PG Diploma in Data Analytics

Course: Predictive Analytics - II

Instructor: Prof. G. Srinivasaraghavan

PGDDA-Lecture Notes/PA-II/1

Advanced Regression

Contents

List of Figures

List of Tables

1 Learning Objectives

After this module you are expected to be able to carry out regression (fit numerical models for prediction) on complex data sets. The module builds on what you have already learnt as part of linear regression in Predictive Analytics - I.

1. Build the Conceptual Foundation for Generalized Linear Regression:
   (a) Understand what is 'linearity' in this context and why it is desirable.
   (b) Be able to choose an appropriate family of functions for a particular regression task.
   (c) Build geometric intuition about regression in general.
2. Understand how various popular regression algorithms are variants of the generalized linear regression framework developed earlier.
   (a) Relate to and be able to use the standard linear regression implementations such as `lm` and `glmnet` in various popular libraries such as R, Matlab etc. in terms of the framework and be able to use them effectively.
   (b) Understand the implications of the various parameters used by these implementations and be able to tune them appropriately for specific regression problems in practice.
3. Understand how non-linear regression is different from linear regression and circumstances under which non-linear regression may be required.
   (a) Understand how non-linear regression works.
   (b) Be able to identify when non-linearity might be required.
   (c) Be able to use non-linear regression implementations such as `nls` in R to solve non-linear regression problems.
4. Learn to work with generalized linear models (GLM)
   (a) Understand generalized linear models and how they are different from the 'generalized linear regression' discussed above.
   (b) Be able to use GLM implementations such as `glm` in R to solve regression problems.
5. Evaluation / Comparison of regression solutions.
   (a) Be able to evaluate and assess any given regression solution for a given dataset.
   (b) Be able to objectively compare regression solutions and make a choice.
   (c) Learn how to read the summary of a model produced by implementations like `lm` and `nls` in R.

2 Legend and Conventions, Notation

There are several parts of the test that have been highlighted (in boxes). The document uses three kinds of boxes.

Example 0: Title

One of the running examples in the document. The reader must be able to reproduce the observations in these examples with the corresponding R code provided in Section **??**.

Title

Good to remember stuff. These boxes typically highlight the key take-aways from this document.

Title

Things that are good to be aware of. This is primarily targeted at the reader who wishes to explore further and is curious to know the underlying (sometimes rather profound) connections with other fields, alternative interpretations, proofs of some statements, etc. However one could safely skip these boxes completely if you are only interested in the primary content/message of this document.

Title

Code snippets. Implementation Tips. Do's and Dont's.

$\boldsymbol{x}$	A *vector* $\boldsymbol{x}$
x	A scalar quantity x
term	Refer to the glossary as a quick reference for 'term'. These are typically prerequisite concepts that the reader is expected to be familiar with.
`code`	Code snippets, actual names of library functions, etc.

3 Introduction

Regression was introduced in Predictive Analytics-I. We will build on what we have learnt from the first course on Predictive Analytics, to explore regression a little further. We will deliberately explore regression in a slightly more generalized form than what you may have been exposed to till now. This would expand the scope of regression significantly, as you would see very soon and also make regression lot more robust. As a quick recap, here's a short quiz (11.1) for you to try recollecting some of what was covered in the first course. Note that some of this may sound a little counter-intuitive at first, may be even contradictory to what you would have learnt till now. However these apparent contradictions will disappear after a little thinking through and of course when we work through the rest of this module.

Regression in general is a statistical technique that allows you to make statements like the ones illustrated below. We will use the following as running examples throughout this module.

Example 1: Sales Figures Regression

The sales figures S_m of a company between the years 1965 and 1971 as a function of the number of months m starting from Jan 1965 is given by

$$S(m) = 30.99 * \sqrt{m} * \cos(0.53 * m) + 5.83 * m + 76.61$$

This expression is arrived at using regression on the dataset at `https://datamarket.com/data/set/22mp/sales-of-company-x-jan-1965-to-may-1971#!ds=22mp&display=line` in the Time Series data repository [1]. Figure 1 shows the raw data (as points) and the curve obtained through a regression fit, as a red line. The R code used to generate this plot is given below.

```
sdata <- read.csv("sales-data.csv")
plot(sdata, xlab='Month', ylab='Sales', type='b')
a_start <- -10
b_start <- 0.53
c_start <- 100
d_start <- 200
mymodel <- nls(Sales ~ a * sqrt(Month) * cos(b*Month) + c * Month + d,
               data=sdata,
               start=list(a=a_start, b=b_start, c=c_start, d=d_start),
               trace=TRUE)
with(sdata,
     lines(seq(1, 77, by=1),
     predict(mymodel, data.frame(Month=seq(1, 77, by=1))),
     col='red', lwd=2)
     )
```

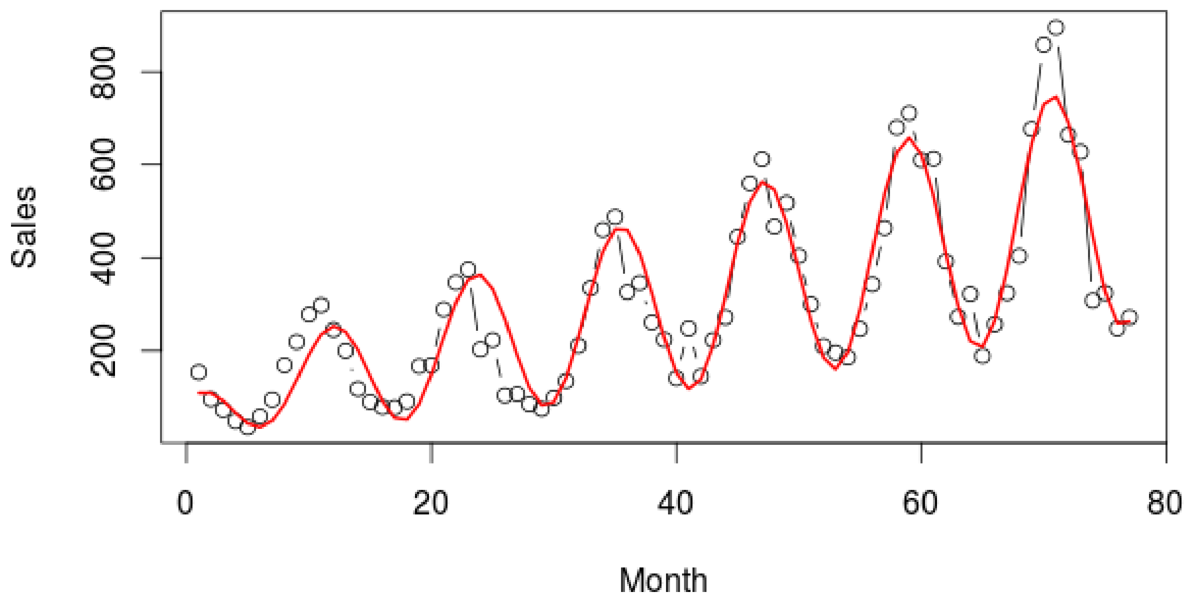

Figure 1: Sales Figures Regression

Example 2: Total Electricity Consumption in the U.S

The Total electricity consumption in the U.S. (in Million kilowatt-hours) in any any year y during the period 1920–1970 is given by

$$C(y) = -112637300000 + 175430200.0 * y - 91078.44 * y^2 + 15.76219 * y^3$$

This expression is arrived at using regression on the dataset at `https://datamarket.com/data/set/22vi/total-electricity-consumption-us-kilowatt-hours-millions-1920-1970#!ds=22vi&display=line` in the Time Series data repository [1]. Figure 2 shows the raw data (as points) and the curve obtained through a regression fit, as a red line. The R code used to generate this plot is given below.

```
cdata <- read.csv("total-electricity-consumption-us.csv")
plot(cdata, xlab='Year', ylab='Consumption', type='b')
mymodel <- lm(Consumption ~ Year + I(Year^2) + I(Year^3), data=cdata)
with(cdata,
      lines(seq(1920, 1970, by=1),
             predict(mymodel, data.frame(Year=seq(1920, 1970, by=1))),
             col='red', lwd=2
      )
)
```

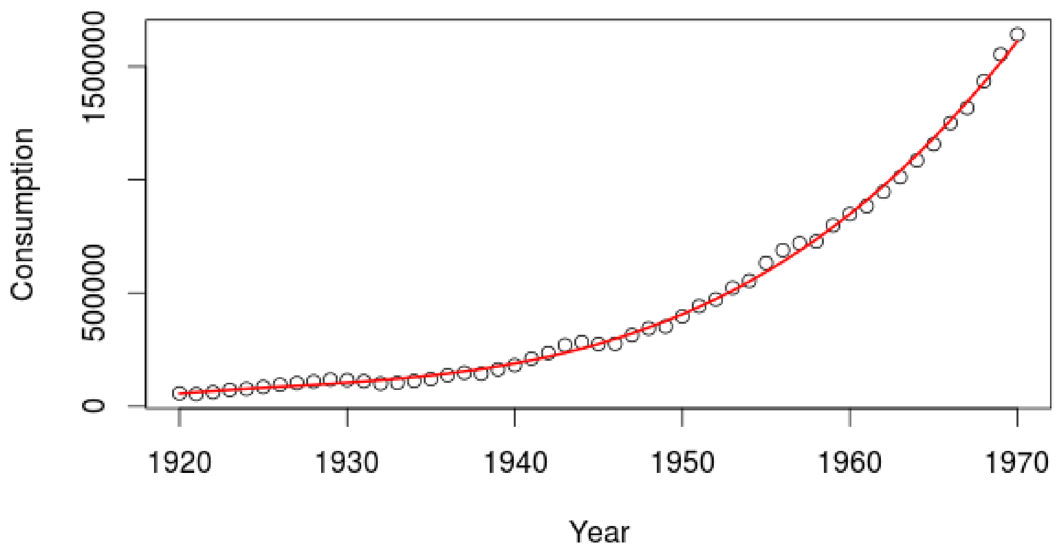

Figure 2: US Electricity Consumption Regression

Example 3: Monthly Measles Cases in Baltimore

The total number of monthly measles cases reported in Baltimore (U.S) in the period Jan 1939 to June 1972 is given by

$$C(y,m) = -112637300000 + 175430200.0 * y - 91078.44 * y^2 + 15.76219 * y^3$$

where $1939 \leq y \leq 1972$ is the year and $1 \leq m \leq 12$ is the month. This expression is arrived at using regression on the dataset at `https://datamarket.com/data/set/22lx/monthly-reported-number-of-cases-of-measles-baltimore-jan-1939-to-june-1972#!ds=22lx&display=line` in the Time Series data repository [1]. Figure 3 shows the raw data (as points) and the curve obtained through a regression fit, as a red line. The R code used to generate this plot is given below.

```
code
```

We will develop the intuition and the theory behind carrying out such regressions in the rest of this module.

In this module we will first develop a broad intuitive framework for linear regression in its more general form. We will then examine regularization that is essential for making algorithms in machine learning less brittle. We put the two together to finally describe the full form of linear regression, the way it is most often used in practice. Later we also examine a few other advanced concepts related to regression such as Generalized Linear Models followed by Kernel Regression that will also form a bridge to the next module on Kernel Methods in this course.

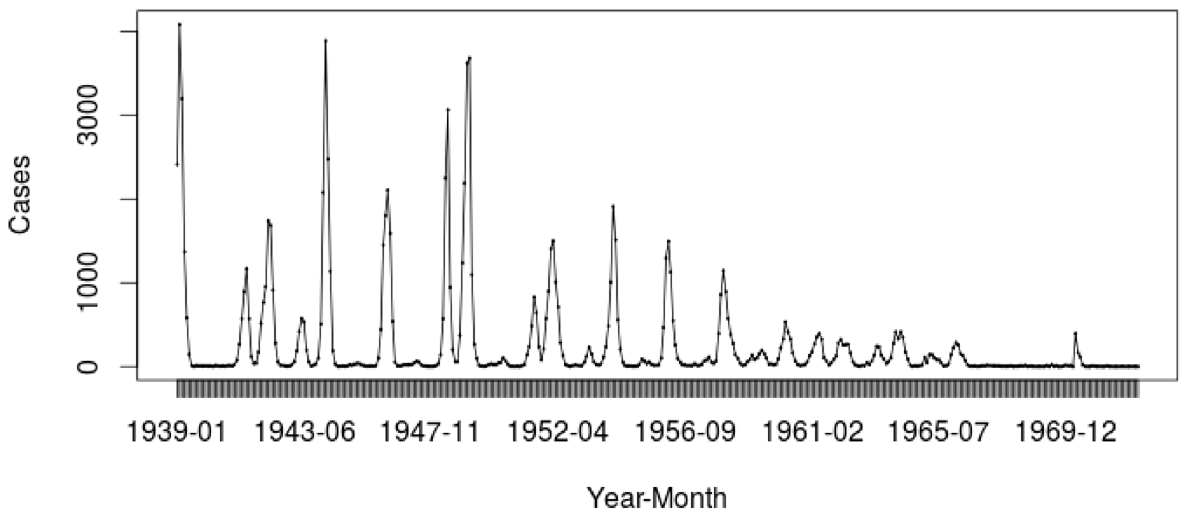

Figure 3: Monthly Measles Cases Reported in Baltimore between Jan 1939 and June 1972

4 Generalized Linear Regression — A Broad Framework

In general this is how regression works. We are given training data consisting of n entities with d numeric attributes each (called the *explanatory* variables) that we believe determine a numeric *response* attribute.

Regression Assumes an I.I.D Dataset

This is a crucial assumption in regression — that the data instances in the dataset have all been sampled **independently** from the **same distribution**. The acronym *iid* stands for *Independent and Identically Distributed.* One way to interpret this statement is that the data instances have no systematic bias in them. Suppose I am trying to find the relationship between say height, age and weight among Indians and want to write weight as a function of height and age. The idea is to carry out a regression on age-height-weight data collected from several individuals. In this case the iid assumption means:

1. (**Independent**) The samples are not biased towards say 'obese' individuals. In other words the proportion of samples from obese people is not statistically unrealistic for the Indian population.
2. (**Identically Distributed**) The relationship is expected to be different for young children, middle-aged adults and senior citizens. The distributions for the three categories are not expected to be the same. The iid assumption then implies that the data samples are all from a 'homogeneous' group — it is not the case that some data is for

senior citizens above 80 and some for children below 10.

The iid assumption is the key difference in the connotations of the terms *regression* and *time series analysis*. Regression assumes iid data and time-series analysis usually doesn't. We will discuss time series analysis in the subsequent module.

We denote the d explanatory variables together for the i^{th} entity as the *vector* $\boldsymbol{x}_i$ and the value of the response variable as y_i. Our aim is to arrive at a *function* $f(\boldsymbol{x})$ such that $\forall 1 \le i \le n, f(\boldsymbol{x}_i) \approx y_i$. In other words we intend to arrive at a function f that takes the explanatory attributes $\boldsymbol{x}$ as input and predicts the value of the response variable y as accurately as possible. The function is expressed in a parametrized form (see the discussion below) — the training results in the values of the function parameters.

Geometric Interpretation of Data

We often use geometric interpretations of data, interchangeably with other (algebraic) representations and interpretations, in much of machine learning and regression is no exception. We recap some of these in this box. An entity with d attributes (that can be reduced to numeric values) can be thought of as a point or a vector in d-dimensional space where each axis corresponds to one of the attributes of the object. A function $f(\boldsymbol{x})$ of a d-dimensional vector $\boldsymbol{x}$ outputs a value given values for each of the components (attributes) of the vector $\boldsymbol{x}$. The equation $f(\boldsymbol{x}) = c$ for some constant c represents a d-dimensional surface — it connects all points $\boldsymbol{x}$ for which $f(\boldsymbol{x}) = c$ — these are called the **contours** of $f(x)$. Refer the Wikipedia page `https://en.wikipedia.org/wiki/Contour_line` for an extensive list of contexts where contours are used. The function $f(\boldsymbol{x})$ can also be thought of as a $(d+1)$-dimensional surface which is at a 'height' $f(\boldsymbol{x})$ (along the $(d+1)^{th}$ dimension, given the values for the d-attributes (the first d coordinates). See Figures 4 for an illustration. When we plot data or refer to data as a point, we are essentially resorting to this geometric interpretation of data. Figure 5 shows another interesting example — this is a contour plot of the ocean depth for different latitudes and longitudes.

Parametrized Family of Functions

Examples:

- For functions with 2 arguments x_1 and x_2, the function family $\mathscr{F}(a,b,c \mid x_1,x_2) = a + b.x_1 + c.x_2$ defines the class of all straight lines on a plane formed by x_1 and x_2 as the two axes. Every choice of the parameters a,b,c determines a line in this family and any straight line involving x_1 and x_2 can be expressed this way.

- For functions with 3 arguments x_1, x_2, x_3, the function family

$$\mathscr{F}(a_0, a_i, b_i, c_i \mid x_1, x_2, x_3) = a_0 + \sum_{i=1}^{3} a_i.x_i + \sum_{i=1}^{3} b_i.x_i^2 + \sum_{i=1}^{3} c_i.x_i^3$$

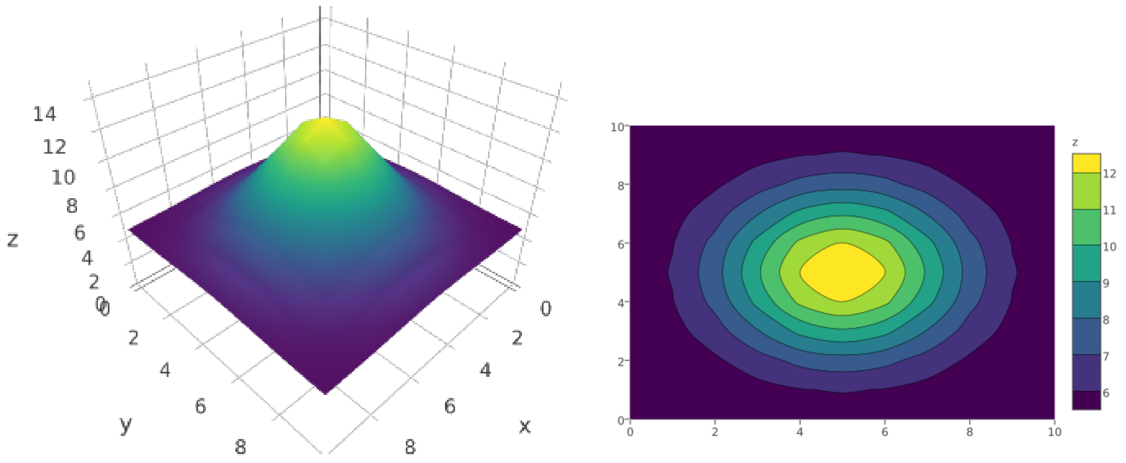

Figure 4: Gaussian-like function $z = f(x, y) = 5 + 8 * \exp\left(\frac{-(x-5)^2-(y-5)^2}{8}\right)$ in 2 Variables. The function value is shown in the z-axis in the figure on the left. The right hand side figure shows the contours for $f(x)$.

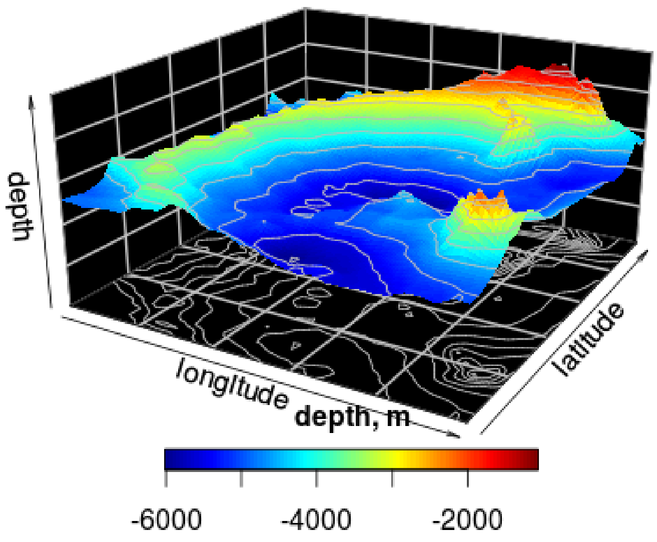

Figure 5: Ocean Depth Contours at various latitudes and longitudes, over a 'patch' on the Atlantic. This dataset and plot is taken from the 'plot3D' package of R.

defines the class of cubic functions that do not account for combinations of variables. The parameters in this case are $a_0, a_i, b_i, c_i, 1 \le i \le 3$.

- Functions with just one argument x, $\mathscr{F}(a, b, c \mid x) = a + b.\sin x + c.e^x$ defines the class of all functions that can be expressed as a combination of a sinusoidal function and an exponential function.

A parametrized family of functions defines an entire family of functions with a few parameters — each function in the family is determined by a choice of specific values for the parameters, and vice-versa. In Regression we typically start with such a parametrized family of functions and then determine the values of the parameters for which the function defined by the chosen parameter values fits the data best. It is also common to represent the set of parameters of a family of functions as a vector.

What we refer to as the *model* produced by the regression algorithm after training, is the collection of parameter values (equivalently the actual function $f(\boldsymbol{x})$) determined by the training. For instance if the parameter values determined after training are $a, b, c = 1, 2, 3$ for the first example in the box above, then the best fit model is the function $f(x_1, x_2) = \mathscr{F}(1,2,3 \mid x_1, x_2) = 1 + 2x_2 + 3x_2$. The typical process one would follow for carrying out the regression on the given dataset is as follows.

1. Pick an appropriate family of functions that you think would be reasonable (not too complex, not too simplistic and näive for the problem). This could be done by a little bit of exploratory data analysis - visually examining how the data points are distributed. Some thumb rules for instance could be:

   - 'Flatter' the arrangement of points looks, less 'complex' the family of functions should be — for lack of a better guess, a linear function is invariably the best bet.
   - If there is some periodicity in the way the points are arranged, then some of the trigonometric functions may be appropriate.
   - If the values taken by the response variable vanish (to say zero) for large values of the explanatory variables — think in terms of inverse exponential functions or even a Gaussian.
   - If the values taken by the response variable increases very rapidly with increasing values of the explanatory variables, then try higher degree polynomials or even exponential functions.

   Suppose we fix a set of k **basis** functions $\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \ldots, \phi_k(\boldsymbol{x})$ — the idea is that we expect the final function to be a *linear combination* (weighted sum) of these basis functions. So essentially we are talking about the family

   $$\mathscr{F}(a_0, a_1, \ldots, a_k \mid \boldsymbol{x}) = a_0 + \sum_{j=1}^{k} a_j.\phi_j(\boldsymbol{x}) = \mathscr{F}(\boldsymbol{a} \mid \boldsymbol{x}) = \langle \boldsymbol{a}, \boldsymbol{\phi}(\boldsymbol{x}) \rangle$$

   where $\boldsymbol{a}$ is the parameter vector $(a_0, a_1, \ldots, a_k)$, $\boldsymbol{\phi}(\boldsymbol{x})$ is the vector of values $(\phi_0(\boldsymbol{x}), \ldots, \phi_k(\boldsymbol{x}))$, $\langle ., . \rangle$ denotes the dot-product between two vectors, and $\phi_0(\boldsymbol{x})$ is just the constant function 1 introduced just to write the above expression as a neat dot-product between two vectors. A technical assumption that one needs to be aware of is that it is best if the basis functions chosen are *linearly independent* — i.e., none of the basis functions can be written as a sum of the other functions multiplied by some constants. Refer Figure 6 for an illustration of functions and their shapes. In this figure the coefficients have been adjusted to bring all the function to similar scales.

   A few things to note about the illustration.

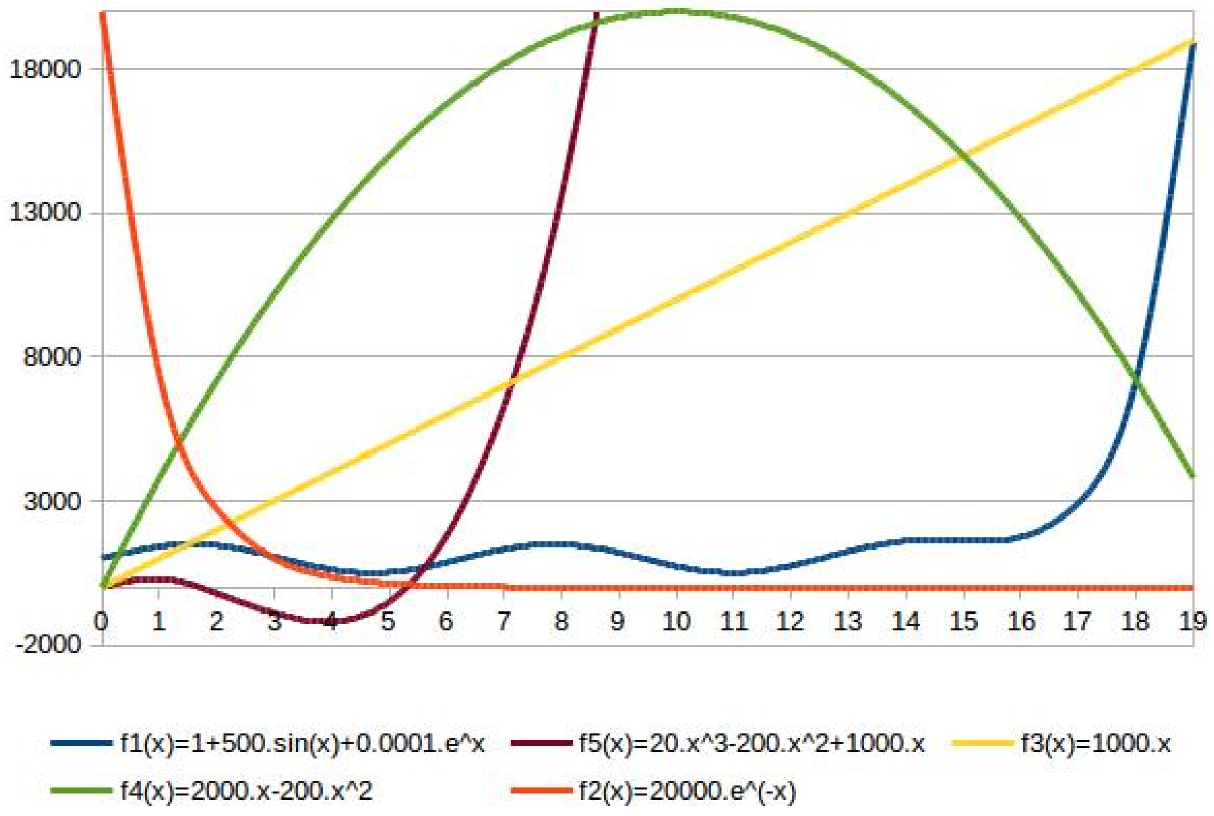

Figure 6: Shapes of Functions

- The function with $\sin x$ and e^x one can see the wavy shape for smaller values of x where $\sin x$ dominates and later e^x takes over resulting in a rapid increase in the function value.
- e^{-x} shows a rapid decline with increasing x.
- The cubic has a hump and a trough, whereas the quadratic has just one hump.

2. Determine the values of the parameters for which the 'cost' of using the model is the smallest. The cost consists of two components — (i) the **error** committed by the model, quantified by comparing the response variable 'predicted' by the model with the actual value given in the training data, and (ii) a regularization term, meant to keep the model complexity under check, that acts as a penalty in case the model becomes way too 'complex' — we will discuss this issue in greater detail below. With specific values for the model parameters, we can now construct the function represented by the parameters. Note that this finds a function *from the chosen family* that best fits the data. If you had chosen a different family of functions, you may have got a better fit or worse.

3. Given a new entity with the numeric attribute (independent explanatory variables) vector $\boldsymbol{x}$, we can now compute the predicted value of the response attribute y, by just plugging in the values from $\boldsymbol{x}$ into our model.

Regression and Noisy Data

Think about this — how many points do you need to provide to define a line in 2-d? Just two. Then why do we need a dataset with 1000's of points to determine the regression line? Recall a question in the quiz about noisy data. Well regression needs noisy data for it to work !! When we carry out regression, what we are assuming is that there is an 'ideal' function from the chosen family that every data point follows: i.e., for each data point with explanatory attribute vector $\boldsymbol{x}$ and response variable y, $f(x)$ is supposed to be exactly equal to y. However the data is noisy, and all real-life data is indeed noisy (due to measurement errors, system instabilities, unknowns, etc.). So in reality $f(x) = z \neq y$ for some z that is close to but not exactly equal to y. Think of y as z with some random noise ϵ. So $y = z + \epsilon$, for some unknown z and ϵ. Regression tries to 're-engineer' this ideal function $f(\boldsymbol{x})$ that is characteristic of the system that produced this noisy data. The function that is computed by the regression algorithm is therefore expected to come as close as possible to each of the points, but is not really expected to exactly pass through any of them. Well we can show that the standard least-squares regression is equivalent to a maximum likelihood estimate on data in which for any given explanatory attribute vector $\boldsymbol{x}$ the response variable y is random following a Gaussian Distribution 'centered' (mean) at the ideal value $f(\boldsymbol{x})$.

The box below introduces the `lm` call in R for linear regression.

Linear Regression in R

The function typically used for linear regression in R is `lm`. The function signature, reproduced from the R documentation, is given below

```
lm(formula, data, subset, weights, na.action,
  method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
  singular.ok = TRUE, contrasts = NULL, offset, ...)
```

We will focus only on the first argument which is the crux of the call — the reader is referred to the R documentation for more details about the other arguments. The `formula` argument is a symbolic construct of the form

```
response ~ f1 + f2 + f3 + \ldots
```

which essentially says that the `response` variable is a linear combination of the basis functions `f1`, `f2`, `f3`, etc. In the Example 1 below we have used the formula

```
Sales ~ cos(0.55*Month) + Month
```

which says that the sales figure will be a linear combination of the `cosine` of the `0.55*month` and the value of the `month` attribute itself. One could use most functions of the explanatory attributes of the objects involved in, as basis functions in the formula. For instance `I(x^2)` would denote a quadratic basis function in the explanatory variable x. The `lm` function returns the model object that can be examined to get the coefficients of the linear model and measures of how good the fit is and/or plotted. The argument `data` denotes a data frame containing the regression data.

Please answer Quiz-2 (11.2) to test your understanding of what we have covered till now, before you proceed.

Some things are now easy to see:

1. The Linear Regression discussed in Predictive Analytics-I is just a special case of regression in this more general setting, where the basis functions are just the individual coordinates (attributes).
2. This potentially lets us fit a very wide class of functions to data and we are no longer constrained by straight lines alone.
3. We can pick any set of linearly-dependent basis functions and 'fit' them to the data — of course how good the fit is will depend on the choice of the basis functions. Choice of the basis functions is not really part of regression algorithms (at least the parametric ones) — these have to be chosen by the user based on past-experience with similar data, exploratory data analysis, domain knowledge and intelligent guesswork.

This should clarify many of the apparently contradictory statements we made about regression in the preceding paragraphs and quizzes. Thankfully much of the elegance of the solution one obtains for linear regression carries almost as is to this generic framework for regression as well, as we will see a little later. We will again have a nice formula that will tell us the values of the best-fit model parameters, corresponding to the best-fit function from within the chosen family of functions, for the given training data. Also the assumptions required for regression to work remain more-or-less unchanged when we move from the simple linear regression to this more generic regression formulation.

Strictly speaking what we are exploiting in the solution to this more generic formulation, just as we did for linear regression, is the linearity in terms of the parameters. Here's something good to remember.

'Linearity' in Linear Regression

The word 'linear' in the term linear regression in fact refers to the linearity in the model parameters (coefficients) and not linearity in the attribute values. Henceforth in this document, the term 'linear regression' would refer to this more general formulation of regression.

Let's revisit our three running examples and examine them in the light of the discussion above on basis functions.

Example 1: Sales Figures Regression

A few observations about the data:

- Clearly there is a periodicity to the sales figures, suggesting some trigonometric function.
- The sales figures (the peaks, troughs and the average values in a year) seem to be also increasing steadily with the months. So we try a linear function of the number of months, as the additional basis function.
- The value goes up for small values of the m variable. This suggests a cos(.) function instead of a sin(.).
- The argument to the cos(.) function determines the frequency of the cosine-wave. We therefore need a 'dampening' factor (here we use 0.5) with the months as the argument to the cos(.) function.

The family of functions of m (the number of months from Jan 1965) we will consider is therefore:

$$S(m) = c_0 + c_1 * cos(0.5 * m) + c_2 * m$$

The fit obtained with this is shown in Figure 7. The R code used to generate this plot is given below. Linear regression on this gives the optimum values for the coefficients $c_0 = 78.64, c_1 = 126.02, c_2 = 5.77$ resulting in the model:

$$S(m) = 78.64 + 126.02 * cos(0.5 * m) + 5.77 * m$$

This seems to capture the trend but is clearly very inaccurate for larger values of m. This is because we have not accounted for the fact that the 'amplitude' of the periodic part of the dataset is also increasing with the months. Trying to take care of this would make our model non-linear. We defer the refinement to account for this to a later section.

```
mymodel <- lm(Sales ~ cos(0.55*Month) + Month, data=sdata)
with(sdata,
     lines(seq(1, 77, by=1),
                predict(mymodel, data.frame(Month=seq(1, 77, by=1))),
                col='red', lwd=2
        )
)
```

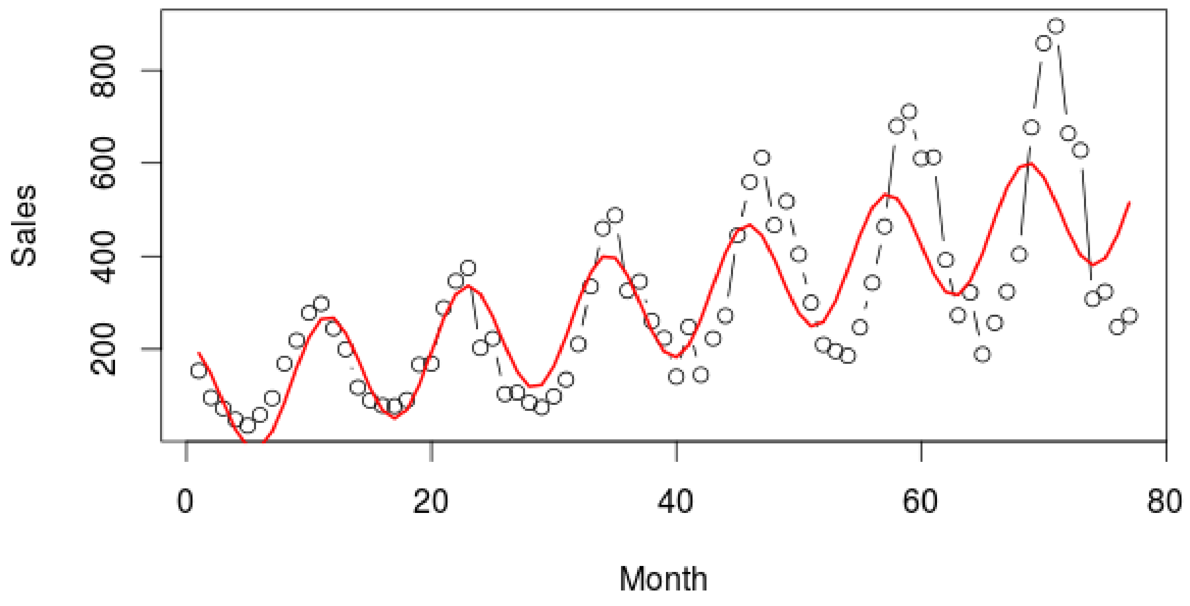

Figure 7: Linear Regression on Sales Figures

Example 2: Total Electricity Consumption in the U.S

The data clearly doesn't look linear — the electricity consumption evidently is not a linear function of the year. It also seems to make domain sense, because over the years the consumption would have grown at an accelerated pace. So non-linearity is only to be expected. Also the shape of the trend suggests a polynomial, probably quadratic, increase in the consumption with the year. So here is a trial code to see a quadratic does well.

```
cdata <- read.csv("total-electricity-consumption-us.csv")
plot(cdata, xlab='Year', ylab='Consumption', type='b')
mymodel <- lm(Consumption ~ Year + I(Year^2), data=cdata)
with(cdata,
        lines(seq(1920, 1970, by=1),
                predict(mymodel, data.frame(Year=seq(1920, 1970, by=1))),
                col='red', lwd=2
        )
)
```

The fit obtained with this model is shown in Figure 8. Contrast this with the quality of the fit (Figure 2) obtained with a cubic model (see the Example 2 box in the Introduction section). We will see how we can choose between these two candidate models, later in this module.

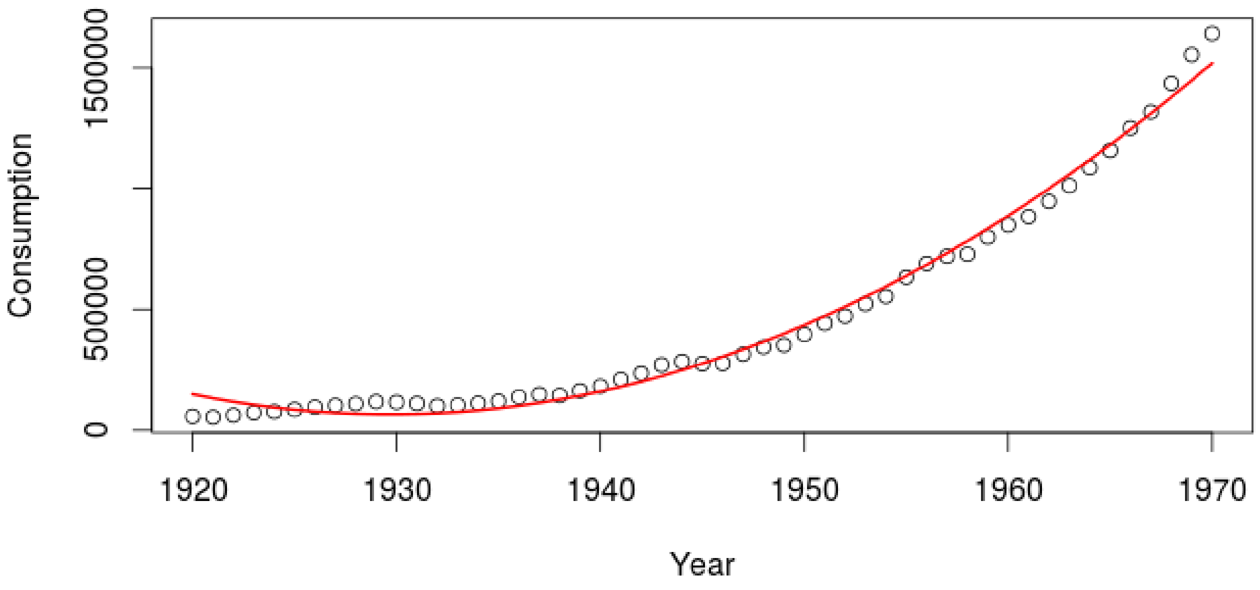

Figure 8: Regression on US Electricity Consumption Data using a Quadratic Model

5 Regularization

A predictive model has to be as simple as possible, but no simpler. Often referred to as the **Occam's Razor**, this is not just a convenience but a fundamental tenet of all of machine learning. To understand why this is the case, here's a short explanation. Let's first get some intuitive understanding of what it means for a model to be 'simple'. To measure the simplicity we often use its complementary notion — that of the complexity of a model. More complex the model, less simple it is. There is no universal definition for the complexity of a model used in machine learning. However here are a few typical ways of looking the complexity of a model.

1. Number of parameters required to specify the model completely. If you consider a parametrized family of functions to choose your model from, then clearly more the number of basis functions used to construct the model, higher the complexity of the model.

2. The *degree* of the function, if it is a *polynomial*.

3. Size of the best-possible representation of the model. For instance the number of bits in a binary encoding of the model. For instance more complex (messy, too many bits of precision, large numbers, etc.) the coefficients in the model, more complex it is. For example the expression $(0.552984567 * x^2 + 932.4710001276)$ could be considered to be more 'complex' than say $(2x + 3x^2 + 1)$, though the latter has more terms in it.

There is a deep relationship between the complexity of a model and its usefulness in a learning context. We elaborate on this relationship below.

1. **Simpler models are usually more 'generic'** and are more widely applicable (are generalizable).
    - One who understands a few basic principles of a subject (simple model) well, is better equipped to solve any new unfamiliar problem than someone who has memorized an entire 'guidebook' with a number of solved examples (complex model). The latter student may be able to solve any problem extremely quickly as long as it looks similar to one of the solved problems in the guidebook. However given a new unfamiliar problem that doesn't fall neatly into any of the 'templates' in the guidebook, the second student would be hard pressed to solve it than the one who understands the basic concepts well and is able to work his/her way up from first principles.
    - A model that is able to accurately 'predict

Generalization Bounds

Statistical Learning Theory does establish bounds of the following kind for many of the models we will encounter, including regression.

$$Pr\left(|E_T(M) - E_G(M)| > \delta\right) \leq e^{-f\left(n, \frac{1}{\mathscr{C}(M)}\right)}$$

Such bounds are known as *PAC* (**Probably Approximately Correct**) bounds. The basic idea is that data available for training is always finite while the domain on which the learnt model needs to be applied eventually is potentially infinite. So how does one know if what we have learnt from the finite set of training examples is good enough to be applicable in general across the domain from which the examples have been drawn? This question is not unlike any typical survey that involves the entire population (health surveys, election poll, etc.) — the survey samples just a small set of people but your predictions have to be for the entire population. A good model is one whose behaviour when applied to the training data is a reasonably accurate indicator of the expected behaviour of the model when used for prediction tasks outside the training set.

Here's a short explanation of the PAC bound that we stated above. The model M produced by the learning algorithm obviously depends on the training set given. Some training sets will result in a good model and others may be not so good. The general spirit of the PAC statement is that no matter what the training set is (remember the training set is really not in your control — you invariably have to work with what is available), under certain mild assumptions, the probability that the behaviour of the model on the training data is very different from the behaviour of the model when applied to the entire domain, is rather small. The 'behaviour' of the model is quantified by the *Empirical* or Training error $E_T(M)$ committed by the model on the training data, and the expected error $E_G(M)$ when it is applied across the domain. The statement above is a claim that the probability that $E_T(M)$ (what is observed) differs from

$E_G(M)$ (what can be expected from the model used for predictions outside the training set) by more than an error margin of δ is upper bounded by a small quantity. The bound on the right hand side is the inverse exponential of a function $f(.,.)$ that increases with size of the training sample n and the inverse of the complexity of the model $\mathscr{C}(M)$. So for achieving the same level of confidence (probability bound) one would require a higher number of training samples for more complex models. Conversely given a training set, the generalizability of the model is much more for simpler models than it is for the more complex ones. Note that the probability in the bound is taken over all possible training datasets of size n.

These bounds form the theoretical basis for the usefulness of many of the well known classical learning algorithms such as Logistic Regression, Linear Regression and SVM.

2. **Simpler models require fewer training samples** for effective training than the more complex ones and are consequently easier to train. In machine learning jargon, the **sample complexity** is lower for simpler models.

3. **Simpler models are more robust** — they are not as sensitive to the specifics of the training data set as their more complex counterparts are. Clearly we are learning a 'concept' using a model and not really the training data itself. So ideally the model must be immune to the specifics of the training data provided and rather somehow pick out the essential characteristics of the phenomenon that is invariant across any training data set for the problem. So it is generally better for a model to be not too sensitive to the specifics of the data set on which it has been trained. Complex models tend to change wildly with changes in the training data set. Again using the machine learning jargon **simple models have low variance, high bias and complex models have low bias, high variance**. Here 'variance' refers to the variance in the model and 'bias' is the deviation from the expected, ideal behaviour. This phenomenon is often referred to as the **bias-variance tradeoff**.

4. **Simpler models make more errors in the training set** — that's the price one pays for greater predictability. **Complex models lead to overfitting** — they work very well for the training samples, fail miserably when applied to other test samples.

Regularization is the process used in machine learning to deliberately simplify models. Through regularization the algorithm designer tries to strike the delicate balance between keeping the model simple yet not making it too näive to be of any use. For the data analyst, this translates into the question "How much error am I willing to tolerate during training, to gain generalizability?"

The simplification can happen at several levels — choice of simpler basis functions, keeping the number of basis functions chosen small, if the family is that of polynomials then keeping the degree of the polynomial low, etc. These are done by the designer and is typically not referred to as regularization. Regularization is the simplification done by the training algorithm (in this case regression) to control the model complexity. The most common forms of regularization for regression involve adding a regularization term to the cost that adds up the absolute values or the squares of the parameters of the model.

Time to test your understanding with another short quiz. Please work on Quiz-3 (11.3) before you proceed.

6 Generalized Robust Linear Regression

We are now ready to put all the above together to finally get to the not-so-complex mathematics and consequently the implementation of generalized linear regression with regularization. Though you will rarely require to implement all of this yourself, it certainly helps to peep a little under the hood to get a sense of how these things work.

As we discussed earlier, we start with a family of functions which are linear combinations of a set of basis functions that we have chosen. Assume the family is of the form

$$\mathscr{F}(\boldsymbol{\alpha} \mid \boldsymbol{x}) = \sum_{i=0}^{k} \alpha_i \phi_i(\boldsymbol{x})$$

where α_i are the coefficients and the functions $\phi_i(.)$ are the basis functions. We of course take $\phi_0(,)$ to be the constant 'function' 1 to accommodate the constant term in our regression function. The vector $\boldsymbol{x}$ denotes the vector of all the explanatory variables. Also on the left-hand side we have denoted the set of α_i coefficients as the vector $\boldsymbol{\alpha}$. A more convenient form of the expression above is to write it in the vector form as

$$\mathscr{F}(\boldsymbol{\alpha} \mid \boldsymbol{x}) = \langle \boldsymbol{\alpha}, \boldsymbol{\phi}(\boldsymbol{x}) \rangle$$

where $\langle ., . \rangle$ denotes the usual dot-product between two vectors (sum of the products of the corresponding elements of the two vectors) and $\boldsymbol{\phi}(\boldsymbol{x})$ is the vector $(\phi_0(\boldsymbol{x}), \phi_1(\boldsymbol{x}), \ldots, \phi_k(\boldsymbol{x}))$.

Suppose we have the training dataset consisting of n-datapoints $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ where $\boldsymbol{x}_i$ denotes the vector of explanatory variables of the i^{th} data point and y_i is the response variable. Suppose we had a model from the above family of functions for some specific values for the coefficients α_i. Then the y-value predicted by the model for the i^{th} data point would have been

$$y'_i = \langle \boldsymbol{\alpha}, \boldsymbol{\phi}(\boldsymbol{x_i}) \rangle$$

However the actual y value given in y_i. We therefore try to find the values of the coefficients $\boldsymbol{\alpha}$ that will minimize the aggregated 'gap' between the actual value and the predicted value, across all the training data points. We can therefore think of the regression problem as that of finding $\boldsymbol{\alpha}^*$ for which the error $E_{\boldsymbol{\phi}}(\boldsymbol{\alpha})$ is a minimum.

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} E_{\boldsymbol{\phi}}(\boldsymbol{\alpha}) = \arg\min_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{n} ||y_i - y'_i|| \right) = \arg\min_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{n} ||y_i - \langle \boldsymbol{\alpha}, \boldsymbol{\phi}(\boldsymbol{x_i}) \rangle|| \right)$$

where arg min refers to the value of $\boldsymbol{\alpha}$ for which the expression inside the big-braces attains the smallest value and $||y_i - y'_i||$ denotes some measure (norm) of the gap between y_i and y'_i. Leaving it at this has no in-built defense against overfitting — the solution to the above optimization problem could result in an $\boldsymbol{\alpha}$ that is way too complicated. Refer back to the discussion in Section 5 on regularization. We therefore need a way to 'penalize' solutions that have unwieldy values for the coefficients and look for a solution that makes the error term as small as possible while trying to keep the model simple. We do this by adding a regularization term to the error. The final regularized regression problem is now formalized as:

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{n} ||y_i - \langle \boldsymbol{\alpha}, \boldsymbol{\phi}(\boldsymbol{x_i}) \rangle|| + \lambda.||\boldsymbol{\alpha}|| \right)$$

where $||\boldsymbol{\alpha}||$ refers to some measure of the complexity of the coefficient vector $\boldsymbol{\alpha}$ itself and λ is the regularization coefficient. Including this in the quantity (called the *cost/loss function*) to be minimized, discourages very complex $\boldsymbol{\alpha}$s as solutions. Just to get the interpretation of this correct, note that we see the expression above as the cost/loss incurred while trying to fit a model to the data. The cost incurred will increase whenever the error (gap between the observed and predicted values) increases or the model complexity increases. Our objective is keep the cost as low as possible. Some observations about the regularization coefficient λ.

Regularization Coefficient

The regularization coefficient λ is a constant whose value is chosen upfront, before carrying out the regression. The choice of an appropriate value for λ can be important in practice. Though one would largely arrive at the correct value of λ by some amout of trial and error, a few thumbs rules can be good to remember.

- Choosing 'too large' a value for λ can result in unrealistically 'simple' models. Note that in this case, any small increase in the complexity of the $\boldsymbol{\alpha}$ vector gets penalized very heavily.
- Choosing a very small value for λ can result in overfitting — there is hardly any disincentive for increased complexity of the solution and hence the regression is likely to result in something that is way too complex than it needs to be.

Regularization Coefficient and Constrained Optimization

The regularization part of the formulation can be viewed as a constrained optimization problem where we wish to put a cap on the complexity of the model coefficients. The regression problem can now be seen as that of minimizing the error subject to the cap c on the model complexity. Formally

$$\min_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{n} ||y_i - \langle \boldsymbol{\alpha}, \boldsymbol{\phi}(\boldsymbol{x_i}) \rangle|| \right)$$
$$\text{Subject to}: \ ||\boldsymbol{\alpha}|| \leq c$$

Standard (Lagrangian) optimization theory tells us that the equivalent optimization problem is

With this general framework, we can now 'derive' multiple versions of linear regression for various choices of the norms $||y_i - \langle \boldsymbol{\alpha}, \boldsymbol{\phi}(\boldsymbol{x_i}) \rangle||$ used to quantify the gap between the actual value and the value predicted by the model, along with the complexity norm for the coefficient vector $||\boldsymbol{\alpha}||$. We describe some of the popular variants of linear regression in the subsections below.

Training vs Deployment

There is a subtle difference in the way the coefficients ($\boldsymbol{\alpha}$) of the model and the response 'variable' y are interpreted during training and during deployment (when the 'learnt' model is being used for prediction). During training the response values y_i are known (given) and are hence treated as constants and we need to determine values for the coefficients towards minimizing the cost. The coefficient values found would characterize the best-fit model. So the $\boldsymbol{\alpha}$s are being treated as unknown 'variables' whose values are to be determined during training. On the other hand when the model is being used (deployment) to predict the response (variable) y for a given explanatory attribute vector $\boldsymbol{x}$, the coefficients α_is are treated as constants (learnt/determined during training).

6.1 Ridge Regression

In ridge regression (this is the 'standard' *least squares regression*) we choose the square norms.

$$||y_i - \langle \boldsymbol{\alpha}, \boldsymbol{\phi}(\boldsymbol{x_i})\rangle|| = \left(y_i - \langle \boldsymbol{\alpha}, \boldsymbol{\phi}(\boldsymbol{x_i})\rangle\right)^2, \qquad ||\boldsymbol{\alpha}|| = \sum_{i=0}^{k} \alpha_i^2$$

Note that in this case $||\boldsymbol{\alpha}||$ is just (Euclidean) length of the vector $\boldsymbol{\alpha}$. This is the most common form for linear regression, for several reasons. Firstly this form leads to a clean closed form solution (explicit formula) for the best-fit coefficient vector $\boldsymbol{\alpha}$. Let Φ denote the *matrix* of *order* $n \times (k+1)$ formed from the basis functions $\phi_i(.)$ and the data point attribute vectors $\boldsymbol{x}_i$ as follows:

$$\Phi \equiv \begin{bmatrix} \phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & \dots & \phi_k(\boldsymbol{x}_1) \\ \phi_0(\boldsymbol{x}_2) & \phi_1(\boldsymbol{x}_2) & \dots & \phi_k(\boldsymbol{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\boldsymbol{x}_n) & \phi_1(\boldsymbol{x}_n) & \dots & \phi_k(\boldsymbol{x}_n) \end{bmatrix}$$

Also let $\boldsymbol{y}$ denote the vector $(y_1, \dots, y_n)$. Then we directly write a formula for the coefficients that form the best fit.

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{n} \left(y_i - \langle \boldsymbol{\alpha}, \boldsymbol{\phi}(\boldsymbol{x_i})\rangle\right)^2 + \sum_{i=0}^{k} \alpha_i^2 \right) = \left(\Phi^T \Phi + \lambda I\right)^{-1} \Phi^T . \boldsymbol{y}$$

where I denotes the *identity matrix* of order $(k+1) \times (k+1)$, the superscript $.^{-1}$ denotes the inverse a matrix and the superscript $.^T$ denotes the transpose of a matrix. Readers interested in the proof (derivation) of this solution are referred to Section 12 devoted exclusively for the supporting mathematical proofs of statements made in this document. Proof of the closed form solution to the ridge regression formulation is given in Theorem 1, in the Proofs section. Practical implementations employ several tricks to ensure that computation using this formula is numerically stable, the matrix $\left(\Phi^T \Phi + \lambda I\right)$ is invertible, etc. The standard R function `lm` essentially is a ridge regression implementation. The reader is referred to more advanced material (such as the expositions — Chapter 7 — in well known texts like [2]) for more details.

Gaussian Interpretation of Least Squares Regression

There is an interesting probabilistic interpretation (justification ?) of the least squares formulation we saw here. The idea is to imagine that the data points have been generated by a process that has an 'ideal' behaviour characterized by a model from the chosen family of functions. So given the other explanatory attributes $\boldsymbol{x}$ of any data point, the ideal value of the response variable y' ought to have been $y' = f(\boldsymbol{x})$ where $f(.)$ is the function from the family that characterizes the behaviour of the underlying system. Note that the function $f(.)$ is unknown, to be estimated by the regression algorithm. We only hypothesise the *existence* of such a function. However we assume that due to noise (unpredictable system behaviour, measurement errors, etc.) the actual value observed y for the response is different from y'. We model this scenario by assuming that y follows a *Gaussian distribution* with y' as the mean — y will often be pretty close to y' and rarely deviates too far from y'. We now ask the question: 'which model best explains the behaviour of the system?'. The equivalent mathematical criterion is 'for which model is the probability of seeing the dataset that was observed, the most likely?'. The probability of seeing the data point $(\boldsymbol{x}_i, y_i)$ is — the y-value is supposed to have been $f(\boldsymbol{x}_i)$ (the expected value), however it is observed as y_i due to (Gaussian) noise. The probability of seeing y_i (given the attributes $\boldsymbol{x}_i$) in a Gaussian distribution with mean $f(\boldsymbol{x}_i)$ is

$$Pr(Y = y_i \mid \boldsymbol{x}_i, f) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\left(\frac{y_i - f(\boldsymbol{x}_i)}{\sigma}\right)^2}$$

Since we have assumed all the data points are independent of each other, the joint probability of seeing all of them is just the product of the individual probabilities. Hence the probability of seeing the entire data set (denoting this as D) is

$$Pr(D|f) = \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^n e^{-\left(\frac{\sum_{i=1}^n (y_i - f(\boldsymbol{x}_i))^2}{\sigma^2}\right)}$$

where σ^2 is the variance of the Gaussian. Maximizing this probability is the same as minimizing the exponent, since it is negative. So

$$\max_f Pr(D \mid f) = \min_f \sum_{i=1}^n (y_i - f(\boldsymbol{x}_i))^2$$

which is what we set out to do, other than the regularization. We are assuming here that the variance of the noise is a constant, identical for all the data points — in other words all data points are influenced by 'similar' noise that is independent of the explanatory attribute vector $\boldsymbol{x}$. It is possible to 'derive' the regularization term through a Bayesian formulation with an appropriate *prior*. A detailed explanation of this more advanced formulation is however beyond the scope of this course.

6.2 Lasso Regression

Lasso regression uses the same square error as Ridge regression but the regularization term is different — Lasso uses the sum of the absolute values of the coefficients as the regularization term, instead of the sum of squares.

$$||y_i - \langle \boldsymbol{\alpha}, \boldsymbol{\phi}(\boldsymbol{x_i})\rangle|| = \left(y_i - \langle \boldsymbol{\alpha}, \boldsymbol{\phi}(\boldsymbol{x_i})\rangle\right)^2, \qquad ||\boldsymbol{\alpha}|| = \sum_{i=0}^k |\alpha_i|$$

This rather simple (and ad-hoc) looking change actually has a profound effect on the kind of solutions that emerge. Refer Figure 9 for the discussion below. Recall that the regression cost function that we discussed earlier has two components — error and regularization. Figure 9 is a schematic illustration of both the error (red lines) and regularization (green dotted for ridge regression and black lines for lasso) contours. Recall that a contour of a function $f(\boldsymbol{\alpha})$ is a trace (locus) of the points that satisfy the equation $f(\boldsymbol{\alpha}) = c$ for some constant c. Also note that during training the coefficients α_i are treated as variables — so the contours are plots of the error / regularization terms as functions of $\boldsymbol{\alpha}$. Figure 9 shows contours for various values of c — the 'inner' contours represent lower values of the error/regularization terms. The key observation here is that at the optimum solution for $\boldsymbol{\alpha}$ (the place where the sum of the two terms is a minimum), the corresponding regularization contour and the error contour must 'touch' each other tangentially and not 'cross'. The blow-up of one of the crossings between an error contour and a regularization contour in the Figure illustrates this — one could move along the arrow shown to keep the error term same and reduce the regularization term, giving a better solution (the sum of the terms reduces). The '8-Point stars' highlight the touch points between the error contours and the lasso regularization contours. The '5-point stars' highlight the touch points between the error contours and the ridge regularization terms. The picture illustrates the fact that because of the 'corners' in the lasso contours, the touch points are more likely to be on one or more of the axes. This implies that the other coefficients become zero.

Variable Shrinkage vs Selection

The regularization term results in *variable shrinkage* — makes some of the attributes or basis functions more/less important than some others. The bases with large coefficient values are more influential in determining the value of the response variable and hence are more important. This is a characteristic common to both ridge and lasso regression. However lasso is more likely to make the coefficients of the irrelevant attributes otr basis functions zero. Therefore lasso also eliminates the irrelevant basis functions (it *selects* only the relevant ones) and attributes effectively simplifying the model significantly.

However the price we pay for the advantage we get in lasso is that the use of absolute values in the regularization terms makes it a rather non-trivial optimization problem and does not lead to a neat closed form solution that we obtained in ridge regression.

6.3 Robust Regression

Please carry out the experiment suggested in Exercise 11.4.1 before you proceed.

You must have observed from the experiment that ridge regression has a tendency to somehow 'accommodate' all the data points given. Since it tries to minimize the sum of square errors, a point that is far way from what should have been the trend line (such points are referred to as *outliers*) has a lot of influence on the final model. Consider this simple example. Suppose there are 100 values of which 50 are -0.5, 49 are 0.5 and one is 100.0. Note that the last value of 100.0 is clearly an outlier and we would rather ignore this as an aberration. All the other points seem to be centered around 0.0 and hence that should be considered the mean value, that captures the trend of these data points. However if we carry out a least squares fit — i.e., find a mean value say $a > 0$ such that the sum of the square errors given by

$$50 * (a + 0.5)^2 + 49 * (a - 0.5)^2 + (100 - a)^2 = 100a^2 - 199a + 100^2 + 99 * 0.5^2$$

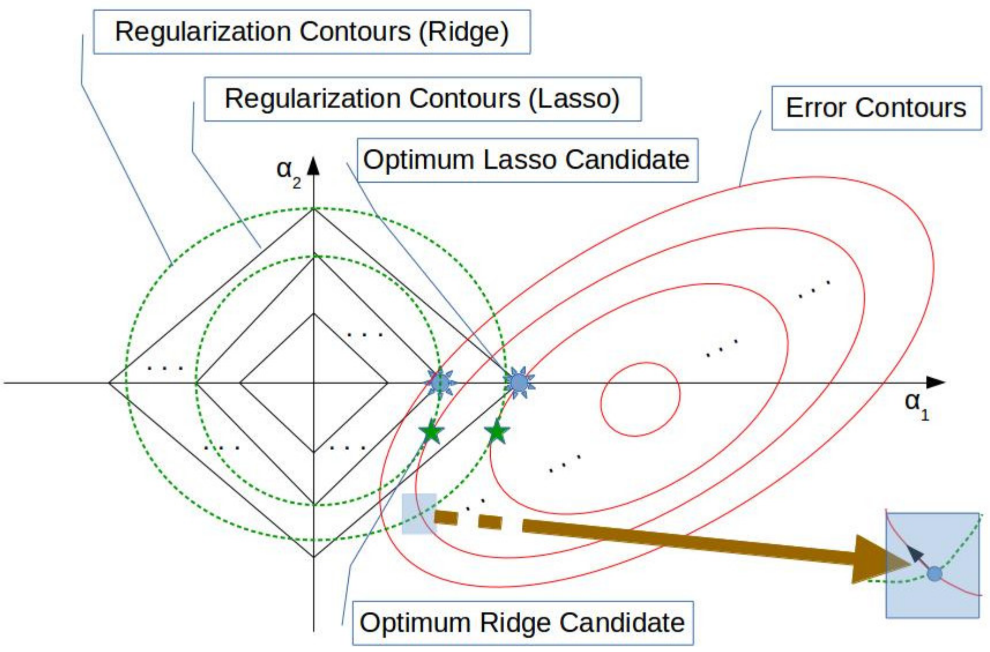

Figure 9: Lasso vs Ridge Regression

is the least. The minimum value for the above expression occurs at $a = 0.995$. So the least squares estimate will not be 0.0 — what has happened is that the outlier has influenced the solution to pull the mean towards itself. The culprit in this case seems to be the fact that we are squaring the errors which causes the contribution from the points that are farther away from the mean to get 'blown up'. Robust regression tries to address this issue by taking the absolute value of the difference between the model prediction and the actual value rather than the square, as the error term. The robust regression solution is therefore

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \sum_{i=1}^{n} |y_i - \langle \boldsymbol{\alpha}, \boldsymbol{\phi}(\boldsymbol{x_i}) \rangle|$$

The

Choice between Ridge, Lasso and Robust Regression

We give a few thumb-rules that will help you decide which regression method is the most appropriate for a given regression problem.

- If you suspect that one or more of the explanatory attributes in the dataset are redundant (they do not have an influence on the value fo the response variable) the Lasso might be the option — it can eliminate the redundant attributes with some variable selection.
- If you suspect there could be extreme outliers in the data which are hard to detect

otherwise, you might want to use robust regression.

•

7 Non-Linear Regression

Linear regression, in the more generic form presented here, is an extremely powerful tool. However there are situations where linear regression is not adequate. Linear regression rests on a few assumptions about the data — collectively called the *Gauss-Markov Conditions*. We state the converse of these conditions here: any of the following is a violation of the Gauss-Markov conditions and in such cases linear regression is unlikely to be an adequately accurate model.

Conditions

Any of the following conditions typically make it essential to have a non-linear model for the regression.

- Errors (noise) in data points are correlated with each other. Imagine data coming from say a faulty equipment that gets 'stuck'. So if the previous data point had a lot of noise, it is possible that the noise will (because the equipment got stuck at the new position) be even greater for the next data point. Linear Regression assumes that the errors are all independent of each other — that the error in one data point does not influence the error in the subsequent data points.
- The errors are dependent on the explanatory variables. Imagine an audio amplifier and suppose we are trying to fit a regression model that will predict the output volume v in dB from the setting s (on a 1–10 scale) of the volume knob. Most so the regression model we are targeting is of the form $v = f(s)$. The actual volume observed however has a noisy relationship with the knob setting. It is a common characteristic of most amplifiers that the noise levels will increase with increase volume. So the actual volume observed could be of the form $v = f(s) + \epsilon(s)$ where the random noise term $\epsilon(s)$ is itself a function of amplifier setting s. One common way to model such errors is as a Gaussian whose variance is a function of s. This is unlike the assumption we made in linear regression that the error Gaussian has a constant variance for all the data points. See the box on the Gaussian interpretation of least squares above.

Assuming this

8 Generalized Linear Models

9 Kernel Regression

This is a non-parametrized approach to regression and is in general enough to be able to model a rather wide class of curves. This section will also serve as a natural bridge to the next module on *Kernel Methods*. The idea is to have a

10 Evaluation and Choice Between Regression Solutions

11 Exercises

11.1 Quiz-1

Exercise 11.1.1 *Regression is a statistical technique to:*

a. Fit a straight-line through a set of points.

b. Arrive at an empirical formula (from data) that best captures how one of the attributes of a certain class of entities depends on its other attributes.

c. Construct a smooth curve that passes through a set of points.

d. Predict the outcome given a set of input conditions.

***Answer**: **(b)**. This is a broad definition for regression. The standard linear regression roughly corresponds to option (a).* ■

Exercise 11.1.2 *Which of these are necessary assumptions for the normal linear regression algorithm to work?*

a. The entities are all independently drawn from a common distribution. ***Yes***

b. The attributes are all independent of each other. ***No***

c. The response value being predicted is assumed to follow a Gaussian Distribution. ***Yes***

d. The independent variables used for the regression need to follow a Gaussian Distribution. ***No***

e. More noise at larger values of the independent variables is acceptable. ***No***

Exercise 11.1.3 *Answer Yes or No for the following.*

a. Regression cannot be done on noisy data. ***No***

b. In principle it is possible to 'fit' any class of curves (circles, straight lines, sine curves,...) to any data set. ***Yes***

c. Over-fitting cannot happen in regression. ***No***

d. Regression cannot be done on categorical attributes. ***Yes***

e. *The outcome of regression training is a predicted value for an attribute whose value is unknown.* ***No***

f. *Which among these are NOT likely candidates for a regression fit:*

   (a) *Short term trends in stock price of a specific scrip* ***Yes***

   (b) *Commodity Prices based on factors such as agricultural production, inflation rate, subsidy index, rainfall etc.* ***Yes***

   (c) *Fraud Detection* ***No***

   (d) *Sales forecast* ***Yes***

   (e) *Determining the risk factor in a given insurance claim.* ***No***

11.2 Quiz-2

Exercise 11.2.1 *Which of the following are examples of function contours.*

a. *National boundaries on a political map.* ***No***

b. *Heat maps from a infrared camera.* ***Yes***

c. *Isobars (altitude), Isotherms on a geographical map.* ***Yes***

d. *Road network on Google maps.* ***No***

e. *Densities of where http requests are originating from for a website shown on Google Maps.* ***Yes***

Exercise 11.2.2 *How many parameters would you require to describe the family of all quadratic functions in three variables (3-dimensions)? Can you generalize this to a general formula for d dimensions and a k^{th} degree polynomial? The latter is meant to be more of an interesting mathematical exercise.*

Exercise 11.2.3 *Can you think of a function of a single positive variable x that grows monotonically with x, but grows much slower than x^a for any real number $a > 0$?*

Exercise 11.2.4 *Describe the family of all ellipses on the plane (2 variables)? Note that this may not necessarily be linear in the parameters. How may parameters does this family require?*

Exercise 11.2.5 *Carry out this data experiment to get a feel for the fact that regression is essentially meant to deal with noisy data.*

a. *Assume a linear function in one variable — any $f(x)$ of the form $ax+b$ for some constants a, b.*

b. *Generate the dataset $D = ((x_1, z_1), \ldots, (x_n, z_n))$ for some n, where $\forall 1 \leq i \leq n,\ \ z_i = f(x_i)$ — pick some arbitrary values for the x_is and compute the corresponding z_is.*

c. *Add a noise to each data point that is generated. Assume a small variance (some number say in the range* $0.5\ldots3$*) and use a Gaussian random number generator to generate noise values* ϵ_i *for* $1 \le i \le n$*. Note that you can use the function* `rnorm(1)` *in R to generate a Gaussian random number with variance* 1.

d. *Generate a new (noisy) dataset* $D' = ((x_1, y_1), \ldots, (x_n, y_n))$ *where* $y_i = z_i + \epsilon_i$.

e. *Run linear regression on* D' *and compare it with the original function* $f(x)$ *from which the data was generated.*

f. *Try the above with different values of* n *and verify that you are getting closer to the original function* $f(x)$ *with larger values of* n.

g. *Also examine the effect of the noise variance on the number of training samples required to get a good fit.*

h. *What happens when* $n = 2$*?*

11.3 Quiz-3

Exercise 11.3.1 *Carry out this data experiment to get a feel for why more-often-than-not you are better off with simpler models than more complex ones.*

a. *Assume a cubic function in one variable — any* $f(x)$ *of the form* $a + bx + cx^2 + dx^3$ *for some constants* a, b, c, d.

b. *Generate the dataset* $D = ((x_1, z_1), \ldots, (x_n, z_n))$ *for some* n*, where* $\forall 1 \le i \le n,\ \ z_i = f(x_i)$.

c. *Add a noise to each data point that is generated. Assume a small variance (some number say in the range* $0.5 - -3$*) and use a Gaussian random number generator to generate noise values* ϵ_i *for* $1 \le i \le n$.

d. *Generate a new (noisy) dataset* $D' = ((x_1, y_1), \ldots, (x_n, y_n))$ *where* $y_i = z_i + \epsilon_i$.

e. *Run two linear regressions on* D' *— one with a linear model and another with a cubic model. Plot the two and compare the quality of the fit.*

f. *Since the data was generated from a cubic, it seems natural to expect that the cubic model should give a better fit than the linear model. Is that what you observe as result of this experiment?*

g. *Try the above with different values of* n *and verify that as* n *becomes large the cubic will tend to do better than the linear model. For how large a* n *did the cubic model actually outdo the linear model?*

11.4 Quiz-4

Exercise 11.4.1 *Carry out this data experiment to understand the effect of outliers on the ridge regression solution.*

a. *Assume a linear function in one variable — any $f(x)$ of the form $ax+b$ for some constants a,b.*

b. *Generate the dataset $D = ((x_1,z_1),\dots,(x_n,z_n))$ for some n, where $\forall 1 \le i \le n,\ \ z_i = f(x_i)$.*

c. *Add a noise to each data point that is generated. Assume a very small variance (some number say 0.1) and use a Gaussian random number generator to generate noise values ϵ_i for $1 \le i \le n$.*

d. *Generate a new (noisy) dataset $D' = ((x_1,y_1),\dots,(x_n,y_n))$ where $y_i = z_i + \epsilon_i$.*

e. *Add another 'outlier' data point with some arbitrary x_0 and $y_0 = f(x_0)+\delta$ for some large δ (say $\delta = 100$), to the dataset D.*

f. *Run linear regression on D' and compare it with the original function $f(x)$ from which the data was generated.*

g. *Try the above with different values of n and δ and may be even additional outliers and explore the effect of outliers on the final model.*

12 Proofs

Theorem 1 *Ridge Regression has a closed form solution for the optimum choice of the model parameters.*

Proof: Let Φ denote the data matrix introduced in the section on Ridge Regression and let $\mathscr{C}(\boldsymbol{\alpha})$ denote the cost function (as a function of the model parameters $\boldsymbol{\alpha}$). Recall that Φ is the matrix

$$\Phi \equiv \begin{bmatrix} \phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & \dots & \phi_k(\boldsymbol{x}_1) \\ \phi_0(\boldsymbol{x}_2) & \phi_1(\boldsymbol{x}_2) & \dots & \phi_k(\boldsymbol{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\boldsymbol{x}_n) & \phi_1(\boldsymbol{x}_n) & \dots & \phi_k(\boldsymbol{x}_n) \end{bmatrix}$$

$$\begin{aligned}
\mathscr{C}(\boldsymbol{\alpha}) &= \sum_{i=1}^{n}(y_i - (\Phi.\boldsymbol{\alpha})_i)^2 + \lambda \sum_{i=1}^{k} \alpha_i^2 && \text{(Cost; } (\Phi.\boldsymbol{\alpha})_i \text{ is the } i^{th} \text{ component of } \Phi.\boldsymbol{\alpha}) \\
&= \langle (\boldsymbol{y}-\Phi.\boldsymbol{\alpha}), (\boldsymbol{y}-\Phi.\boldsymbol{\alpha})\rangle + \lambda\langle \boldsymbol{\alpha},\boldsymbol{\alpha}\rangle && \text{(Cost, Vector Form)} \\
&= \langle \boldsymbol{y},\boldsymbol{y}\rangle - 2.\boldsymbol{\alpha}.\Phi.\boldsymbol{y} + \boldsymbol{\alpha}^T\Phi^T\Phi\boldsymbol{\alpha} + \lambda\langle \boldsymbol{\alpha},\boldsymbol{\alpha}\rangle \\
\frac{d\mathscr{C}(\boldsymbol{\alpha})}{d\boldsymbol{\alpha}} &= -2\Phi.\boldsymbol{y} + 2\Phi^T\Phi\boldsymbol{\alpha} + 2\lambda\boldsymbol{\alpha} = \boldsymbol{0} && \text{(To minimize } \mathscr{C}(\boldsymbol{\alpha})\text{, setting the gradient w.r.t } \boldsymbol{\alpha} \text{ to 0)} \\
\boldsymbol{\alpha} &= \left(\Phi^T\Phi + \lambda I\right)^{-1}.\Phi.\boldsymbol{y}
\end{aligned}$$

■

13 Glossary

This section lists all the mathematical terms (jargon) used in this document without explanation. These are effectively the prerequisites that reader is expected to be familiar with. The reader is referred to other sources listed below from where the reader can fill the gaps in the prerequisites if any. This section just lists the terms that have been taken for granted with a very sketchy one-line description of each term without any further explanation. The reader is expected to seek other sources such as the ones listed about for a more elaborate exposition.

function takes a set of *arguments* and produces an output value. The space of input values (for the arguments) is called the *domain* of the function and the space of possible output values is called the *range* of the function. The function $f(x) \equiv x^2$ for instance maps the set of all real numbers to the set of positive real numbers. A function that takes only one argument is called a *univariate* function and a function that takes multiple arguments is called a *multivariate* function.

distribution is an assignment of probability for each possible outcome of a statistical experiment. The probabilities are all non-negative and must add up to 1.

dot-product Also called the inner-product of two *vectors* $\boldsymbol{a} \equiv (a_1, \ldots, a_n)$ and $\boldsymbol{b} \equiv (b_1, \ldots, b_n)$. It is defined as the sum $\sum_{i=1}^{n} a_i.b_i$.

Gaussian/Normal distribution is arguably the most important *distribution* in statistics. It is characterized by the density function

$$Pr(X = x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{\sigma^2}}$$

This gives the probability that a random variable X takes the value x (the outcome). X can take any real value. The Gaussian represents the famous *bell curve* — values around a mean μ are the most probable and the probability of X taking a value that is $\pm c$ from μ drops (exponentially) rapidly as c increases. σ is the standard-deviation of the distribution, characterizing the 'spread' of the bell curve — sharper the bell curve, concentrated around its mean value, smaller the value of σ. σ^2 is called the *variance* of the Gaussian.

identity matrix Identity matrix is a special *square* matrix in which all the diagonal elements (row and column indices are equal) are 1 and all other elements are 0. The identity matrix, usually denoted as I has the nice property that $I.\boldsymbol{x} = \boldsymbol{x}$, $\boldsymbol{y}.I = \boldsymbol{y}$, $I.M = M$, $N.I = N$, where $\boldsymbol{x}, \boldsymbol{y}$ are any vectors and M, N are any matrices, provided these products are well defined.

matrix The most elementary definition of a matrix is that it is a rectangular array of numbers, arranged as rows and columns. A matrix of *order* $n \times m$ is a matrix that has n rows and m columns. The matrix is called a *square matrix* if $m = n$ (number of rows is equal to the number of columns). Elements of a matrix are indexed by an ordered pair (i, j) of indices - the first index i is the row index and the second j is the column. Please refer other sources for details about operations on matrices (transpose, inverse, etc.), operations between *vectors* and matrices, operations between (scalar) constants and matrices, etc.

polynomial (univariate) A (univariate) polynomial is an expression of one variable x that is of the form $a_0 + a_1.x + a_2.x^2 + \ldots + a_k.x^k$ for some fixed constants $a_0, \ldots, a_n, k$. This is an expression involving various powers of the variable x upto a maximum power k. k is the *degree* of the polynomial. A polynomial of degree 1 is called a *linear* function.

polynomial (multivariate) A multivariate polynomial on m variables of degree k is one that is defined as a sum of *terms* each of which is the product of the powers of each of the variables, where the powers add up to not more than k. For example $x+4y^2z^3+5x^2y+3z^4$ is a multivariate polynomial of degree 5 in variables x, y, z.

prior Assumption about the nature of a random variable, typically coming from domain experience, that is made even before the actual data is seen or examined. The priors encode domain expertise and typically their influence in the inferences diminish as more and more real data is made available. Priors are both an essential and possibly the most controversial part of the Bayesian 'school' in statistics.

vector Vector is an ordered collection of values (called the *components* of the vector) which are usually real numbers (actually vectors are much more general mathematical objects in which the components can be from any domain that satisfies some basic properties such as existence of product and sum operations, existence of a zero element and a unit element etc. - so for example one could have vectors of complex numbers, images, etc..). Vector is represented as a tuple of its components $\boldsymbol{a} \equiv (a_1, \ldots, a_n)$ — this is a vector of dimension n and a_is are the components of $\boldsymbol{a}$. One could think of a vector as the 'arrow' from the origin to the point in n-dimensional space whose coordinates are $(a_1, \ldots, a_n)$. We often use the terms point and vector (defined in this sense) interchangeably.

References

[1] R.J. Hyndman. *Time Series Data Library*. `http://data.is/TSDLdemo`. Accessed on July 3, 2016 (cit. on pp. 7–9).

[2] Kevin P. Murphy. *Machine Learning - A Probabilistic Perspective*. Second. MIT Press, 2012 (cit. on p. 24).