1 Naive Bayesian Model

Once you understand the idea of MLE (Maximum Likelihood Estimate) calculation, you can turn it into a supervised learning model very easily. It's a rather poor model, but it's the basic building block for more complex, more expressive models. It is therefore important to have a good understanding of this basic brick. In langage models, it's not so bad after all..

In addition, it makes a good exercise of manipulation of indices and sums or products, that are still quite simple.

Here, to be concrete, we will choose a Bernoulli model. We remind you that Bernoulli's probability distribution can be written: $\mathbb{P}(X=1)=p, \mathbb{P}(X=0)=1-p$, which can be summarized in the denser form: $\mathbb{P}(X=x)=p^x(1-p)^{1-x}$. (we use the fact that x is necessarily worth 0 or 1, and that $A^0=1, \forall A$).

1.1 Problem and data structure

One can imagine that we are interested in supervised classification of black and white images (or grey levels).

The structure of the data is specified: N, the size of the learning data. D, the size of the data
K, the number of classes present in the data to be classified
$X_{(N,D)}$: the training data (typically a set of images). $\vec{x}_n$ is the image number n, it corresponds to D real numbers (gray intensities of pixels) $y_{(N)}$: the classes of the data ($y[n]$ is the class of the data number n (typically, with values in $[1,...,K]$)
Θ: the parameters to be optimized (the detail depends on the chosen model, but there are at least K parameters, probably KD or more).

1.2 Learning the model parameters

We wish to maximize the likelihood of data, for a chosen model, i.e. for a fixed form of $\mathbb{P}(X=(\vec{x})_n|y=k,\Theta)$. For example, Gaussian or Bernoulli model, or another explicit form of the relationship between $\mathbb{P}(X=(\vec{x})_n|y=k,\Theta)$ and the variables $x_{n,d}, y, \Theta_{k,d}$. Intuitively, the term likelihood can be replaced by credibility, degree of realism, or realistic-ness: we ask ourselves: "what are the parameters Θ that make the observation of such or such data (image) credible/realistic?". It is therefore quite reasonable to maximize this "Likelihood of the data".

Maximizing the likelihood of data means learning the values of the Θ parameters such that $\mathcal{L}(\Theta)=\mathbb{P}(X=(\vec{x})_n|y=k,\Theta)$ is maximal. We can call Θ^* the solution to this optimization problem:

$$\Theta^* = argmax_\Theta(\mathcal{L}(\Theta)) \tag{1}$$

This expression is developed in order to obtain an explicit form whose derivative is not too hard to calculate:

$$\begin{aligned}
\Theta^* &= argmax_\Theta\left(\log(\mathcal{L}(\Theta))\right) && (2)\\
&= argmax_\Theta\left(\log\mathbb{P}(X=(\vec{x})_n|y_{(n)},\Theta)\right) && (3)\\
&= argmax_\Theta\left(\log\prod_n^N \mathbb{P}(X_n=\vec{x}_n|y_n=k,\Theta)\right) && (4)\\
&= argmax_\Theta\left(\sum_n^N \log\mathbb{P}(X_n=\vec{x}_n|y_n=k,\Theta)\right) && (5)
\end{aligned}$$

At this stage we cannot go any further in the general case, and we have to choose an explicit form for the slightly abstract expression $\mathbb{P}(X_n=\vec{x}_n|y=k,\Theta)$. This is what we call a *model* of the data.

The **naive** Bayesian models correspond to suppose that the input features are **independent** ! This is a huge simplification! In real life, in typical data, there are correlations between features (this is what we see for example with PCA). Concretely, feature independence means that for each sample $\vec{x}_n$ (data point), each value $(x_n)_d$ is independent of the others $(x_n)_{d'}$ (of the ohter pixels of the same image). So we write:

$$\mathbb{P}(X_n=\vec{x}_n|y_n=k,\Theta)=\prod_d^D \mathbb{P}(x_{n,d}=x_{n,d}|y_n=k,\Theta) \tag{6}$$

There is here a small abuse of notation: one writes $\mathbb{P}(x_{n,d}=x_{n,d})$, the term on the left is the random variable $x_{n,d}$, and the term on the right is the value taken in the realization of this random variable, which is also denoted $x_{n,d}$ (it is a slight abuse).

We note $\mathbb{1}_{y_n=k}$ the function that is 1 when $y_n=k$ and 0 otherwise. It's a quick way to enforce that $k=y_n$ everywhere in the equation.

In the case where the input feature distribution model is chosen Bernoulli, for each feature, this is written, more concretely: $\mathbb{P}(X_n = \vec{x}_n | y_n = k, \Theta) = \prod_d^D p_{k,d}^{x_{n,d}} (1 - p_{k,d})^{(1-x_{n,d})} \mathbb{1}_{y_n=k}$. Here the parameters Θ are, concretely, the $p_{k,d}$.

$$\Theta^* = argmax_\Theta \left(\sum_n^N \log \Big(\prod_d^D p_{k,d}^{x_{n,d}} (1 - p_{k,d})^{(1-x_{n,d})} \Big) \mathbb{1}_{y_n=k} \right) \tag{7}$$

$$= argmax_\Theta \left(\sum_n^N \sum_d^D \log \Big(p_{k,d}^{x_{n,d}} (1 - p_{k,d})^{(1-x_{n,d})} \Big) \mathbb{1}_{y_n=k} \right) \tag{8}$$

$$= argmax_\Theta \left(\sum_n^N \sum_d^D \left[x_{n,d} \log(p_{k,d}) + (1 - x_{n,d}) \log(1 - p_{k,d}) \right] \mathbb{1}_{y_n=k} \right) \tag{9}$$

To find the maximum of this function in Θ, we derive with respect to Θ (here, concretely, $p_{k,d}$) and we look for the value of $p_{k,d}$ such that the derivative is null:

$$0 = \frac{\partial \log \mathcal{L}(\Theta)}{\partial p_{k,d}} \tag{10}$$

$$= \frac{\partial}{\partial p_{k,d}} \Big(\sum_n^N \sum_{d'=1}^D \left[x_{n,d'} \log(p_{k,d'}) + (1 - x_{n,d'}) \log(1 - p_{k,d'}) \right] \mathbb{1}_{y_n=k} \Big) \tag{11}$$

$$= \sum_n^N \left[\frac{x_{n,d}}{p_{k,d}} - \frac{1 - x_{n,d}}{1 - p_{k,d}} \right] \mathbb{1}_{y_n=k} + \sum_{d' \neq d}^D (0) \tag{12}$$

$$0 = \sum_n^N \left[x_{n,d}(1 - p_{k,d}) - (1 - x_{n,d}) p_{k,d} \right] \mathbb{1}_{y_n=k} \tag{13}$$

$$= \sum_n^N \left[x_{n,d} - p_{k,d} \right] \mathbb{1}_{y_n=k} \tag{14}$$

Which gives, by distributing the sum:

$$\sum_n^N p_{k,d} \mathbb{1}_{y_n=k} = \sum_n^N x_{n,d} \mathbb{1}_{y_n=k} \tag{15}$$

$$p_{k,d} \sum_n^N \mathbb{1}_{y_n=k} = \sum_n^N x_{n,d} \mathbb{1}_{y_n=k} \tag{16}$$

$$p_{k,d} = \frac{\sum_n^N x_{n,d} \mathbb{1}_{y_n=k}}{\sum_n^N \mathbb{1}_{y_n=k}} \tag{17}$$

This corresponds to averaging the pixel number d of the images of class k: indeed the denominator is simply a way to count the number of images in the class k.

We can notice that in this case, we were able to do the calculations exactly until the end, and therefore the learning process is done in one shot, analytically, and so there are no iterations (epochs).

We can also notice that the final result is extremely simple: we simply take the empirical average of the pixel number d of all the images being of the class k, and this gives us the parameter p_{kd} associated to this pixel. In the case where there are correlations between pixels, modeled by chosen laws, the computation can quickly become much more complex, or even analytically unfeasible.

1.3 Decision function

In this part, we will use the Bayes formula, in a somewhat generalized form:

$$\mathbb{P}(A|B, C) \mathbb{P}(B|C) = \mathbb{P}(A, B|C) \tag{18}$$

For a new data point (validation or test data), we know $\vec{x}$ and we want to predict y. We now know the parameters Θ. We choose the class that maximizes the likelihood of the data.

Intuitively, we choose the class k to be the one that most likely corresponds to the observed image $\vec{x}$, assuming it is generated by the parameters we learned (the vector (p_k), of D dimensions).

Mathematically, this corresponds to:

$$\begin{aligned}
k^* &= argmax_k(\mathbb{P}(y=k|\vec{x},\Theta)) && (19)\\
&= argmax_k\left(\frac{\mathbb{P}(y=k\cap\vec{x}|\Theta)}{\mathbb{P}(\vec{x}|\Theta)}\right) && (20)\\
&= argmax_k\left(\frac{\mathbb{P}(\vec{x}|y=k,\Theta)\mathbb{P}(y=k|\Theta)}{\mathbb{P}(\vec{x}|\Theta)}\right) && (21)\\
&= argmax_k\mathbb{P}(\vec{x}|y=k,\Theta)\mathbb{P}(y=k|\Theta) && (22)
\end{aligned}$$

The last line is obtained by noticing that the denominator does not depend on k, so it has no importance in determining the location of the maximum.

The term $\mathbb{P}(\vec{x}|y=k,\Theta)$ has already been seen above, it depends on the model chosen (Bernoulli Naïf, Gaussian Naïf or other).

The term $\mathbb{P}(y=k|\Theta)$ corresponds to the probability of observing a class image k, knowing the learned parameters (Θ) but without knowing the image, $\vec{x}$. That is, in fact, the probability that a randomly drawn image is of class k, among the images of *the learning set*. It is therefore a quantity (K real number) that must also be *learned*:

$$\mathbb{P}(y=k|\Theta) = \frac{\sum_n^N \mathbb{1}_{y_n=k}}{\sum_n^N 1} \qquad (23)$$

We can see that this "learning" is done all at once, without iterations. It is simply the calculation of the frequency of the k class in the training set.

So, in concrete terms, we have:

$$k^* = argmax_k\mathbb{P}(\vec{x}|y=k,\Theta)\mathbb{P}(y=k|\Theta) = argmax_k\Big(\prod_d [p_{k,d}^{x_{n,d}}(1-p_{k,d})^{(1-x_{n,d})}]\frac{\sum_n^N \mathbb{1}_{y_n=k}}{N}\Big) \qquad (24)$$

Numerically, it is preferable to maximize the log of this number, to avoid rounding errors. However, one must be careful that none of the $p_{k,d}$ is zero. If this is the case (and it will most likely be the case, in MNIST or this kind of very simple dataset), this numerical problem must be corrected, by replacing:

$$k^* = argmax_k\Big(\sum_d [(x_d)\log(p_{k,d}) + (1-x_d)\log(1-p_{k,d})] + \log\mathbb{P}(y=k)\Big) \qquad (25)$$

by

$$k^* = argmax_k\Big(\sum_d [(x_d)\log(p_{k,d}+\varepsilon) + (1-x_d)\log(1-p_{k,d}+\varepsilon)] + \log\mathbb{P}(y=k)\Big) \qquad (26)$$

With $\varepsilon = 10^{-8}$ for example.

1.4 Pre-processing

Here, a Bernoulli model was used. Implicitly, this assumes that the inputs take values in the set $\{0,1\}$. If we want to apply this algo on data (black and white images), what should we do as pre-processing?

1.5 Priors

Imagine you are given an unbalanced training set, where a class is over-represented. Assume you know that in the data, in general (so, in the test set, in particular), the data is evenly distributed between all classes. What can you change in the decision function, to account for that? This is a very simple example of a *prior*.

1.6 Naive Gaussian Model

Compute the learning equations for the Naive Gaussian Model.

1.7 Not-so-naive Gaussian Model: Gaussian Mixture Models

To generate a multi-variate Gaussian distribution with correlated entries, there is a very straightforward way. We can define a covariance matrix Σ, and then draw d-dimensional samples from it like this:

$$\mathbb{P}(X = x) \sim \mathcal{N}(\mu, \Sigma) \sim \frac{1}{(2\pi)^{d/2} det(\Sigma)^{1/2}} \exp\Big(- \frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\Big) \tag{27}$$

Since most matrices are invertible, we can pick Σ in any way we like (like, random) and it's going to be ok.
To train, we will need to solve the MLE for this model.
Exercise: prove that the MLE estimators for the parameters μ^*, Σ^* (for a generic class) are of the form:

$$\mu^* = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{28}$$

$$\Sigma^* = \frac{1}{N}\sum_{n=1}^{N}(x_n - \mu^*)^T(x_n - \mu^*) = \text{ the covariance matrix of the data!} \tag{29}$$

Then, if we have K classes and D dimensions, how many parameters are there in our model ?
Are you worried ?
Are you happy ?

This stuff is very related to "Gaussian Mixture Models" (GMMs).