Probabilistic models for classification

Probabilistic generative models

Let's consider the usual problem of classification. We want to find a function $f: X \to C$, given a training dataset D, with the goal of being able to classify new instances $x' \notin D$.

💡 Generative models have a **two-step approach** to this problem. The first one consists in computing the **class-conditional density** $P(x|C_i)$. This is a description of how the dataset has been generated in terms of a probabilistic model, meaning that given this first model we could be able to generate new data sampling from it. The second step for this approach will be using the class-conditional density in order to compute the **posterior probability** $P(C_i|x)$ **using Bayes' theorem**.

During the following explainations, the term D representing the dataset will be omitted from formulas in order to use a lighter notation.

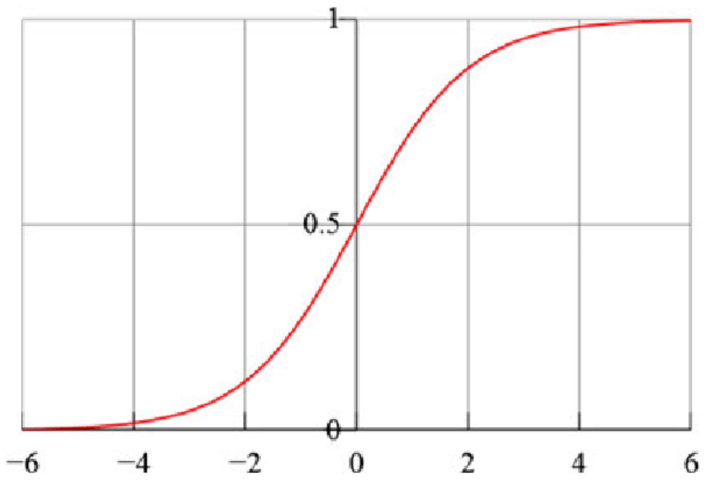
The two classes case

We want to find the posterior probability assuming that we have two classes C_1, C_2:

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x)} = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)} = \frac{1}{1+e^{-a}} = \quad = \sigma(a)$$

This first formula shows us that we can model the computation of posterior probability as a sigmoid function, having as a term $a = ln\frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)}$, the logaritm of the ratio of the two classes.

A sigmoid function has the interesting probability of squeezing every value $\in (-\infty, +\infty)$ in a range between 0 and 1.



We are now interested in computing the conditional probability of a sample given a class. We will start from the important assumption that samples are generated by an underlying Gaussian distribution. We obtain the following equality:

$$P(x|C_i) = \mathcal{N}(x; \mu_i, \Sigma)$$

Where the Gaussian distribution is parametrized by two terms: the mean for each class μ_i and the covariance matrix Σ, which is the same for every class. That said, we can update our formula for the term a:

$$a = ln\frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)} = ln\frac{\mathcal{N}(x; \mu_1, \Sigma)P(C_1)}{\mathcal{N}(x; \mu_2, \Sigma)P(C_2)}$$

Now it comes an important trick. We define the terms:

$$w = \Sigma^{-1}(\mu_1 - \mu_2)$$

$$w_0 = -\frac{1}{2}\mu_1^T\Sigma^{-1} + \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + ln\frac{P(C_1)}{P(C_2)}$$

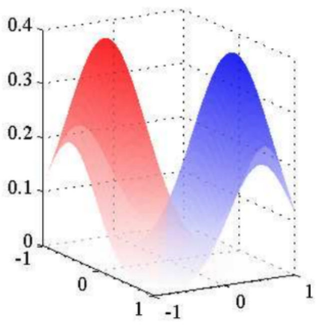
That after some computation will let us express the term a as a linear combination of terms in x:

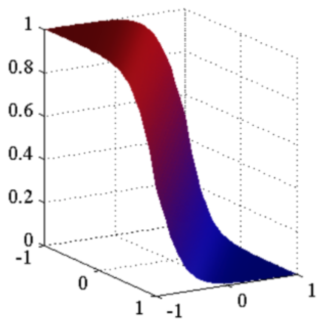
$$a = ln\frac{\mathcal{N}(x; \mu_1, \Sigma)P(C_1)}{\mathcal{N}(x; \mu_2, \Sigma)P(C_2)} = ... = w^Tx + w_0$$

Since a is the only variable in the formulation of the posterior probability as a sigmoid, we reduced the problem of computing the posterior probability as a linear equation:

$$P(C_1|x) = \sigma(w^Tx + w_0)$$

Let's now assume that we can compute w^T and w_0. This means that we know the distribution of the classes over the samples, an example of that is the left illustration in the image below. Starting from this information, we will able the compute the posterior probability for every class. The logic here is that by placing any new instance on the distribution we will be able to know how likely it is that it belongs to a certain class.


$P(\mathbf{x}|C_1), P(\mathbf{x}|C_2)$


$P(C_1|\mathbf{x})$

In the two class cases the decision making policy is simple:

$$P(C_1|x) \begin{cases} > 0.5 & \text{output } C_1 \\ < 0.5 & \text{output } C_2 \end{cases}$$

The k-class case

Let's suppose the general case, where we have three classes. The prior probability for each class is:

$$P(C_1) = \pi_1$$
$$P(C_2) = \pi_2$$
$$\dots$$
$$P(\pi_k) = 1 - \sum_{i=1}^{k-1} \pi_i$$

Exam question:
How many independent parameters does a model with k classes have?
A model with k classes has k-1 independent parameters, since we just need k-1 parameters in order to computed the k-th one.

Computing the solution for 2 classes

We want to find a way to compute the parameters of our problem. Let's take a step back to the two classes case.

Given the previous assumptions we have $P(C_1) = \pi, P(C_2) = 1 - \pi$. The class-conditional distributions are $P(x|C_1) = \mathcal{N}(x|\mu_1, \Sigma), P(x|C_2) = \mathcal{N}(x|\mu_2, \Sigma)$.

We want to find a solution to computing the parameters μ_1, μ_2, Σ.

The first step is to properly encode the training dataset. Considering the two classes case, our dataset will be a set of pairs in the form $D = \{(x_n, t_n)_{n=1}^N\}$, where t_n is a binary variable that is 1 if $x_n \in C_1$, 0 if $x_n \in C_2$.

The result is a vector $t = [t_1 \ t_2 \ ... \ t_n]^T$ that encodes every class for every sample.

Now we define a likelihood function:

$$P(t|\pi, \mu_1, \mu_2, \Sigma, D) = \prod_{n=1}^{N} [\pi \mathcal{N}(x_n; \mu_1, \Sigma)]^{t_n} [(1-\pi)\mathcal{N}(x_n; \mu_2, \Sigma)]^{1-t_n}$$

Since we only know D, our goal is to determine the parametes $\pi, \mu_1, \mu_2, \Sigma$ that maximize the likelihood:

$$\pi, \mu_1, \mu_2, \Sigma = argmax_{\pi, \mu_1, \mu_2, \Sigma} P(t|\pi, \mu_1, \mu_2, \Sigma)$$

In order for the problem to be easier to solve, we can formulate it as maximizing the log-likelihood, which means literally to find the argmax for the logaritghm of the previous function (the result of this operation will be the same). In order to compute the argmax we compute the derivative of the function, put it equals to 0 and find the solution for each term.

The estimations for each term will be:

$$\pi = P(C_1) \approx \frac{|\{x, C_1\}|}{|D|}$$

$$\mu_1 = \frac{1}{N_1} \sum_{x_i \in C_1} x_i$$

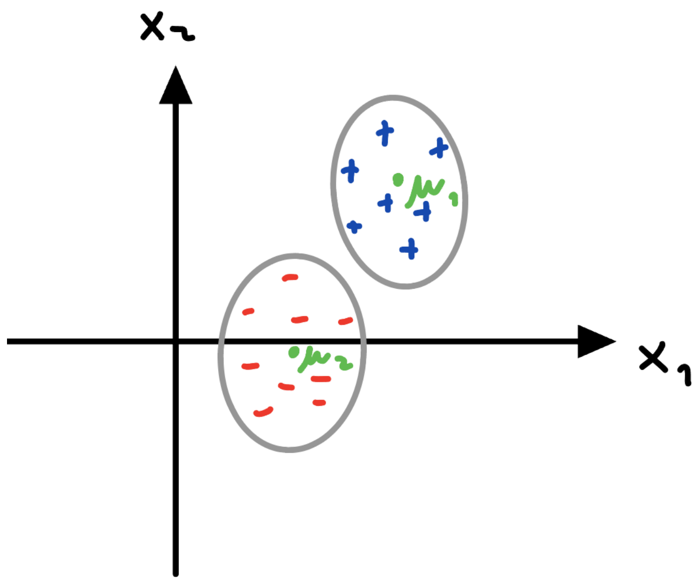
N_1 is the number of samples in class 1 and $\sum_{x_i \in C_1} x_i$ is the sum of samples in class 1.

$$\mu_2 = \frac{1}{N_2} \sum_{x_i \in C_2} x_i$$

$$\Sigma = \frac{N_1}{N} S_1 + \frac{N_2}{N} S_2$$

Σ is the weighted average of the distances of each point from their mean.

The following graph shows the distribution of two samples in 2D:



Notice that the model is robust to noise as long as noisy samples are a minority inside the distribution where they not belong.

❗ The model will work as long as samples follow a Gaussian distribution. It will fail in every other case.

A general solution for k classes

In the general case the steps needed to find the solution will be the same, we will just have to adapt the encoding of the training dataset. It will still be represented as a set of pairs $D = \{(x_n, t_n)\}$. But we will use a 1-out-of-k encoding, meaning that for each sample the classes will be encoded as a 1xK vector. There will be a 1 in the position corresponding to the class it belongs, and 0 in every other position:

$$t = [\ ...\ [0\ 0\ 0\ ..\ 0\ 1\ 0..\ 0]\ ...\ ]^T$$

The maximum likelihood will be computed by taking the derivative with respect to each class.

Probabilistic discriminative models

The discriminative approach to probabilistic models is about computing directly the posterior probability of the class given a sample. This means that we won't be able to generate new samples given their distibution, but only to classify them. The reason for this is that we will extend the assumption of studying only Gaussian distributions, but this will make us lose the ability to compute the underlying parameters, making us unable to generate new samples.

This method of classification is called **Logistic Regression.** Notice that even if the name "Regression" is often associated to models in the continuous space, here we will refer to classification problems.

Our first consideration is that posterior probability can be computed not only for Gaussian distribution, but for all exponential ones. In the general case (k>2) the posterior probability will be expressed as a linear combination of the input features:

$$P(C_i|x) = \frac{e^{a_k}}{\sum_j e^{a_j}}$$

With the term $a_k = w^T x + w_0$.

In order to have a more compact notation, we can assign the following terms:

$$w^T x + w_0 = \begin{bmatrix} w_0 & w \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix}$$

$$\tilde{w} = \begin{bmatrix} w_0 \\ w \end{bmatrix} \quad \tilde{x} = \begin{bmatrix} 1 \\ x \end{bmatrix}$$

So that $a_k = w^T x + w_0 = \tilde{w}^T \tilde{x}$

Since we are not interested in particular parameters, we generically define parametric models as $\mathcal{M}_\Theta := P(t|\Theta, D)$.

The maximum likelihood function will be the set of parameters that will maximize the log-likelihood:

$$\Theta^* = argmax_\Theta \; ln(P(t|\Theta, D))$$

When $\mathcal{M}_\Theta$ belongs to the exponential family, we know that its parameters can be expressed as a linear combination of values, meaning that we can formulate the maximum likelihood as:

$$\tilde{w}^* = argmax_{\tilde{w}} \; ln(P(t|\tilde{w}, x))$$

Two-class logistic regression

In the two classes case we will have $D = \{(x_n, t_n)\}$ with $t_n \in \{0, 1\}$. The likelihood function will be:

$$P(t|\tilde{w}) = \prod_{n=1}^{N} y_n^{t_n} (1 - y_n)^{1-t_n}$$

Notice that t_n is the actual value of the dataset, while $y_n = P(C_1|\tilde{x}_n) = \sigma(\tilde{w}^T \tilde{x}_n)$ is the posterior prediction of the current model.

The error function for the model is the negative log-likelihood, defined as:

$$E(\tilde{w}) = -ln(P(t|\tilde{w})) = -\sum_{n=1}^{N} [t_n ln(y_n) + (1 - t_n) ln(1 - y_n)]$$

This error function is tipically defined as the *cross-entropy* function.

Our goal will be minimizing the error, namely finding the argmin for the error function:

$$\tilde{w}^* = argmin_{\tilde{w}} E(\tilde{w})$$

❗ **Exam question:**
What is the relationship between cross-entropy and likelihood?
Cross entropy is an error defined as the negative of the log-likelihood.

Solving this minimization problems becomes just a matter of optimization.

A possible approach to the resolution is finding an iterative weightes least squares algorithm. It consists in starting from a random point in space and performing gradient descent until a minimum is found. A main problem with this technique is that we are not guaranteed to find the global minimum, but just a local one.

Multi-class logistic regression

The same concept as before can be extended when we have to predict k classes. We want to find $f : \mathbb{R}^d \rightarrow (C_1, ..., C_k)$.

We will use a 1-out-of-k encoding to represent the output of the dataset. This will result in a matrix made by row vectors.

The procedure will be the same: encode training inputs and outputs, define a likelihood function and solve the resulting optimization problem.

A really useful notion about this method is that it can be applied for any transformed space of the input (feature space).