MLISP: Machine Learning in Signal Processing

Lecture 9

Prof. V. I. Morgenshtern

Scribe: M. Elminshawi

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

Agenda:

1. Multi-dimensional splines
2. Learning adversarial classifiers in high dimensions
3. Models beyond smoothness
4. A problem with Fourier transform
5. Short time Fourier transform

1 Multi-dimensional splines

Suppose $\mathbf{x} = [x_1, x_2] \in \mathbb{R}^2$ and we have a set of basis functions

$$h_{1k}(x_1),\ k = 1, 2, \dots, M_1$$

for representing functions of coordinate x_1, and likewise

$$h_{2k}(x_2),\ k = 1, 2, \dots, M_2$$

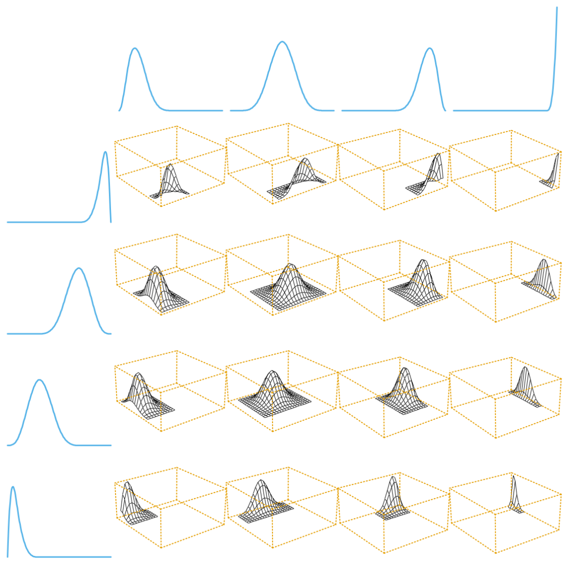
for coordinate x_2. Then we can build the $M_1 \times M_2$ dimensional *tensor basis*:

$$h^{2D}_{jk}(\mathbf{x}) = h_{1j}(x_1)h_{2k}(x_2),\ j = 1, 2, \dots, M_1,\ k = 1, 2, \dots, M_2.$$

With this basis we can represent two-dimensional functions:

$$h^{2D}(\mathbf{x}) = \sum_{j=1}^{M_1}\sum_{k=1}^{M_2} \theta_{jk} h^{2D}_{jk}(\mathbf{x}).$$

Here is an example of the tensor basis in $2D$:



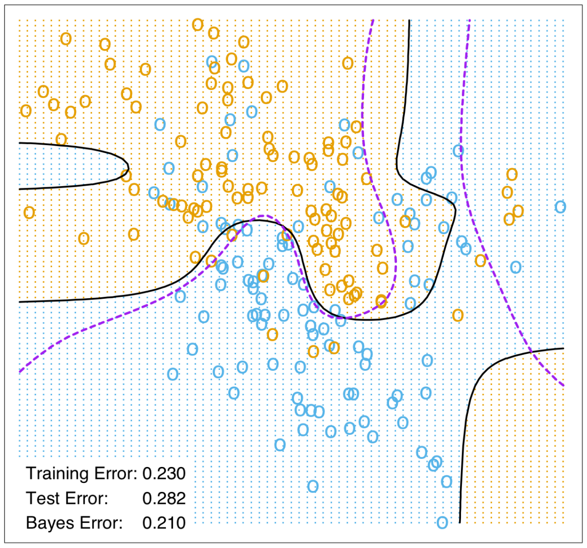
Using this construction, we can fit the logistic regression

$$\text{logit}[\mathbb{P}(\text{ORANGE}|\mathbf{x})] = \mathbf{h}^{2D}(\mathbf{x})^T\boldsymbol{\theta}$$

to the ORANGE and BLUE points dataset with $M_1 = M_2 = 4$. The logit function is the inverse of the logistic (sigmoid) function:

$$\text{logit}(p) = g^{-1}(p) = \log\left(\frac{p}{1-p}\right).$$

Here is the result:



The contour $\mathbf{h}^{2D}(\mathbf{x})^T\boldsymbol{\theta} = 0$ is plotted in purple and the Bayesian decision boundary for this problem is in black.

The construction can be generalized to d dimensions with the tensor basis given by

$$h^{d-D}_{j_1 j_2 \cdots j_d}(\mathbf{x}) = h_{1j_1}(x_1) \times \ldots \times h_{dj_d}(x_d),\ j_d = 1, 2, \ldots, M.$$

Note that the number of basis functions is M^d, i.e., it is exponential in d. This is a yet another manifestation of the curse of dimensionality.

2 Learning adversarial classifiers in high dimensions

Suppose that we have a 256-dimensional signal $\mathbf{x} = [x_1, \ldots, x_{256}]$, as in the phoneme recognition example. Let's make a simplifying assumption: each x_i can only take one of two values, either 0 or 1. How many possible signals there are? Clearly, 2^{256}. Suppose that each of these 2^{256} signals belongs either to class 0 (encoding 'aa') or to class 1 (encoding 'ao'). Of corse, this is not a real situation: in the real world, most of the high-dimenstional signals have no meaning and belong to the class 'noise'. However, let's make the assumption that there are only two classes for the sake of the argument. For each $\mathbf{x} \in \{0, 1\}^{256}$, let $y = f(\mathbf{x})$ denote the correct answer, i.e., class to which the signal belongs:

$$f : \{0, 1\}^{256} \to \{0, 1\}.$$

The function f is chosen by nature and is hidden from us. Can we learn the classification function f from data? How many data points are needed? Can we learn it using a linear classifier, such as the linear or the logistic regression?

In the general case, we need 2^{256} data points, which is never available. Why? Suppose that the function $f(\cdot)$ did not have any particular structure: it has been generated by nature randomly: for each $\mathbf{x} \in \{0, 1\}^{256}$, $f(\mathbf{x})$ was chosen to be 0 or 1 with probability 1/2 for each outcome. In this case, there is no other way to learn $f(\cdot)$, except for memorizing $f(\mathbf{x})$ for each possible $\mathbf{x}$. This requires a database with 2^{256} data points.

Can we express the memorization process via a linear model? Yes, if we use 2^{256} features. Concretely, our underlying feature vector is $[x_1, \ldots, x_{256}]$. From this, we can generate a new feature vector of dimension 2^{256}:

$$\tilde{\mathbf{x}} = [\tilde{x}_1, \ldots \tilde{x}_{2^{256}}]$$

where only one of the $\tilde{x}_i$'s is nonzero, corresponding to the index of the binary vector $[x_1, \ldots, x_{256}]$ in the ordered list of all 256-dimensional binary vectors (one-hot-vector encoding). Can we express this encoding using, for example, polynomials? Yes. Here is the encoding:

$$\tilde{\mathbf{x}} = \begin{bmatrix} (1-x_1)(1-x_2) \times \cdots \times (1-x_{255})(1-x_{256}) \\ (1-x_1)(1-x_2) \times \cdots \times (1-x_{255})x_{256} \\ \vdots \\ (1-x_1)(1-x_2) \times \cdots \times x_{255}x_{256} \\ \vdots \\ (1-x_1)x_2 \times \cdots \times x_{255}x_{256} \\ \vdots \\ x_1 x_2 \times \cdots \times x_{255}x_{256} \end{bmatrix}.$$

Above, all possible combinations of terms of the type x and $(1-x)$ are listed in the lexicographic order. Each new feature is the a polynomial of degree 256 in the old features. We have used exponentially many (in $d = 256$) such polynomials. (Think about how you could also use the d-dimensional tensor splines from the previous subsection here.) It is not difficult to conclude that in the new feature space there is a linear classifier that correctly separates the data:

$$I(\tilde{\mathbf{x}}^{\mathsf{T}}\hat{\boldsymbol{\theta}} > 0) = f(\mathbf{x}) = y$$

for all $\mathbf{x}$, where $\hat{\boldsymbol{\theta}} = [\hat{\theta}_1, \ldots, \hat{\theta}_{2^{256}}]$. Of corse, learning all the 2^{256} parameters of $\hat{\boldsymbol{\theta}}$ (more than then number of atoms in the universe) is never possible from data.

Which classification functions can we learn in high dimensions? What was special about the phoneme recognition example so that we have successfully learned to separate 'aa' from 'ao' phonemes?

We can learn only those functions that are 'simple' in some way, i.e., there is structure in the function and a corresponding simple decision rule. Random decision functions have no structure. On the contrary, the function

$$f(\mathbf{x}) = \begin{cases} 0, & \text{if} \quad \sum_{i=1}^{256} x_i > 128 \\ 1, & \text{otherwise.} \end{cases}$$

is a function with structure and therefore is reasonably easy to learn from data.

Most of the functions are random, without structure. At the same time, it is an empirical fact that most of the functions of interest in science and engineering are structured and simple.

For example, consider the function that classifies all images into two categories: there is a cat in the image / there is no cat in the image. This is a simple function (which might not be obvious) because humans and deep neural networks can learn it from a reasonably small amount of training data. The fundamental reasons why this function is simple are (i) the images with cat represent a tiny fraction of all possible images; (ii) to recognize a cat it is sufficient to recognize few of its basic features: cat's nose, cat's ears, cat's fur and so on; in other words this function can be represented as a hierarchical composition of even simpler functions and those can be learned from data directly. We will return to this point of compositionality when we study neural networks in the last part of the class.

Another example of a simple function is the classifier of phonemes that we learned earlier:

$$f(\mathbf{x}) \approx I[g(\mathbf{x}^{\mathsf{T}}\mathbf{H}\boldsymbol{\theta}) > 0.5]$$

where $I[\cdot]$ is the indicator function and $g(z) = 1/(1 - \exp(-z))$. This function is simple because the decision function turned out to be a smooth function of frequency. There are few contiguous intervals on the frequency axis with the property that all frequencies within those contiguous intervals affect the classification function in the same way. Therefore, the 256 dimensional space effectively becomes 12 dimensional. Even a very complex function of 12 variables can be learned from data directly.

3 Models beyond smoothness: invitation to wavelets

Recall the example of phoneme classification from the previous lecture. In the example of audio classification, we have seen how we can improve performance by using splines to smooth the data and regularize the linear classifier. Concretely, our approach can be summarized as follows: For a signal $\mathbf{x} \in \mathbb{R}^{256}$ we would normally need 256 basis functions to represent $\mathbf{x}$ exactly. Instead, we only used $D = 12$ basis functions $h_1(\cdot), \ldots, h_{12}(\cdot)$ and the functions are designed in such a way that every function in the span of $h_1(\cdot), \ldots, h_{12}(\cdot)$ is smooth. The processed data is:

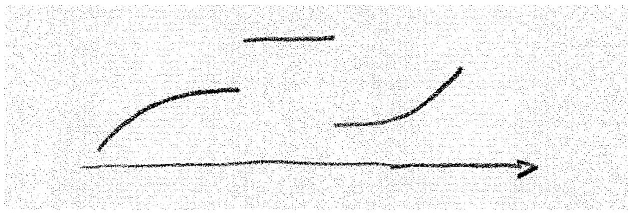
$$\mathbf{x}_m^* = [\mathbf{H}^T \mathbf{x}]_m = \sum_{j=1}^{256} h_m(f_j) x_j = \mathbf{h}_m^\mathsf{T} \mathbf{x}.$$

Let us decompose $\mathbf{x} = \mathbf{x}^{smooth} + \mathbf{x}^{non-smooth}$, where $\mathbf{x}^{smooth}$ is the projection of $\mathbf{x}$ onto the span of $h_1(\cdot), \ldots, h_{12}(\cdot)$, and $\mathbf{x}^{non-smooth}$ is the projection of $\mathbf{x}$ onto the orthogonal complement of the span of $h_1(\cdot), \ldots, h_{12}(\cdot)$. Then,

$$\begin{aligned} \mathbf{h}_m^\mathsf{T} \mathbf{x} &= \mathbf{h}_m^\mathsf{T} (\mathbf{x}^{smooth} + \mathbf{x}^{non-smooth}) \\ &= \mathbf{h}_m^\mathsf{T} \mathbf{x}^{smooth} + \underbrace{\mathbf{h}_m^\mathsf{T} \mathbf{x}^{non-smooth}}_{0} \\ &= \mathbf{h}_m^\mathsf{T} \mathbf{x}^{smooth}. \end{aligned}$$

We see that the non-smooth part of the data does not affect the coefficients $\mathbf{x}_m^*$, and hence does not affect learning. This is filtering: the non-smooth part of signal happens to be irrelevant for separation of ‘ao’ and ‘aa’ phonemes. Hence, removing that part improves the outcome.

Often, our true signal is far from smooth:



Discontinuities are important part of natural phenomena. For example, in images edges carry an important information:

Wavelets are the tools that allow us to capture structures beyond smoothness and still separates out the irrelevant part.

4 A problem with Fourier transform

Fourier transform reveals important information about a signal: its frequency content. If the function $f(t)$ is periodic on the interval $[-\pi, \pi]$, it can be written as the Fourier series:

$$f(t) = \sum_{n=-\infty}^{+\infty} c_n \phi_n(t)$$

where

$$c_n = \langle \phi_n, f \rangle = \int_{-\pi}^{\pi} \phi_n(t) f(t) dt$$

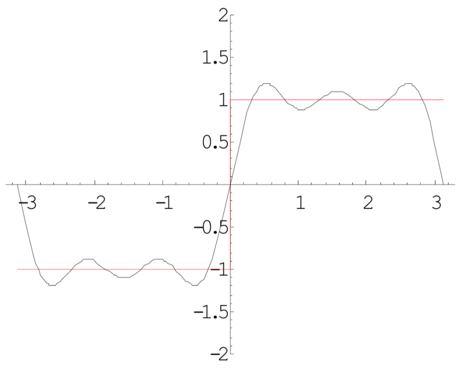
and the basis functions are

$$\phi_n(t) = \frac{1}{\sqrt{2\pi}} e^{int}.$$

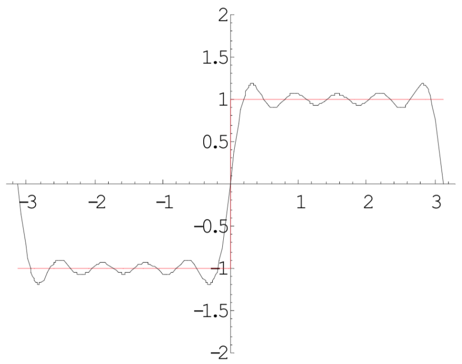
Importantly, the Fourier basis function form an orthonormal basis: $\langle \phi_n, \phi_n \rangle = 1$ and $\langle \phi_n, \phi_k \rangle = 0$ if $n \neq k$. This property makes all computations and algorithms very simple.

If $x(t)$ represents a piece of music, then the Fourier coefficients $\{c_n\}$ reveal the tones that form that piece of music.

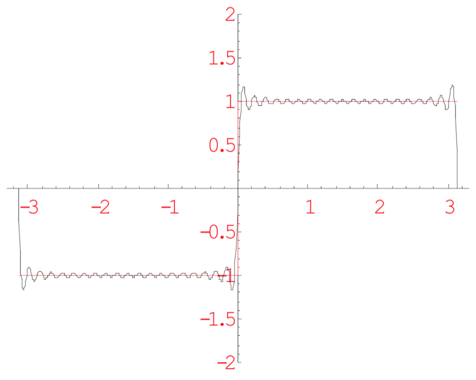
The problem is that Fourier transform is a global transformation, a local change in the signal affects the Fourier transform everywhere. Concretely, consider a function with one discontinuity. The first 5 terms of Fourier series approximate the function as follows:



The first 20 terms of Fourier series approximate it as follows:



The first 40 terms of Fourier series approximate it as follows:



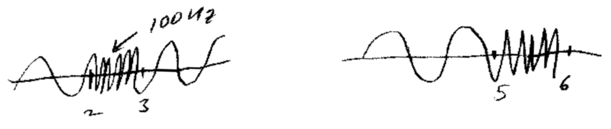
We observe a jump of about 9% near the singularity, no matter how many terms we take. In general, singularities (discontinuities in $f(x)$ or in its derivatives) cause high frequency coefficients (c_n with large n) in the Fourier series

$$f(t) = \frac{1}{\sqrt{\pi}} \sum_{n=-\infty}^{+\infty} c_n e^{int}$$

to be large. Note that the singularity is only in one point, but it causes all c_n (for large n) to be large. This is bad for two reasons:

(1) If there is even one singularity in the signal, then many Fourier coefficients will be large. Hence, we cannot use Fourier coefficients for efficient signal compression.

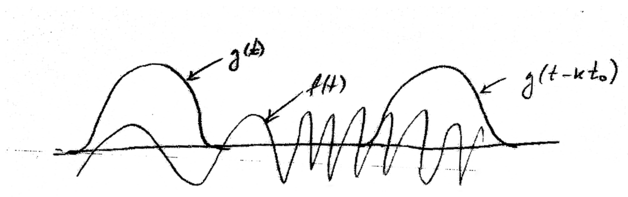
(2) The Fourier series tells us nothing about the time location of an interesting features in the signal. For example, consider two signals:



For both signals c_{100} is large, but the information on where in time does the frequency burst happen is not immediately available.

5 Short time Fourier transform

One possible solution for the frequency localization problem is to use the window function like this:



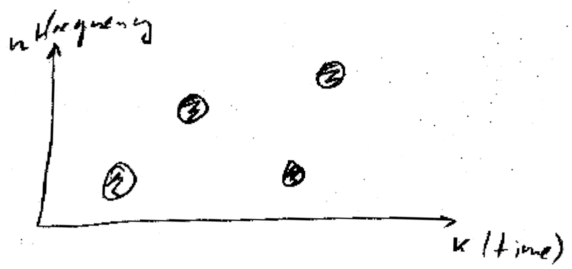
and calculate that Fourier coefficients for the windowed function

$$f(t) \cdot g(t - kt_0).$$

Specifically

$$c_{nk} = \frac{1}{\sqrt{2\pi}} \int_{-\pi}^{\pi} f(t) g(t - kt_0) e^{-int} dt.$$

This is the short-time Fourier transform. It decomposes the 1D signal into the 2D time-frequency representation (k indexes time, n indexes frequency) like this:



This representation is very similar to the way people write the sheet music. Such representation is often useful: for example, the Shazam algorithm for music recognition is based on it.

Note however that the functions

$$g_{nk}(t) = g(t - kt_0) e^{int}$$

are not orthogonal, unlike $\phi_n(t) = \frac{1}{\sqrt{2\pi}} e^{int}$ in Fourier series. Therefore these functions do not form a nice basis. We need something better.