MLISP: Machine Learning in Signal Processing

Lecture 8

Prof. V. I. Morgenshtern

Scribe: M. Elminshawi

Illustrations: The elements of statistical learning, Hastie, Tibshirani, Friedman

Agenda:

1. Logistic regression
2. Phoneme recognition example

1 Logistic regression

Our goal in this lecture is to apply splines to the problem of speech-to-text conversion. Concretely, based on the sound recording, we want to distinguish between two phonemes "aa" and "ao" that are really difficult to distinguish. This is a two class classification problem.

The simplest algorithm we already know for a two class classification problem is the linear regression followed by thresholding. Let's denote the labels of the two classes by 0 (encoding "aa") and 1 (encoding "ao"). Then, this method amounts to learning a coefficient vector $\boldsymbol{\theta}$ and using the decision rule:

$$\begin{aligned} &\text{If} \quad \mathbf{x}^\mathsf{T}\boldsymbol{\theta} \leq 0.5 \quad \text{predict} \quad \mathrm{y} = 0 \\ &\text{else} \quad \text{predict} \quad \mathrm{y} = 1. \end{aligned}$$

Note that the class identity is predicted directly.

More accurate results can be obtained using the *logistic regression*. In contrast to predicting the class identity directly, in logistic regression we predict the probability:

$$\mathbb{P}[\mathrm{y} = 1 | \mathbf{x} = \mathbf{x}].$$

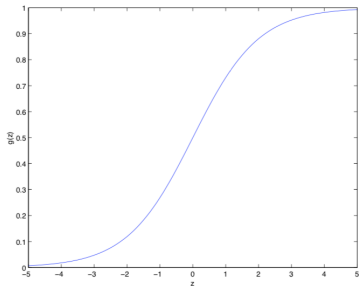
This probability is modeled by a linear function of the features followed a nonlinearity:

$$\mathbb{P}[\mathrm{y} = 1 | \mathbf{x} = \mathbf{x}] = h_{\boldsymbol{\theta}}(\mathbf{x})$$

with

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g(\mathbf{x}^\mathsf{T}\boldsymbol{\theta}) = \frac{1}{1 + e^{-\mathbf{x}^\mathsf{T}\boldsymbol{\theta}}}.$$

Above, the parameter $\boldsymbol{\theta}$ is estimated from data and $g(z) = \frac{1}{1+e^{-z}}$ is called the *logit* function or the *sigmoid* function. It looks like this:



Note that:

$$g(z) \to 1 \quad \text{as } z \to \infty$$
$$g(z) \to 0 \quad \text{as } z \to -\infty$$

it is bounded between 0 and 1 and hence it can represent probability. Therefore, $h_\theta(\mathbf{x})$ is also bounded between 0 and 1.

To make sure that probabilities sum up to one, we set:

$$\mathbb{P}[\mathtt{y} = 0 | \mathtt{x} = \mathbf{x}] = 1 - \mathbb{P}[\mathtt{y} = 1 | \mathtt{x} = \mathbf{x}] = 1 - \frac{1}{1 + e^{-\mathbf{x}^\intercal \theta}} = \frac{e^{-\mathbf{x}^\intercal \theta}}{1 + e^{-\mathbf{x}^\intercal \theta}}.$$

For future reference, let's calculate the derivative of $g(z)$:

$$\begin{aligned} g'(z) &= \frac{d}{dz} \frac{1}{1 + e^{-z}} \\ &= \frac{1}{(1 + e^{-z})^2} e^{-z} \\ &= \frac{1}{(1 + e^{-z})} \cdot \left(1 - \frac{1}{1 + e^{-z}}\right) \\ &= g(z)(1 - g(z)). \end{aligned}$$

How do we fit the parameters?

Logistic regression is a special case of generalized linear models (GLMs). To fit generalized linear models, people usually use the principle of "maximum likelihood". The main motivation for this choice is that it leads to simple formulas and elegant algorithms. There are also statistical optimality theorems that show that maximum likelihood estimation is optimal under certain assumptions. The principle is also intuitively meaningful.

Recall that we are trying to model probabilities as follows:

$$\mathbb{P}[\mathtt{y} = 1 \mid \mathtt{x} = \mathbf{x}; \theta] = h_\theta(\mathbf{x}) \tag{1}$$
$$\mathbb{P}[\mathtt{y} = 0 \mid \mathtt{x} = \mathbf{x}; \theta] = 1 - h_\theta(\mathbf{x}). \tag{2}$$

Above, $\mathtt{x}$ are random variables and θ are fixed parameters of the model. This can be written more compactly as:

$$\mathbb{P}[\mathtt{y} = y | \mathtt{x} = \mathbf{x}; \theta] = (h_\theta(\mathbf{x}))^y \left(1 - h_\theta(\mathbf{x})\right)^{1-y}.$$

Assume we have n training examples of the form $(y^{(i)}, \mathbf{x}^{(i)})$, $i = 1, \ldots, n$ that were generated independently. Let $\mathbf{y} = [y^{(1)}, \ldots, y^{(n)}]^\mathsf{T}$ and stack all vectors $\mathbf{x}^{(i)}$ as row-vectors into matrix $\mathbf{X}$. With these notations we can write:

$$\begin{aligned}\mathbb{P}[\mathbb{y} = \mathbf{y}|\mathbb{X} = \mathbf{X}; \boldsymbol{\theta}] &= \prod_{i=1}^{n} \mathbb{P}[\mathbb{y}^{(i)} = y^{(i)}|\mathbb{x}^{(i)} = \mathbf{x}^{(i)}; \boldsymbol{\theta}] \\ &= \prod_{i=1}^{n} \left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\right)^{y^{(i)}} \left(1 - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\right)^{1-y^{(i)}}.\end{aligned}$$

The function $L(\boldsymbol{\theta}) = \mathbb{P}[\mathbb{y} = \mathbf{y}|\mathbb{X} = \mathbf{X}; \boldsymbol{\theta}]$ when considered as a function of $\boldsymbol{\theta}$ is called the *likelihood function*. More precisely, this is the likelihood of the parameters $\boldsymbol{\theta}$, given the observed data $\mathbf{y}, \mathbf{X}$. Likelihood describes the plausibility of a model parameter value, given specific observed data. In order to fit $\boldsymbol{\theta}$ to the data, the principle of *maximum likelihood* prescribes to maximize the likelihood over all choices of $\boldsymbol{\theta}$:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} L(\boldsymbol{\theta}).$$

It is easier to minimize the negative log-likelihood:

$$\begin{aligned}l(\boldsymbol{\theta}) &= -\log L(\boldsymbol{\theta}) \\ &= -\sum_{i=1}^{n} \left(y^{(i)} \log h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}))\right).\end{aligned}$$

This function can be proven to be convex. Hence, we can minimize it via the gradient descent algorithm. Let's start by working with a single training example $(\mathbf{x}, y)$:

$$\begin{aligned}\frac{\partial}{\partial \theta_j} l(\boldsymbol{\theta}) &= -\left(y \frac{1}{g(\mathbf{x}^\mathsf{T}\boldsymbol{\theta})} - (1 - y) \frac{1}{1 - g(\mathbf{x}^\mathsf{T}\boldsymbol{\theta})}\right) \frac{\partial}{\partial \theta_j} g(\mathbf{x}^\mathsf{T}\boldsymbol{\theta}) \\ &= -\left(y \frac{1}{g(\mathbf{x}^\mathsf{T}\boldsymbol{\theta})} - (1 - y) \frac{1}{1 - g(\mathbf{x}^\mathsf{T}\boldsymbol{\theta})}\right) g(\mathbf{x}^\mathsf{T}\boldsymbol{\theta})(1 - g(\mathbf{x}^\mathsf{T}\boldsymbol{\theta})) \frac{\partial}{\partial \theta_j} \mathbf{x}^\mathsf{T}\boldsymbol{\theta} \\ &= -\left(y(1 - g(\mathbf{x}^\mathsf{T}\boldsymbol{\theta})) - (1 - y) g(\mathbf{x}^\mathsf{T}\boldsymbol{\theta})\right) [\mathbf{x}]_j \\ &= -\left(y - h_{\boldsymbol{\theta}}(\mathbf{x})\right) [\mathbf{x}]_j\end{aligned}$$

where $[\mathbf{x}]_j$ denotes the jth component of $\mathbf{x}$ and we used that $g'(z) = g(z)(1 - g(z))$ as derived above.

Returning to the case when we have n data points, this gives us the following stochastic gradient descent algorithm. Repeat until convergence:

$$\begin{aligned}&\text{for } i \text{ in } [1 : n] : \\ &\qquad \theta_j := \theta_j + \alpha \left(y^{(i)} - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\right) [\mathbf{x}^{(i)}]_j \text{ for every } j.\end{aligned}$$

To get the batched gradient descent algorithm, just sum over i in the formula above.

This is the same rule that we had for linear regression with the only difference that

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{x}^\mathsf{T}\boldsymbol{\theta}$$

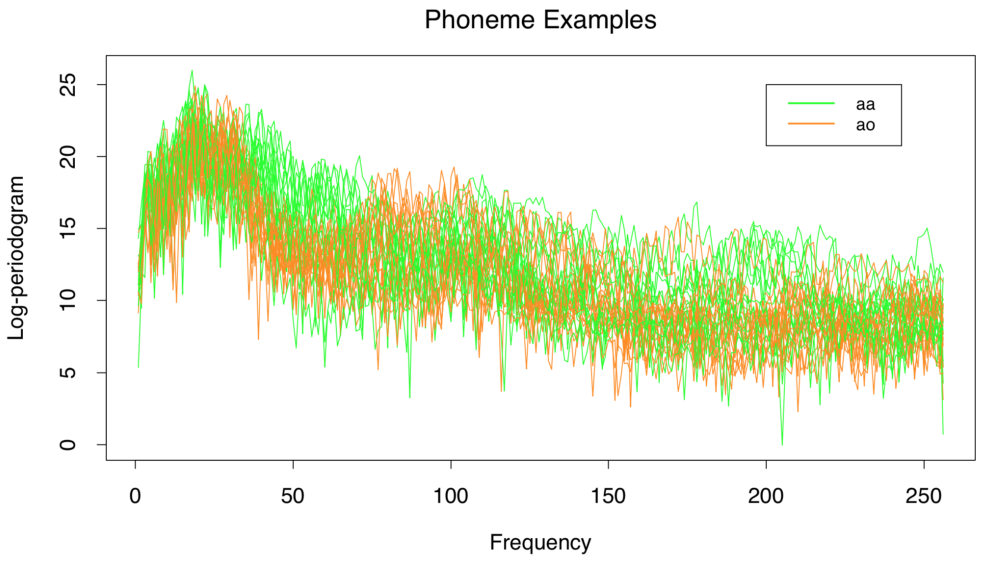
for linear regression, and now it is a nonlinear function

$$h_\theta(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}^\mathsf{T}\theta}}.$$

This is not a coincidence because both models belong to the same general class: generalized linear models.

2 Phoneme recognition example

We now have all the tools we need to do speach-to-text conversion. In the figure below we see 15 log-periodograms for each of the two phonemes "aa" and "ao" that are really difficult to distinguish:

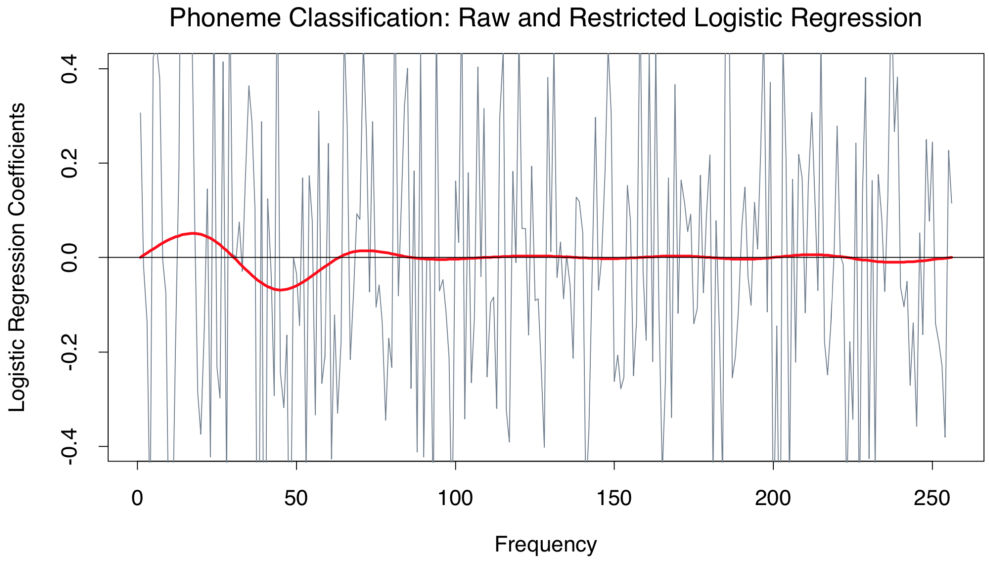


Log-periodogram is $x(f) = \log|\hat{s}(f)|^2$, where $\hat{s}(\cdot)$ denotes the Fourier transform of $s(\cdot)$. We quantized the frequency, f, to make $x(f)$ a $p = 256$ dimensional vector.

First let's fit a logistic regression directly in this 256 dimensional space. From the discussion before we can see that for logistic model, the log-likelihood ratio is a linear function of the features:

$$\log\frac{\mathbb{P}(\text{ao}|\mathbf{x}=\mathbf{x})}{\mathbb{P}(\text{aa}|\mathbf{x}=\mathbf{x})} = \log\frac{\frac{1}{1+\exp(-\int x(f)\theta(f)df)}}{\frac{\exp(-\int x(f)\theta(f)df)}{1+\exp(-\int x(f)\theta(f)df)}} = \int x(f)\theta(f)df \approx \sum_{j=1}^{256} x(f_j)\theta(f_j) = \sum_{j=1}^{256} x_j\theta_j.$$

The coefficients θ_j compute a contrast function and will have appreciable values in regions of frequency where log-periodograms differ between the two classes. In the following figure, we see (the black line) the coefficients θ_j of the logistic regression model fit to a training sample of 1000 ($1000 > 256$) examples:



The black line is very jerky. This is because the values of $x(f)$ for nearby frequencies f_1 and f_2 are highly correlated. For example, if for some frequency, say $j = 101$, the model chose a coefficient θ_j that is too *large and positive*, it tries to compensate for that error by setting the coefficient for $j = 102$ to a *large negative value*. Also observe that we only have about $1000/256 = 4$ training examples per degree of freedom in the model, which is little. Therefore, before we do learning, it might be a good idea to reduce the dimensionality of the problem. To do so, we can force the coefficients to vary smoothly as a function of frequency. When we enforce smoothness, we obtain the red line in the figure above. This immediately makes the solution more interpretable: low frequencies provide most of discriminative power. We also get a much more accurate classifier.

	Raw	Regularized
Training Error	0.080	0.185
Test Error	0.255	0.158

The improvement can be attributed to the fact that the regularized model averages over the nearby data points and provides denoising.

We can use cubic splines to enforce smoothness of $\theta(f)$. Take 8 knots uniformly places over $1, 2, \ldots, 256$. According to the formula derived in the previous lecture this generates a system of splines with

$$D = (9 \text{ regions}) \times (4 \text{ parameters per region}) - (8 \text{ knots}) \times (3 \text{ constraints per knot}) = 12$$

basis functions (degrees of freedom). The 12 basis functions in this case read:

$$
\begin{aligned}
h_1(x) &= 1 \\
h_2(x) &= x \\
h_3(x) &= x^2 \\
h_4(x) &= x^3 \\
h_5(x) &= (x-\varepsilon_1)_+^3 \\
h_6(x) &= (x-\varepsilon_2)_+^3 \\
h_7(x) &= (x-\varepsilon_3)_+^3 \\
h_8(x) &= (x-\varepsilon_4)_+^3 \\
h_9(x) &= (x-\varepsilon_5)_+^3 \\
h_{10}(x) &= (x-\varepsilon_6)_+^3 \\
h_{11}(x) &= (x-\varepsilon_7)_+^3 \\
h_{12}(x) &= (x-\varepsilon_8)_+^3.
\end{aligned}
$$

Require that $\theta(f)$ belongs to the span of these basis function. Mathematically, this means that $\theta(f)$ can be written as

$$\theta(f) = \sum_{m=1}^{D} h_m(f)\beta_m$$

with some coefficients $\beta_m, m = 1, \ldots, 12$ that are to be estimated. For discrete set of frequencies $f_j = 1, \ldots, 256$ we can now write

$$\boldsymbol{\theta} = \mathbf{H}\boldsymbol{\beta}$$

where $\boldsymbol{\theta} = [\theta(f_1), \ldots, \theta(f_{256})]^\mathsf{T}$ and $\mathbf{H}$ is $p \times D$ basis matrix of cubic splines whose elements are $\mathbf{H}_{jm} = h_m(f_j)$.

Since $\mathbf{x}^\mathsf{T}\boldsymbol{\theta} = \mathbf{x}^\mathsf{T}\mathbf{H}\boldsymbol{\beta}$, we can change variables: $\mathbf{x}^* = \mathbf{H}^\mathsf{T}\mathbf{x}$ and fit $\boldsymbol{\beta}$ by a linear logistic regression on $\mathbf{x}^*$. The fitted coefficients are denoted $\hat{\boldsymbol{\beta}}$. The red curve is

$$\hat{\theta}(f) = \sum_{m=1}^{D} h_m(f)\hat{\beta}_m = \mathbf{h}^\mathsf{T}(f)\hat{\boldsymbol{\beta}}$$

where $\mathbf{h}(f) = [h_1(f), \ldots, h_D(f)]^\mathsf{T}$.

Preprocessing of high dimensional features is a very powerful method of improving learning algorithms.