathbf{P}(D^{(N+1)}|\mathbf{d}) = \sum_j \mathbf{P}(D^{(N+1)}|h_j)P(h_j|\mathbf{d}) \tag{4}$$

Since the candy is sold in very large bags, it is reasonable to assume that the data are i.i.d. (although we don't rewrap and return the opened candy back into the bag). Thus the likelihood is the product of the likelihoods for each separate candy as was given in (5). We still need the prior probabilities of h_j's. Suppose this prior distribution is known (e.g., was made pubic in the advertisements of the manufacturer) to be $\mathbf{P}(h_1, \ldots, h_5) = \langle 0.1, 0.2, 0.4, 0.2, 0.1 \rangle$.

With this, we have all what is needed to calculate the posterior probability of each h_i given the observed data $\mathbf{d}$ and, using that, also to predict the taste of the next candy. In Fig. 1, the posterior probabilities of the hypotheses and the probability of a given flavor (lime) of the next

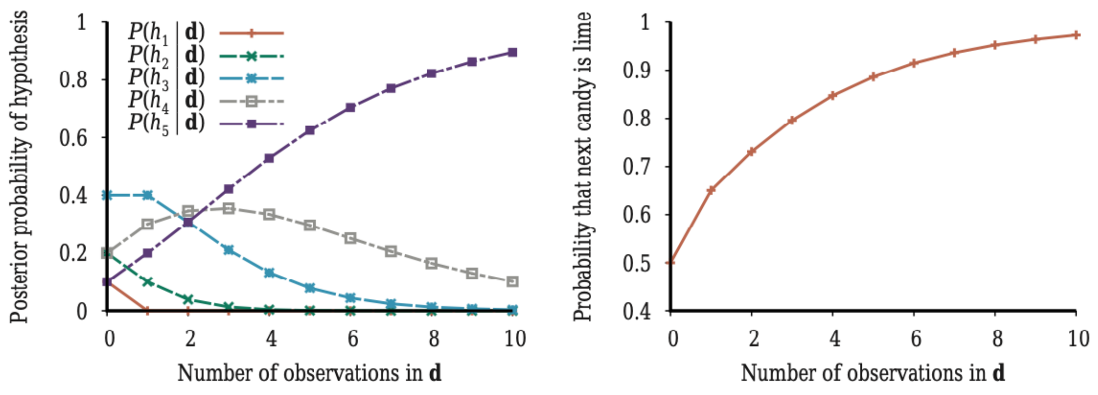

Figure 1: Bayesian learning example on the Surprise Candy case **Left**: Posterior probabilities $P(h_j|d^{(1)}, \dots d^{(N)})$. The number of observations N ranges from 1 to 10 and *each* observation is a *lime* candy. **Right**: Bayesian prediction $P(D^{(N+1)} = lime|d^{(1)}, \dots, d^{(N)})$. Figure from [3].

candy are shown for the different numbers of observations $N = 1, \dots, 10$, and assuming that each observation was *lime*.

The posterior probability plots on the left of Fig. 1 were obtained using Eq (1), where the hypothesis prior was as given above $\mathbf{P}(h_1, \dots, h_5) = \langle 0.1, 0.2, 0.4, 0.2, 0.1 \rangle$, and where we used the i.i.d. assumption to express the likelihood as in Eq (5). For example, with $N = 10$ and all ten being *lime* observations, the data likelihood under the hypothesis h_3 is

$$P(\mathbf{d}|h_3) = \prod_{i=1}^{10} P(D^{(i)} = lime|h_3) = (0.5)^{10} \tag{5}$$

Observe that the initial values of the probabilities of the hypotheses (for $N = 0$) are their prior probabilities, so the probability for h_3 is initially the largest one. Already after the first candy is opened, the posterior probability of h_1 ("all cherry's") drops to zero, and h_3 is still the most probable hypothesis. After two candy's are open and both are lime, h_4 becomes the most probable, but h_5 is now only a little behind, its probability is increasing quickly. After ten all-lime candy's, h_5 is overwhelmingly the most probable hypothesis.

The probabilities of the prediction shown on the right Fig. 1 are calculated using Eq (2), (or concretely Eq (4), where $X = D^{(N+1)}$). There we use the already calculated posterior probabilities of the hypotheses $P(h_j|\mathbf{d})$, and $\mathbf{P}(D^{(N+1)}|h_j)$ is given by the problem description (e.g., $\mathbf{P}(D^{(N+1)}|h_1) = \langle 1, 0 \rangle$ and $\mathbf{P}(D^{(N+1)}|h_2) = \langle 0.75, 0.25 \rangle$). The probability plot agrees with what we would expect – the predicted probability that the next candy is lime is increasing monotonically toward 1.

1.2 Maximum a Posteriori and Maximum-Likelihood learning

The previous example illustrated a very important characteristic of the Bayesian learning: the *Bayesian prediction eventually (after sufficient number of observations) agrees with the true hypothesis*. What's nice is that this behavior does not depend on specifying a very accurate prior as long as it is reasonable enough – for any fixed prior that does not rule out the true hypothesis, the posterior probability of any false hypothesis will, under certain technical conditions, eventually vanish [3]. This is because it is not likely to generate repeatedly data that are "uncharacteristic" (they will be in negligible amounts in large datasets). Moreover, Bayesian prediction is *optimal regardless of whether the dataset is small or large.* Under a given prior, any other prediction will on the average be less correct.

While this optimality holds in theory, we might not always be able to use full Bayesian learning in practice. When the hypothesis space is very large this approach may be intractable. We then need to resort to approximate or simplified methods, and two common approaches are

- **Maximum a Posteriori** (**MAP**) learning:

$$h_{MAP} = \arg\max_{h \in \mathcal{H}} P(h|\mathbf{d}) = \arg\max_{h \in \mathcal{H}} P(\mathbf{d}|h)P(h)$$

- **Maximum-Likelihood** learning:

$$h_{ML} = \arg\max_{h \in \mathcal{H}} P(\mathbf{d}|h)$$

MAP learning (or MAP *estimation*) makes predictions based on a single most probable hypothesis. It is a commonly adopted approach in science. Maximum-likelihood approach can be seen as a simplification of the MAP learning by imposing a uniform prior on h. It is very common in statistics because many researchers distrust the subjective nature of hypothesis priors. Maximum-likelihood learning (also called maximum-likelihood estimation) is a good approximation for Bayesian and MAP learning with large data sets. This is because when the data set is large the prior is less important (evidence from data is strong enough to "swamp" the prior distribution) [3].

2 Learning with complete data

We address now the task of learning a probabilistic model from the data. This task is called **density estimation**[1] and is a form of *unsupervised learning.* In general, this problem refers to learning the entire probabilistic model (e.g., also the structure of a Bayesian network). We will focus only on **parameter learning**, i.e., learning the values of the parameters of a given probability model whose structure is already known or fixed. In particular, we will address learning of the **conditional probability tables** in Bayesian networks (particularly focusing on the naive Bayesian model) and also learning the parameters of **continuous probability distributions** from the data. More detailed coverage, including learning Bayesian network structures and density estimation with non-parameteric models can be found compactly described in [3]. We assume that we have **complete data**, i.e., that each data point contains values for every variable in the probability model being learned. Learning with incomplete data (with hidden variables) is postponed to Section 3.

2.1 Maximum-likelihood parameter learning: Discrete models

We will explain the main concepts of the maximum-likelihood parameter learning for discrete models starting from two simple examples that build on the *Surprise Candy* use case from Section 1.1.1.

[1]The term applied originally to probability density functions for continuous variables, but it is used now for discrete distributions too.

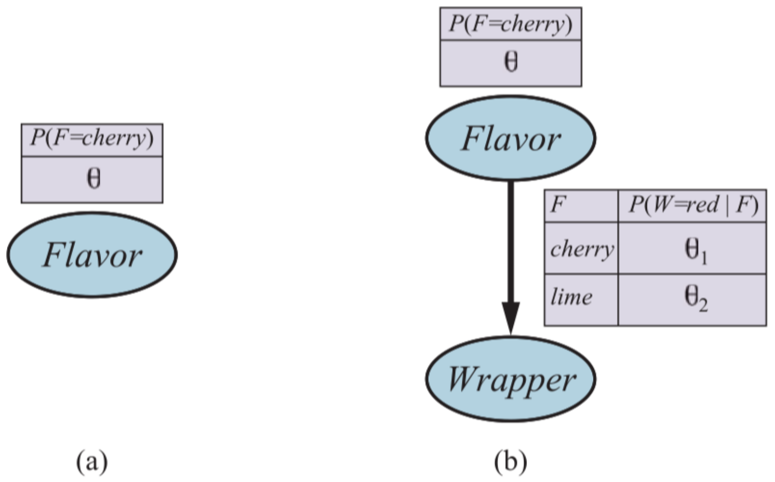

Figure 2: Bayesian network model for (a) **Example 1**: the case of candies with an unknown proportion of cherry and lime. (b) **Example 2**: the case where the wrapper color depends (probabilistically) on the candy flavor. Figure from [3].

2.1.1 Example 1

Consider first a small generalization of the *Surprise Candy* use case as follows. We don't have any more the earlier five upfront defined hypotheses but instead the fraction of cherry is a **parameter** $\theta \in [0,1]$, and the hypotheses are now h_θ. We assume that all proportions of the two candy flavors are equally likely *a priori* and we want to model this situation with a Bayesian network.

There is only one relevant random variable for this model: $Flavor$. The corresponding "Bayesian network" (an extreme case with one r.v.) is shown in Fig. 2 (a) along with the involved parameters, which is here only one parameter characterizing the prior probability $P(Flavor = cherry) = \theta$.

Let us now learn the parameter θ from data. Suppose we unwrap N candies of which c are cherry and $l = N - c$ are lime. The data likelihood given h_θ is

$$P(\mathbf{d}|h_\theta) = \prod_{i=1}^{N} P(d^{(i)}|h_\theta) = \theta^c (1-\theta)^l \tag{6}$$

The corresponding log-likelihood is

$$\ell(\theta) = \log P(\mathbf{d}|h_\theta) = \sum_{i=1}^{N} \log P(d^{(i)}|h_\theta) = c \log \theta + l \log(1-\theta) \tag{7}$$

By setting $d\ell(\theta)/d\theta = 0$ we obtain

$$\theta = \frac{c}{c+l} = \frac{c}{N} \tag{8}$$

i.e., $h_{ML} = h_{c/N}$. Thus, as we would expect, the maximum-likelihood learning asserts that the actual proportion of cherry is the same as the observed proportion.

2.1.2 Example 2

Now we build further on the previous example. Suppose there are two different wrapper colors: *red* and *green.* The wrapper color is selected for each candy depending on its flavor according to some unknown probability distribution. We want to model this new situation with a Bayesian network and to learn its parameters from the data.

First we observe that there are now two relevant r.v.s: $Flavor$ and $Wrapper$, where $Wrapper$ depends on $Flavor$, This we can represent this model with the Bayesian network shown in Fig. 2(b). Since all the r.v.s are Boolean, we have three parameters: θ, θ_1 and θ_2, where θ characterizes as before the prior probability of $Flavor$, while θ_1 and θ_2 characterize the conditional probability table (CPT) $\mathbf{P}(Wrapper|Flavor)$.

For compactness, we denote the two r.v.s with their first letters, so we the CPT is given by $P(W = red|F = cherry) = \theta_1$ and $P(W = red|F = lime) = \theta_2$. (Note that the probability of *green* given *cherry* is simply $1 - \theta_1$ and the probability of *green* given *lime* is $1 - \theta_2$).

We express the joint probability of this Bayesian network conditioned on the parameter values θ, θ_1 and θ_2. For example:

$$\begin{aligned} & P(F = cherry, W = green|h_{\theta,\theta_1,\theta_2}) \\ = \; & P(F = cherry|h_{\theta,\theta_1,\theta_2})P(W = green|F = cherry, h_{\theta,\theta_1,\theta_2}) = \theta(1 - \theta_1) \end{aligned}$$

To learn the parameter values, we unwrap N candy's; c of these are cherry and l are lime. r_c of the cherry candy's have red wrappers and g_c green. Similarly, r_l of the lime candy's have red wrappers and g_l green wrappers.

The likelihood of the data is

$$P(\mathbf{d}|h_{\theta,\theta_1,\theta_2}) = \theta^c(1 - \theta)^l \cdot \theta_1^{r_c}(1 - \theta_1)^{g_c} \cdot \theta_2^{r_l}(1 - \theta_2)^{g_l}$$

Setting the partial derivatives of the log-likelihood $\ell(\theta, \theta_1, \theta_2) = \log P(\mathbf{d}|h_{\theta,\theta_1,\theta_2})$ to zero yields

$$\theta = \frac{c}{c + l}, \quad \theta_1 = \frac{r_c}{r_c + g_c}, \quad \theta_2 = \frac{r_l}{r_l + g_l}$$

This example shows us that with complete data, the maximum-likelihood parameter learning problem for a Bayesian network decomposes into separate learning problems, one for each parameter!

2.1.3 Naive Bayes model

We could extend the previous example by adding another attribute, say *Shape* of the candy (e.g., being *round* or *square*), again depending on the candy's flavor. If this new attribute is conditionally independent of the wrapper color, given the flavor, we would represent the new situation with the second Bayesian network in Fig. 3. We can continue adding more and more attributes and as long as they are conditionally independent given $Flavor$, all the resulting models are instances of the **naive Bayes** model.

In general, in a naive Bayesian model, the **class** variable C is the root (to be predicted), and X_i are the **attributes** (**features**). We call it also **naive Bayes classifier**. Recall that we already introduced before the naive Bayesian model, when we dealt with basics of probabilistic reasoning, and at that time we referred to the root as the **cause** and the leaves as **effects** (or **symptoms**).

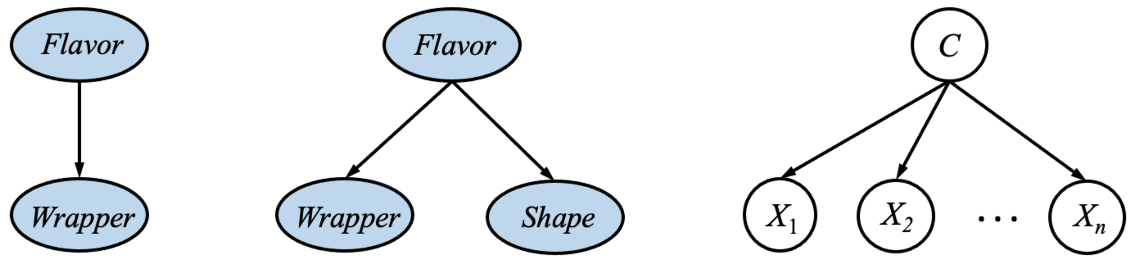

Figure 3: From left to right: two instances of a naive Bayes model and its general structure with a random variable C denoting some class, and n attributes.

We call this model **naive** because of its assumption that the *attributes are conditionally independent given the class* (or put in the words of the other terminology, it's naive because of its assumption that the effects are conditionally independent given the cause).

The joint probability of a naive Bayesian model is given by

$$\mathbf{P}(C|x_1,\ldots,x_n) = \alpha\mathbf{P}(C)\prod_j \mathbf{P}(x_j|C) \tag{9}$$

In the case where all r.v.s are Boolean we need one parameter for the class and two per attribute:

$$\theta = P(C=1), \quad \theta_{j1} = P(X_j = 1|C=1), \quad \theta_{j2} = P(X_j = 1|C=0) \tag{10}$$

The maximum-likelihood estimation of these parameters is exactly the same procedure as was explained in **Example 2**. Let $(\mathbf{x}^{(i)}, c^{(i)})$ be teh ith data point. We have that

$$\theta_{jk} = \frac{\sum_i \mathbb{1}[x_j^{(i)} = 1 \ \wedge \ c^{(i)} = k]}{\sum_i \mathbb{1}[c^{(i)} = k]} \tag{11}$$

where $\mathbb{1}[a] = 1$ if $a = true$ and 0 otherwise. In **Example 2**, we had only one attribute ($j = 1$), so the parameter θ_1 from that example is in this general notation now θ_{11} and Eq (11) gives us:

$$\theta_{11} = \frac{\#[W = red \ \wedge \ F = cherry]}{\#[F = cherry]} = \frac{r_c}{r_c + g_c}$$

in this particular example, as we obtained before.

Example: text classification

Let's consider now as an example a use case which is closer to real applications than the candy flavor prediction: **text classification**[2]:

The task is to classify each sentence into a category that corresponds to one of the major sections of the newspaper: news, sports, business, weather *or* entertainment.

The class variable $Category$ takes values $c \in \{news, sports, business, weather, entertainment\}$ and there are n Boolean attributes $HasWord_i$, $i = 1, \ldots, n$, where a predefined table of n words

[2]Example from [3], chapter 12.

is given. The model is characterized by the prior probabilities $\mathbf{P}(Category)$ and the conditional probabilities $\mathbf{P}(HasWord_i|Category)$.

The prior probability of each category is estimated as the fraction of all previously seen documents that belong to this category (for example, if 15% of articles are about sports, we set $P(Category = sports) = 0.15$. Similarly, $P(HasWord_i|Category = sports)$ is estimated as the fraction of the documents in the sports category that contain word i (e.g., if 18% of the seen articles about sports contain word 5, "fans" we set $P(HasWord_5 = true|Category = sports) = 0.18$). Actually, we are estimating the parameters of the CPT's using Eq (11).

To categorize a new document, we check which key words appear in the document and then apply Eq (9) to obtain the posterior probability distribution over categories. If we have to predict just one category, we take the one with the highest posterior probability.

Properties

Naive Bayes is a commonly used model in machine learning due to its nice properties:

- scales well to very large problems (e.g., with n Boolean attributes, only $=2n+1$ parameters);
- deals well with noisy or missing data;
- gives both probabilistic and deterministic prediction (by choosing the most likely class);
- it is intuitive, simple to implement and performs well in a wide range of applications;
- its boosted version is one the most effective general-purpose learning algorithms.

Next to all these advantages, there are also some downsides. The main drawback is that the conditional independence assumption is seldom accurate, although in practice it is often a good approximation. When this approximation is not well justified, the model can lead to wrong predictions with overconfident probabilities (close to 0 or 1), especially when the number of attributes is large.

2.1.4 Generative and discriminative models

Two kinds of machine learning models are used for classifiers

- **Generative models** – model the probability distribution of each class. From these distributions we can compute the **joint probability** and (by sampling from this joint distribution) we can **generate new examples** that are representative of each class. A representative is the **naive Bayes** model.
- **Discriminative models** – learn the decision boundary between classes. A discriminative model can learn to classify a new input to the correct class, but cannot generate new examples from that class. Representatives are **logistic regression**, **decision trees**, and **support vector machines**.

To understand well the distinction between the two types of models and why the first one can generate the new examples and the other not, consider the task of text categorization described in Section 2.1.3. The naive Bayes classifier creates a separate model for each possible category

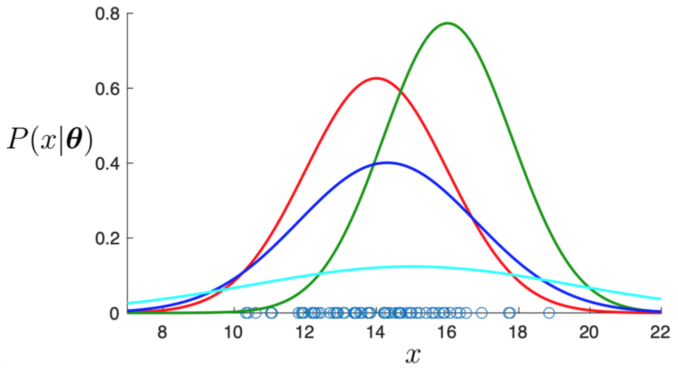

Figure 4: An illustration of the maximum-likelihood parameter estimation. Circles on the horizontal axis are the measurements and four Gaussian distributions, with different parameters $\boldsymbol{\theta} = (\mu, \sigma^2)$, are shown as candidates to fit the true distribution. (The data were actually generated from the distribution with $\mu = 14$ and $\sigma = 2$, which corresponds to the curve shown in red, so the maximum-likelihood estimation should identify this distribution as the best fit.)

of text, given by the prior probability, e.g. $P(Category = weather)$ and the conditional probabilities $\mathbf{P}(Inputs|Category = weather)$. From these, we can compute the joint probability, e.g., $\mathbf{P}(Inputs, Category = weather)$ and we can generate a random selection of words that is representative of texts in the weather category [3]. This makes the model *generative*. In the same text categorization task, a discriminative model would learn directly the posterior probability of the class given the inputs, that is, $\mathbf{P}(Category|Inputs)$, which serves directly the classification task but from which we cannot generate new example inputs.

Discriminative models tend to perform better in the classification tasks on very large data sets but on very small data sets generative models often do better.

2.2 Maximum-likelihood learning: Continuous models

Now we address the task of learning the parameters $\boldsymbol{\theta}$ of some continuous probability distribution $P(\mathbf{x}|\boldsymbol{\theta})$ from the observed data $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$. Consider first the simplest case, illustrated in Fig. 4, where data are one-dimensional and the probability distribution is Gaussian:

$$P(x|\boldsymbol{\theta}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

To learn the parameters $\boldsymbol{\theta} = \{\mu, \sigma^2\}$, we express the log-likelihood:

$$\ell(\mu, \sigma) = -\frac{N}{2}\log(2\pi) - \frac{N}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i^{(i)} - \mu)^2$$

From $\partial\ell/\partial\mu = 0$ and $\partial\ell/\partial\sigma = 0$ we obtain:

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x^{(i)} \quad \text{and} \quad \hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N}(x^{(i)} - \hat{\mu})^2$$

which are exactly the sample mean and the sample variance.

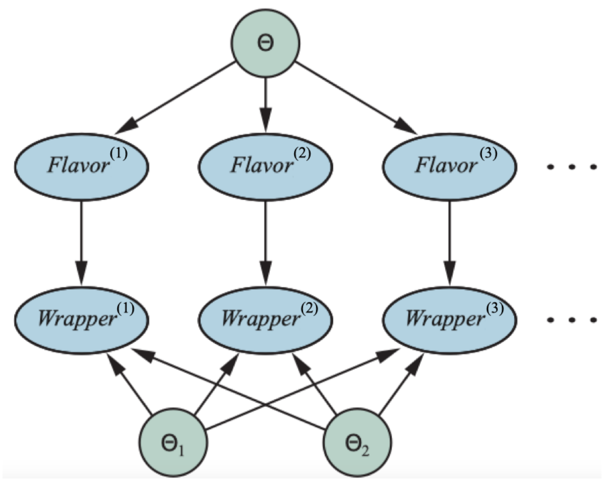

Figure 5: A Bayesian network that corresponds to a Bayesian learning process. Posterior distributions for the parameter variables Θ, Θ_1 and Θ_2 can be inferred from their prior distributions and the evidence in $Flavor^{(i)}$ and $Wrapper^{(i)}$. Figure from [3] with adapted notation.

We can generalize this procedure to the case where our observations are vectors $\mathbf{x}^{(i)}$ (i.e., each measurement is not a single number but consists of d components $\mathbf{x}^{(i)} = [x_1^{(i)}, \dots x_d^{(i)}]$) modelled by a *multivariate* normal distribution $\mathbf{x} \sim \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, with mean $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$. Applying the same procedure as above, we obtain the maximum-likelihood estimates of the parameters as

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}^{(i)}, \quad \text{and} \quad \hat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})^\top \tag{12}$$

These simple examples with the normal distribution are easy to interpret, but we can apply the same procedure to estimate the parameters of other, arbitrary statistical models.

2.3 Bayesian parameter learning

Maximum-likelihood learning is not reliable with small data sets. The Bayesian approach to parameter learning starts with a prior distribution for the hypotheses and **updates this distribution as data arrive**.

In the candy case, the parameter θ was representing the probability that a randomly selected piece of candy is of cherry flavor. In the Bayesian view, θ is a realization of a random variable Θ that defines the hypothesis space. The hypothesis prior is the prior distribution over $P(\Theta)$. Thus, $P(\Theta = \theta)$ is the prior probability that the bag has a fraction θ of cherry candies.

In **Example 1** and **Example 2** in Section 2.1, we simply used a uniform prior for θ, i.e., $P(\theta) = Uniform(\theta; 0, 1)$. This is because we reasoned that we don't know anything about the possible values of θ, so we let them all be equally likely. Now we rather assume some more general, parametrized distribution that is flexible enough and whose parameters can be updated based on the new data. This will allow us to continuously adjust the prior distribution with new data. Similarly, we will do for all the involved parameters that define the hypotheses. This process is illustrated in Fig. 5.

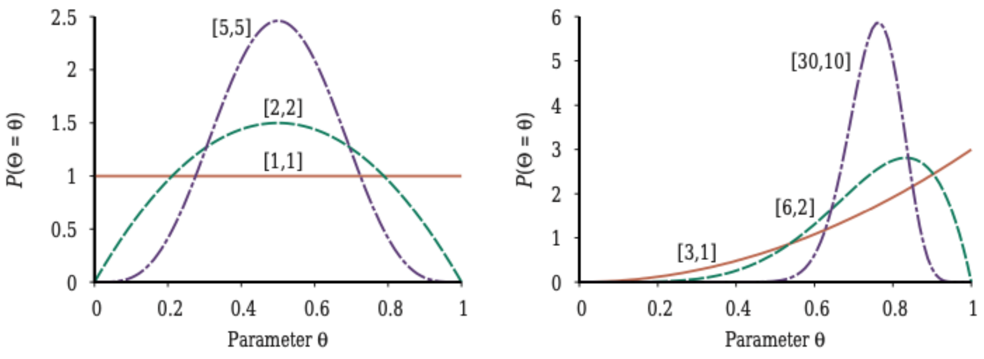

Figure 6: Examples of the $Beta(a, b)$ distribution for different values of (a, b). Figure from [3].

A very convenient distribution for this purpose is the **Beta** distribution:

$$Beta(\theta; a, b) = \alpha\ \theta^{(a-1)}(1-\theta)^{(b-1)}$$

for θ in the range $[0, 1]$. Its mean value is $a/(a + b)$, so larger values of a suggest a belief that Θ is closer to 1 than to 0. Larger values of $a + b$ make the distribution more peaked, suggesting greater certainty about the value of Θ. Fig. 6 illustrates the shape of this distribution with different parameters.

The beta family has a very useful property for the learning process: it is **closed under update**, meaning that the distribution updated with the new data remains the *Beta* distribution, just with the updated parameters.

To see how this updating of the hypothesis prior works, suppose we start from the distribution $P(\theta) = Beta(\theta; a, b)$ with some chosen values of a and b, and we unwrap the first candy. If this first candy is cherry, the update will be as follows:

$$\begin{aligned} \text{E.g.,}\quad P(\theta \mid D^{(1)} = cherry) &= \alpha P(D^{(1)} = cherry \mid \theta)P(\theta) \\ &= \alpha'\theta \cdot Beta(\theta; a, b) = \alpha'\theta \cdot \theta^{(a-1)}(1-\theta)^{(b-1)} \\ &= \alpha'\theta^{a}(1-\theta)^{(b-1)} = \alpha' Beta(\theta; a+1, b) \end{aligned}$$

Thus, seeing a cherry candy leads to incrementing the a parameter of the distribution. Effectively, this increases the belief that θ, which represents the portion of cherry, is closer to one than to zero. Conversely, observing a lime candy will lead to incrementing the b parameter and this way shifting our belief towards smaller values of θ. We obtain this way the desired learning of the parameter distribution from the observed data.

3 Learning with hidden variables

In practice, data entries are often missing resulting in incomplete information to specify a likelihood. This complicates the learning of the model parameters. We make a differentiation between the data that are observable but just not available in some data points (e.g., due to a particular sampling pattern or sensor errors etc.) and data that are never directly observable – these correspond to the so called **hidden** or **latent variables**.

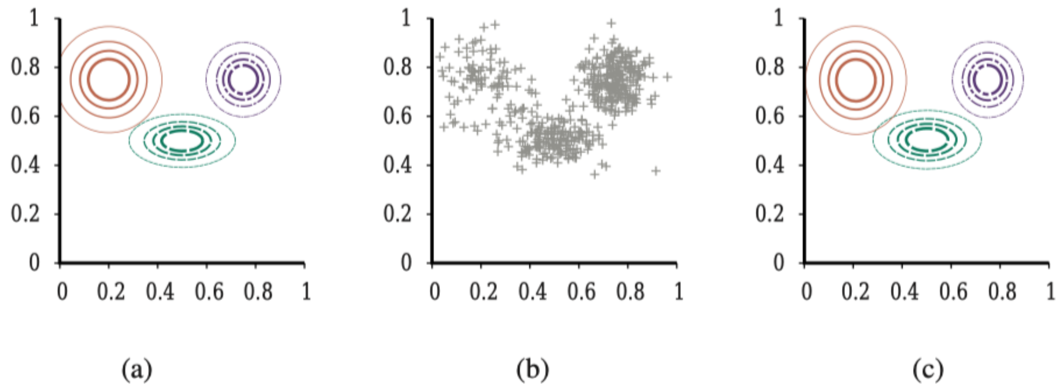

Figure 7: A Gaussian mixture model with three components; the weights (left-to right) are 0.2, 0.3, and 0.5. (b) 500 data points sampled from the model in (a). (c) The model reconstructed by EM from the data in (b). Figure from [3].

Latent variables are random variables that are essential for the model description but never directly observed. The following definition can be found on Wikipedia:
In statistics, latent variables (from Latin: present participle of lateo, "lie hidden") are variables that can only be inferred indirectly through a mathematical model from other observable variables that can be directly observed or measured [1].

For example, the underlying physics of a model may contain latent processes which are essential to describe the model, but cannot be directly measured. Remember also that we sometimes introduce hidden variables to **sparsify** the structure of a Bayesian network. This way we can dramatically reduce the number of parameters required to specify a Bayesian network. We will focus on a particular type of latent variable models: mixture models that we address next.

3.1 Clustering by learning Gaussian mixture models

In general, statistical models that represent distributions of observable variables using latent (or hidden) variables are called **latent variable models**. An important class of these models are the so-called **mixture models**, where the probability distribution is a mixture of k **components**:

$$P(\mathbf{x}) = \sum_{i=1}^{k} \underbrace{P(C=i)}_{\pi_i} P(\mathbf{x}|C=i) \tag{13}$$

Here, C is a random variable that denotes the component, with values $1, \ldots, k$ and $\pi_i = P(C=i)$ is the weight given to the ith component in the mixture. A mixture model naturally represents (soft) clustering of data. We thus also say that mixture models make use of latent variables to model different parameters for different groups (or **clusters**) of data points

When all the components in the mixture model are Gaussian, we have a mixture of Gaussians, also called the **Gaussian mixture model** (**GMM**):

$$P(\mathbf{x}) = \sum_{i=1}^{k} \pi_i \, \mathcal{N}(\mathbf{x}; \mu_i, \Sigma_i) \tag{14}$$

Fig. 7 illustrates data generated from a GMM model, and also fitting the model to data with an algorithm that will be explained in Section 3.2.

Mixture models are very useful in practice. Consider this example from [2]:

We need to build a probabilistic language model of news articles, which assigns probabilities to sequences of words $\mathbf{x}_1, \ldots, \mathbf{x}_n$*. Each article typically focuses on a specific topic e.g., finance, sports, politics. Using this prior knowledge, we may build a more accurate model* $P(\mathbf{x}|c)P(c)$*, in which we have introduced an additional, unobserved random variable* C*. This model can be more accurate, because we can now learn a separate* $P(\mathbf{x}|c)$ *for each topic, rather than trying to model everything with one (very complex)* $P(\mathbf{x})$*.*

Note certain similarities with the description of the naive Bayes use case for text classification in Section 2.1.3. However, since c is now unobserved, we cannot directly use the learning methods that we have studied in Section 2 but we will introduce a new learning algorithm that we will explain particularly for learning the mixtures of Gaussians.

3.2 EM algorithm for the mixtures of Gaussians

Fitting a GMM model to data like in Fig. 7 is in essence a **clustering problem**. Each cluster will be modelled by a distribution (in this particular case, a Gaussian of given mean and covariance) and we will obtain the probabilities describing how likely it is that the data point belongs to a given cluster. This is a form of **soft clustering** but we can obtain easily from it also **hard clustering** (e.g., simply by assigning the data point to the component (cluster) for which the probability is the largest.

The parameters of this model are effectively learned by the so-called **expectation-maximization** (**EM**) algorithm. The basic idea of EM is to start from some initial parameters of the model and to iterate two steps: (1) infer the probability that each data point belongs to each component. (2) refit the components to the data, where each component is fitted to the entire data set with each point weighted by the probability that it belongs to that component.

Concretely, the EM algorithm for GMMs iterates the following two steps [3]:

- **E-step**: *Compute probabilities* $p_{ij} = P(C = i|\mathbf{x}^{(j)})$
  - By Bayes' rule:
    $p_{ij} = \alpha P(\mathbf{x}^{(j)}|C = i)P(C = i)$
  - Define n_i as the effective number of data points assigned to component i:
    $n_i = \sum_j p_{ij}$
- **M-step**: *Compute the new mean, covariance and weights*
  - Means:
    $\mu_i \leftarrow \sum_j p_{ij}\mathbf{x}^{(j)}/n_i$
  - Covariance matrices:
    $\Sigma_i \leftarrow \sum_j p_{ij}(\mathbf{x}^{(j)} - \mu_i)(\mathbf{x}^{(j)} - \mu_i)^\top/n_i$
  - Weights:
    $\pi_i \leftarrow n_i/N$

GMM-based clustering can be seen as a probabilistic version of the deterministic clustering method known as **K-means**.

K-means algorithm initializes (randomly) the **centroids** $\mu_1, \dots, \mu_k$ and iterates two steps:

1. Assign each point to the nearest (in terms of L_2 distance) centroid:

$$\forall j, \quad c^{(j)} = \arg \max_{i=1,\dots,k} \|\mathbf{x}^{(j)} - \mu_i\|^2$$

2. Recompute the centroids based on the assigned points: $i = 1, \dots, k$:

$$\mu_i \leftarrow \frac{1}{|\{j : c^{(j)} = i\}|} \sum_{j:c^{(j)}=i} \mathbf{x}^{(j)}$$

K-means is sometimes employed to initialize GMM-based clustering.

References

[1] Y. Dodge. *The Oxford Dictionary of Statistical Terms.* 2003.

[2] S. Ermon. *Probabilistic Graphical Models. (CS228).* Stanford University, 2024.

[3] S Russel and Norvig. P. *Artificial Intelligence: A Modern Approach, 4th Edition.* Pearson, 2021.