Topic 5
Nonparametric Regression Classification and Model selection
(Draft)

Instructor: Farid Alizadeh*

January 30, 2020

1 Doing Nonlinear Regression

Regression and classification are not restricted to models which are linear in input variables. We could also use nonlinear functions as our models, *as long as dependence on parameters are linear*. For instance, linear regression is perfectly suitable for models like:

$$y = b_0 + b_1 x + b_2 x^2 + b_3 x^3$$
$$y = b_0 + b_1 e^x + b_2 e^{2x} + b_3 e^{3x}$$
$$y = b_0 + b_1 \cos(x) + b_2 \cos(2x) + b_3 \cos(3x)$$
$$y = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_1 x_2 + b_4 x_1^2 + b_5 x_2^2 + b_6 x_1^2 x2 + b_7 x_1 x_2^2$$

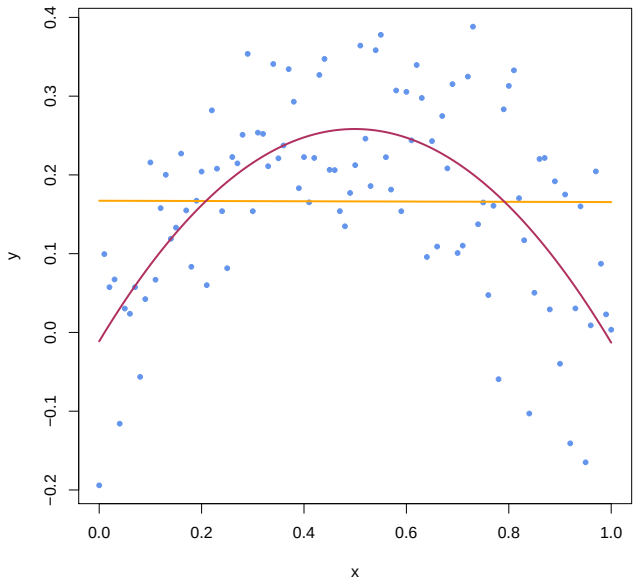
All these models are linearly dependent on the parameters b_i and so they are perfectly fine for linear regression. We only need to compute the extra columns $x^2, x^3, e^x, \cos(x)$, etc. from the data. Once these are created they can be treated as if they are new variables.

Examples of *nonlinear* models include:

$$y = \frac{b_0 + b_1 x}{b_2 + b_3 x}$$
$$y = b_0 + b_1 \cos(b_2 + x) + b_3 \cos(b_4 + x)$$

In these cases some parameters b_i are *nonlinearly* dependent on the model and so applying the maximum likelihood method to them results in nonlinear systems of equations.

Nonlinear formulas are essential and may be a lot more suitable than linear ones. For instance, in the following diagram, clearly there is a relationship between x and y based on the scatter plot shown. But it is not captured by the linear model $y = b_0 + b_1 x$. As you can see the estimated linear function is almost horizontal, indicating $b_1 \approx 0$, and indicating that x is insignificant. However, the model $y = b_0 + b_1 x + b_2 x^2$ fits a lot better.



In fact, here are summaries of linear and quadratic models in R:

```
> summary(linModel)

Call:
lm(formula = y ~ x, data = data.frame(x = x, y = y))

Residuals:
    Min      1Q  Median      3Q     Max
-0.3613 -0.0783  0.0151  0.0777  0.2222

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.16722    0.02492    6.71  1.2e-09 ***
x           -0.00169    0.04306   -0.04     0.97
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
Residual standard error: 0.126 on 99 degrees of freedom
Multiple R-squared:  1.56e-05,  Adjusted R-squared:  -0.0101
F-statistic: 0.00154 on 1 and 99 DF,  p-value: 0.969

> summary(quadModel)

Call:
lm(formula = y ~ x + I(x^2), data = data.frame(x = x, y = y))

Residuals:
     Min       1Q   Median       3Q      Max
-0.23566 -0.06227  0.00003  0.07895  0.18753

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.0112     0.0279   -0.40     0.69
x             1.0798     0.1292    8.36  4.3e-13 ***
I(x^2)       -1.0815     0.1250   -8.65  1.0e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.0955 on 98 degrees of freedom
Multiple R-squared:  0.433,     Adjusted R-squared:  0.422
F-statistic: 37.4 on 2 and 98 DF,  p-value: 8.3e-13

>
```

As can be seen, x is insignificant in the linear model, but both x and x^2 (indicated by `I(x^ 2)`) are very significant in the quadratic model.

2 Extension to Nonparametric Regression

The nonlinear models above are still parametric, since we are imposing a specific formula on the model. What if we have only a data set for x, y and we are asked to fit the best fitting *function* to this data. We already have done this kind of task using the kNN and CART approaches. It turns out that linear regression can also be extended to a nonparametric setting.

The technique we use is along the lines we discussed in the SVM models, that is by basis expansion. Recall that the original set of features x_i is called the *input space*, and the basis expansion is called the *expanded space*.

1. A (possibly) infinite sequence of *basis functions* $\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), \ldots, \varphi_k(\mathbf{x}), \ldots$ are chosen. The requirement is that in the space of all possible functions that the unknown function $f(\cdot)$ could have been drawn from, every func-

tion could be written as a linear combination of (possibly infinite) $\varphi_i(\mathbf{x})$. We will give two examples in the following two subsections.

2. For an unknown function $f(\mathbf{x})$ for which we have only a data set $\mathbf{x}_1, \ldots, \mathbf{x}_N$ and the corresponding values, $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)$, (which, by the way, are only approximations since there are noise in the data), we also need to determine
$$f(\mathbf{x}) \approx b_1\varphi_1(x_1) + \cdots + b_k\varphi_k(\mathbf{x}).$$

So we need to determine two things:

First, the *model complexity* k, that is the number of basis functions to use. The larger k the more flexible our approximation, and the better we can fit our estimated function to the data. However, too large of k, that is too complex of a model, results in overfitting, and a model that may be a poor predictor for new points. In other words, too complex of a model may have poor generalization properties.

Second, the *unknown parameters* b_i which are estimated as usual by least squares method.

The approach is non-parametric since we are not committed to a fixed model. Instead we are considering a *sequence of parametric models* for $k = 1, 2, \ldots$. We then use techniques such as validation set method to try to find the best choice for k. This is similar to the way we chose k in the k nearest neighbor method. More on the sensible choice of k will be given below.

Example: Nonparametric polynomial approximation

Let us consider *polynomials* as a template.

A polynomial of one variable x of degree d is the function:
$$p_d(x) = b_0 + b_1x + b_2x^2 + b_3x^3 + \cdots + b_dx^d$$

So in this case $\varphi_0(x) = 1, \varphi_1(x) = x, \ldots, \varphi_i(x) = x^i, \ldots$ (indexing starts from zero not one to accommodate the constant 1).

First, notice that the larger d is the more flexible a polynomial can be. So in this case model complexity is the degree of the polynomial[1]. For $d = 1$, we have only lines. For $d = 2$, only hyperbolas and lines are possible. For d around 20, polynomials can be flexible enough to mimic most standard functions reasonably closely. So Given a data set and an unknown function $y = f(x)$ we could try to fit polynomials of ever larger degrees until the estimated polynomial fits the data. Actually, there is the following fact from calculus and mathematical analysis that somewhat justifies this approach:

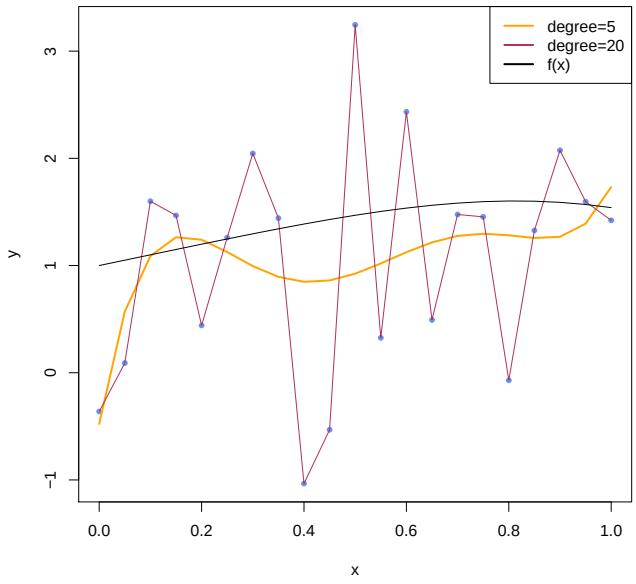
> *Given a continuous function $f(x)$ over an interval $[a, b]$, and given a desired accuracy ϵ, there is an integer d and a polynomial $p(x)$ of degree d such that $p(x)$ approximates $f(x)$ to within ϵ in the interval*

[1] more accurately model complexity is $d+1$, because we have to approximate $d+1$ parameters b_i.

$[a, b]$. *Here, "approximation to within* ϵ*" means that* $|f(x)-p(x)| < \epsilon$ *for every* x *in the range* $[a, b]$.

However, this fact alone cannot be used directly when we have a finite number of data. For example, if we have a data set with N pairs of x, y, choosing $d = N - 1$ will give us an *interpolating polynomial*, that is a polynomial that will go through every single point and, as a result, will have zero training error. But this model will overfit the data and will be a bad predictor for new points.

In the diagram below, a scatter plot of data generated from the function $y = x + \cos(x^2)$ with normal errors of $N(0, 0.05)$ is shown.



The black curve is the original function, the orange curve is a fitted polynomial of degree 5, and the purple one going through all points is the fitted polynomial of degree 20. Since there are only 21 points, this last polynomial has a training error of zero, but it does not in any shape resemble the original function.

Given the degree d the problem is now a regression problem and the parameters b_i can be estimated by the usual least squares method.

We will discuss shortly how to choose the degree sensibly, so that there is a balance between overfitting and flexibility.

When we have several variables $x_1, \ldots, x_d$, we can extend this idea to *multivariate polynomials*. For example, for two variables, x_1, x_2, the basis functions

are:

$$\varphi_0(x_1,x_2)=1,\varphi_1(x_1,x_2)=x_1,\varphi_2(x_1,x_2)=x_2,$$
$$\varphi_3(x_1,x_2)=x_1^2,\varphi_4(x_1,x_2)=x_2^2,\varphi_5(x_1,x_2)=x_1x_2,\ldots$$

It is now clear that we have a problem as the number of variables increases. If we have d variables and wanted polynomials of degree up to k, then the number of basis functions that can be used will be in the order of $\binom{d+k-1}{d}$, a number exponentially large in the number of variables d. For instance for ten variables, and polynomials of degree up to five, the number of basis functions could be 1001. This is indeed a large number of variables for a data set with only ten original variables.

Also, in case of dummy variables x_i that arise from categorical variables, we have $x_i^k = x_i$, since x_i can only be zero or one. In this case polynomial modeling reduces to choosing interaction variables, that is products of the form $x_{i_1}x_{i_2}\ldots x_{i_k}$.

Interpretation of coefficients and their p-values

One issue with nonparametric regression using basis functions is that the estimated parameters b_i are not easily interpretable. First, notice that testing whether $b_i = 0$ or not does not have a particular value. If for instance in a cubic polynomial approximation $y = b_0 + b_1x + b_2x^2 + b_3x^3$, it turns out that $b_2 = 0$ that is of no importance, we still have a cubic polynomials whose quadratic term happens to be zero. However, it is important whether $b_3 = 0$. It says that the cubic term is not statistically significant, and did not add much to the quadratic model $y = b_0 + b_1x + b_2x^2$.

Also in many applications, like medicine or economics, users wish to have some physical interpretation of the estimated coefficients b_i. In nonparametric setting, the b_i's individually don't have any interpretation. Only collectively they define a function that attempts to mimic the underlying unknown function $f(x)$.

Example: Splines as approximators

One issue with polynomials is that as the degree gets larger, the powers x^d become either very large if $|x| > 1$ or very small if $|x| < 1$. So we cannot increase d too much as the numerical roundoff error will soon become dominant in calculations. For instance, in the example posted for estimating a function $f(x)$ in R, when the degree of the polynomial surpasses 28, R cannot compute the results and gives an error message.

An alternative to polynomials is the class of *polynomial splines*. Formally, a function $g(x)$ is a polynomial spline (sometimes called a *B-spline*) if it is constructed as follows:

1. First the range of variable x, $[A, B]$ is subdivided into, say k sub-intervals $A < a_1 < a_2 < \cdots < a_k < B$. The a_i are called the *knots*. They are

usually chosen in equal distant from the next one, though this is not really a requirement.

2. The function $g(x)$ equals a polynomial $p_i(x)$ of degree at most d in each interval $[a_i, a_{i+1}]$:

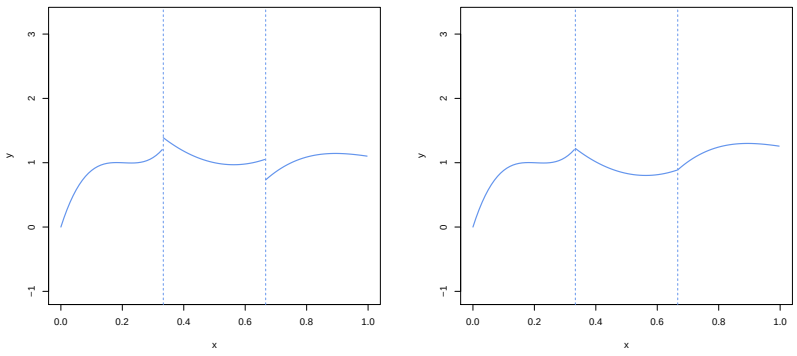
$$g(x) = \begin{cases} p_1(x), \text{ if } A \le x \le a_1 \\ p_2(x) \text{ if } a_1 \le x \le a_2 \\ \quad \dots \\ p_{k+1}(x) \text{ if } a_k \le B \end{cases}$$

The $p_i(x)$ are different from each other, making $g(x)$ a *piecewise polynomial* function. Also the degree of these polynomials d is fixed and usually small, for instance 3 or 4.

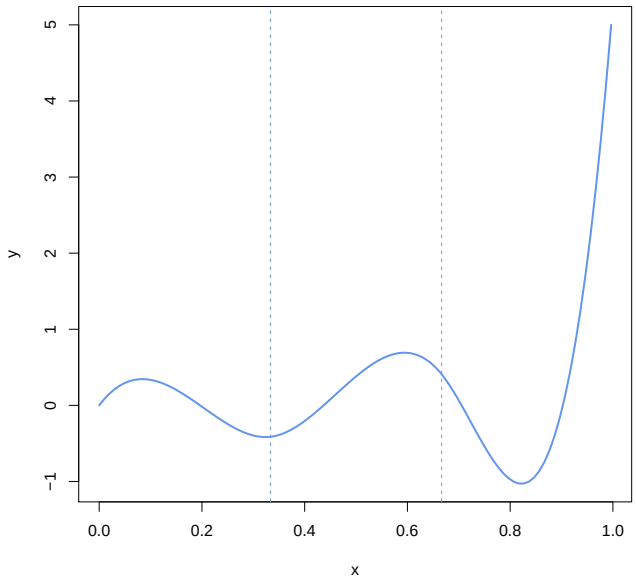
3. In addition every pair of consecutive polynomials are connected to each other *continuously* and *smoothly*, that is at each knot point a_i we have

$$\begin{aligned} p_{i-1}(a_i) &= p_i(a_i) \quad \text{(ensuring continuity)} \\ p'_{i-1}(a_i) &= p'_i(a_i) \\ p''_{i-1}(a_i) &= p''_i(a_i) \\ &\dots \\ p^{(d-1)}_{d-1}(a_i) &= p^{(i-1)}_i(a_i) \quad \text{all derivatives up to order } d-1 \text{ agree on each knot point} \end{aligned}$$

The effect is that even though each piece of $f(x)$ is a different polynomial, they are *stitched* together in a smooth way so that the whole function looks very smooth. See the following diagrams:



The left graph is a piecewise polynomial function that is not even continuous. The right one is continuous, but not smooth, that is its first derivative is not continuous. The following diagram is a polynomial spline of degree 3 (cubic spline), which is continuous, and its first and second derivatives are also continuous.



So, while in the case of polynomials the model complexity is determined by the degree of the polynomial, in the case of splines it is controlled by the number of knots. Note that in splines the degree is fixed (usually at a small number like 3 or 4), but the number of knots can get as large as we wish. To see the effect of knots on model complexity notice that if we have no knots at all, then the spline is just a single polynomial of degree 3 or 4 on the entire interval $[A, B]$. So the model complexity (the number of parameters) is $d+1$. If we have one knot, then we have *two* polynomials of degree d, so we have to determine $2(d+1)$ parameters. However, these two polynomials must have the same value at the knot point a_1, and similarly the same value of all derivatives of order up to $d-1$ at a_1. So these $2(d+1)$ parameters must satisfy $d-1$ linear equations. So the complexity effectively is $(2(d+1)-(d-1) = d+3$. If we have two knots a_1 and a_2, we will have *three* polynomials, so we need to determine $3(d+1)$ coefficients for the three polynomials. But the first two polynomials and their derivatives must agree at a_1 and the second two and their derivatives must agree at a_2. So we get a complexity of $3(d+1)-2(d-1) = d+5$. In general, if we have k knot points we will have to determine $k+1$ polynomials, each of degree d, so there are $(k+1)(d+1)$ coefficients altogether. However, each polynomial $p_i(x)$ must agree with the next one $p_{i+1}(x)$ at the knot point a_{i+1} and so do its derivatives of order up to $d-1$. So we get a total of kd equations for the coefficients. Therefore, the model complexity for splines is $(k+1)(d+1) - kd = k+d+1$. This is the `df` item reported in R.

An advantage of splines is that since the degree is bounded, we are not going to raise numbers to large powers, thus avoiding issues with very large numbers

in the case of polynomials.

Another class of splines are the *natural splines*. These are functions $g(x)$ defined on interval $[A, B]$, just like B-splines, except that at points A and B $g(x) = 0$ and all of its derivatives up to order $d-1$ are also zero. This means that the degrees of freedom for natural splines equals $(k+1)(d+1)-(k+1)d = k+1$. Thus, the complexity here is $k+1$, that is one plus the number of knots. The degree of polynomials does not affect the complexity of the model.

In terms of basis functions, it can be shown that linear combinations the following functions produces all polynomial splines of degree d and k knots:

$$\varphi_0(x) = 1, \varphi_1(x) = x, \ldots, \varphi_d(x) = x^d,$$
$$\varphi_{d+1}(x) = (x - a_1)_+^d, \ldots, \varphi_{d+k}(x) = (x - a_k)_+^d$$

where $(x - a_i)_+^d$ equals $(x - a_i)^d$ if $x \geq a_i$ and zero otherwise.

So, in effect, computing a spline approximation is computing the coefficients b_i of $\varphi_i(x)$ basis functions.

3 Classification and Logistic Regression with nonlinear boundaries

We can extend these ideas to logistic regression as well. So, for instance, if we have two variables x_1 and x_2, our logistic model for two classes would be:

$$\log\left(\frac{p}{1-p}\right) = b_0 + b_1x_1 + b_2x_2 + b_3x_1x_2 + b_4x_1^2 + b_5x_2^2$$

In general, the logistic model can be described by

$$\log\left(\frac{p}{1-p}\right) = f(x_1, \ldots, x_d|b_1, \ldots, b_k)$$

So long as the dependence of the function $f()$ on the *parameters* is linear, we can use logistic regression as before, using a maximum likelihood approach[2].

Notice that the boundary separating two classes is not a line necessarily. This boundary is determined by the equation $f(x_1, \ldots, x_d|b_1, \ldots, b_k) = 0$. This is much more flexible than a linear boundary represented by a hyperplane. Depending on the complexity (measured by the number of parameters k) the boundary could be quite complex and may consist of many different pieces.

Nonparametric Logistic Regression and Classification

Similarly to the linear regression, we can extend logistic regression to a nonparametric setting. In this case we are searching for a function $f(\mathbf{x})$ such that

[2]Even if dependence on parameters is not linear, we can use the maximum likelihood method. But in this case, the underlying likelihood or log likelihood functions may not concave any more, and we have many local minima.

$f(\mathbf{x}) = 0$ defines the boundary between the two classes. (Here $\mathbf{x} = (x1, \ldots, x_d)$ is the feature vector.) If we do not impose any particular formula on $f(\mathbf{x})$ then we have a nonparametric

In general, just like in regression, we have a sequence of *basis functions* $\varphi_1(\mathbf{x}), \ldots, \varphi_k(\mathbf{x}), \ldots$. We require that any potential boundary function $f(\mathbf{x})$ can be written as a linear combination of a (possibly infinite) number of basis functions:

$$f(\mathbf{x}) \approx b_1\varphi_1(\mathbf{x}) + \cdots + b_k\varphi_k(\mathbf{x})$$

Here, the boldface letter $\mathbf{x} = (x_1, x_2, \ldots, x_d)$, is the *feature vector*.

Just as in regression, we need to identify the correct level of complexity, that is k, and then estimate the parameters b_i through maximum likelihood method.

We could use for instance polynomials or polynomial splines. As an example, consider the case of the German credit approval where based on two features `A2, A3` we wished to predict if they will be approved for credit or not. In R using B-splines (with df=4) we get the following R model:

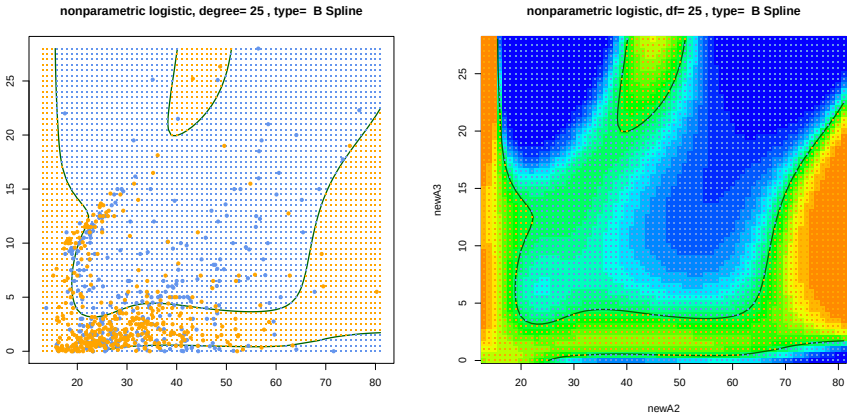
```
> summary(nlogitModel)

Call:
glm(formula = A16 ~ bs(A2, df = d) * bs(A3, df = d), family = binomial,
    data = credit)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.1496  -0.9807  -0.7626   1.1460   2.3787

Coefficients:
                                  Estimate Std. Error z value Pr(>|z|)
(Intercept)                        -1.0752     2.1452  -0.501   0.6162
bs(A2, df = d)1                    -0.8175     3.1451  -0.260   0.7949
bs(A2, df = d)2                     6.8233     2.9070   2.347   0.0189 *
bs(A2, df = d)3                    -2.3240     6.2130  -0.374   0.7084
bs(A2, df = d)4                     6.4269     6.7842   0.947   0.3435
bs(A3, df = d)1                    -0.3875     4.6613  -0.083   0.9337
bs(A3, df = d)2                    -6.2824    12.2322  -0.514   0.6075
bs(A3, df = d)3                    10.4282    31.4871   0.331   0.7405
bs(A3, df = d)4                   -26.7827    70.4710  -0.380   0.7039
bs(A2, df = d)1:bs(A3, df = d)1     0.9602     6.2186   0.154   0.8773
bs(A2, df = d)2:bs(A3, df = d)1    -7.2170     5.0894  -1.418   0.1562
bs(A2, df = d)3:bs(A3, df = d)1     1.7613    11.0553   0.159   0.8734
bs(A2, df = d)4:bs(A3, df = d)1    -3.7949    15.2706  -0.249   0.8037
bs(A2, df = d)1:bs(A3, df = d)2    17.4250    15.4582   1.127   0.2596
bs(A2, df = d)2:bs(A3, df = d)2    -2.8907    11.6360  -0.248   0.8038
bs(A2, df = d)3:bs(A3, df = d)2    24.1685    21.0428   1.149   0.2507
bs(A2, df = d)4:bs(A3, df = d)2   -14.6697    24.6039  -0.596   0.5510
bs(A2, df = d)1:bs(A3, df = d)3   -28.0847    39.3298  -0.714   0.4752
```

```
bs(A2, df = d)2:bs(A3, df = d)3  -8.5620    28.3150  -0.302   0.7624
bs(A2, df = d)3:bs(A3, df = d)3  -5.7986    43.0769  -0.135   0.8929
bs(A2, df = d)4:bs(A3, df = d)3 -17.9890    39.3483  -0.457   0.6475
bs(A2, df = d)1:bs(A3, df = d)4  80.4530    93.2409   0.863   0.3882
bs(A2, df = d)2:bs(A3, df = d)4 -21.5001    59.5594  -0.361   0.7181
bs(A2, df = d)3:bs(A3, df = d)4  54.4123    85.4212   0.637   0.5241
bs(A2, df = d)4:bs(A3, df = d)4  27.9327    72.0366   0.388   0.6982
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 931.48  on 676  degrees of freedom
Residual deviance: 845.34  on 652  degrees of freedom
AIC: 895.34

Number of Fisher Scoring iterations: 6
```

As it can be seen, writing `A16 ~ bs(A2, df = d) * bs(A3, df = d)` in R means that all combination of products of B-spline functions with degrees of freedom from 1 to 4 in variable `A2`, and B-spline functions with degrees of freedom from 1 to 4 for variable `A3` are created, and their cross products are also added to the model. We can also see that the coefficient of each basis function used. The equation written in R that defines the boundary is now given by:

```
-1.0752 -0.8175 * bs(A2, df = 1) + 6.8233 * bs(A2, df = 2) + ...
        + bs(A2, df = 4):bs(A3, df = ) * 27.9327 = 0
```

The graph and the *heat map* of this relation is shown below. The large dots are from the data set published in the UCI repository. The orange point corresponds to the rejected credit and the blue ones to accepted.



The heat map depicts the probabilities of acceptance of credit. The range is from orange, for very small probability of acceptance to blue, for very high probability of acceptance.

Looking at the summary above, it seems as if all variables are insignificant. However, running analysis of deviance results in:

```
> anova(nlogitModel, test="LRT")
Analysis of Deviance Table

Model: binomial, link: logit

Response: A16

Terms added sequentially (first to last)


                             Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                                           676     931.48
bs(A2, df = d)                4   24.314       672     907.16 6.910e-05 ***
bs(A3, df = d)                4   35.283       668     871.88 4.063e-07 ***
bs(A2, df = d):bs(A3, df = d) 16  26.544       652     845.34   0.04684 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

This shows that the model as a whole has significance.

4 Model Selection: Choosing the Right Level of Complexity

In the nonparametric setting, both for regression and classification, if we allow the complexity to become too large, then we may reach a point that the training error will be zero. In regression this means that with high enough complexity the estimated curve will go through every point. For classification, this means that the boundary perfectly separates the classes in the training set.

By now we know that overfitting is no good since a model which fits perfectly, or even too well, to the data may be a poor predictor for new data points.

To find the right level of complexity, there are many strategies. We describe a few here.

Validation set and cross validation

The validation set technique is simple and we have seen it already in the k-NN method before.

Algorithm Validation set
set bestError= ∞, bestk=0
For all possible complexity levels k under consideration do
    Set summError=0

Repeat R times
—Select a random subset of data as training set, and the remainder as test set
—Use the training set to build the model of complexity k
—Compute erroRate for this model for the test set
—set sumError=SumError + erroRate
End Repeat
set ErrorRate(k)=sumError/R
If ErrorRate(k) < bestError
—Set bestError=ErrorRate(k)
—Set bestk=k
End if
End For
Output bestk and bestError
End Algorithm

The result of the validation set method is usually pretty good if the data size is large enough. However, it is relatively expensive, since for each complexity level k we are taking R random samples from the data and, as a result, repeat the learning procedure R times. For example, for logistic regression using splines we will have to run the logistic regression process R times, each for every df level we are considering.

The error rate is set to the least squares value for the linear regression, and to the negative of the likelihood function (or the *deviance* which is $-2\log \text{Lik}$) for logistic regression is used.

r-folding cross validation

There is a variant of validation set method called *r-folding cross validation*. In this method the data is divided into roughly equal and random parts, each of size r (or very close to it). Then for each complexity level k we develop N/r different models. Each model is obtained by setting aside one of the parts as the test set and the remaining $(N/r)-1$ parts are used as the training set. For each of these models, once built, the error rate on the test set is computed and then the average over the r models is the average error rate for complexity level k. The complexity level k for which the smallest error rate is achieved is reported as the optimal complexity level for the given data.

Finally, when $r = 1$ we will get a variant some times called *leave-one-behind cross validation*. In this case one of the data items is removed, and learning is done with the remainder. Then the prediction error is calculated for the one data item set aside. The process is repeated over all data points and the average error is reported as the error rate.

The advantage of leave-one-behind cross validation is that recomputing the maximum likelihood for each model from knowledge of the previous data set is faster. After all data sets only differ in one item. So if we know the maximum likelihood estimates for one we can compute the maximum likelihood estimate

for the other quickly. So this method is somewhat faster than validation set technique.

Bayesian approach to model selection

To be completed...

Akaiki Information Criterion (AIC) and Bayesian Information Criterion (BIC)

Instead of creating a test set and a training set from the data, the following methods simply penalize the complexity directly. There are many approaches for this, but the most popular ones are: *Akaiki Information Criterion (AIC)* and *Bayesian Information Criterion (BIC)*:

$$\begin{aligned} \text{AIC} &= -2\log \text{Lik} + 2k \\ \text{BIC} &= -2\log \text{Lik} + \log(N)k \end{aligned}$$

Here k is the complexity of the model and N is the number of data points, and $-2\log \text{Lik}$ is the deviance.

The maximum likelihood method attempts to (effectively) minimize the deviance. As we add more and more complexity, that is more and more features to the model, the deviance gets smaller and smaller. This is true even if the new features are not significant and add little to the model. In AIC, the term $2k$ adds a penalty of 2 for each new feature added. So when we add a new feature to our model, the improvement (reduction) in deviance must be at least 2, otherwise the AIC value gets larger.

The same is true for the BIC, except that the penalty in the case of BIC is much more severe. So for each new feature, the BIC is increased by $\log(N)$. Therefore, the reduction in deviance as a result of adding a new feature should be more than $\log(N)$ to justify its addition to the model. For $N = 1000$, for instance, $\log(N) = 6.91$, so adding a feature should decrease the deviance by at least 6.91 to be worth including it in the model.

AIC and BIC are much easier and less costly to compute than the cross validation techniques. They are however, rougher estimates, and a bit less reliable than cross validation which is purely data dependent procedure.

5 Regularization

An automatic way to select which features are more significant and which ones are less is through a method called *regularization*. The idea is to penalize the model for the *size* of the parameters. Note that here it is important that the data are scaled so that the sizes of parameters are on the same order of magnitude. In the maximum likelihood context we now seek

$$\theta^* = \text{argmin}_\theta - \log \text{lik}(\theta \mid X) + \alpha\|\theta\|.$$

The notation $\|\cdot\|$ means the *norm* of the vector. There are many popular norms and each one is used with some advantages and disadvantages.

5.1 Regularization with the Euclidean norm and Ridge regression

If we define $\|\theta\| = \|\theta\|_2^2 = \left(\sum_i \theta_i^2\right)$, then we get *ridge regression*, sometimes also called *Tikhonov Regularization*. For linear regression, for instance, the problem is defined as :

$$\min_{w} (y - Xw)^\top (y - Xw) + \lambda \|w\|^2$$

It turns out that the solution for this problem can also be expresses analytically as follows:

$$w = \left(X^\top X + \lambda I\right)^{-1} X^\top y$$

There are several interesting to note about this formulation. First, by increasing λ a value that has to be determined separately, we penalize more heavily on the norm of w. Furthermore, by encouraging $\|w\|$ to stay small, we indirectly guide the process to only pick those features x_i which are essential, and those which are not so essential, their w_i are going to be guided toward smaller numbers.

Another application of this regularization is in the case where the number of features is so large that it might be even larger than number of data points. In this case, the matrix $X^\top X$ is singular, and thus, does not have a well-define inverse. However, Tikhonov regularization removes this problem by adding the term λI. This issue becomes specially important in models such as *neural networks* and *deep learning* where one may have to estimate hundreds of thousands, or even millions of weight values.

In the ridge regression method, while the weights of the "less important" features are smaller, they seldom are zero. So, in this approach all features more or less still have a say in the final model.

Adding the $\lambda\|w\|^2$ term can be used in any approach based on the maximum likelihood principle, in particular, logistic regression. Also, in Support vector machines, as we saw, this term pops up as a penalty term for the violation of separation condition.

Finally, for the linear regression there is a Bayesian interpretation. If we have a prior distribution on the weights which assumes they are independent (that is their joint distribution is the product of their marginal distributions) and each follow the normal distribution with a mean zero and the same variance σ_p^2, then the MAP estimate will be equivalent to the Tikhonov regularization.

5.2 Regularization with the ℓ_1 norm: The Lasso

In this approach, instead of the Euclidean norm the ℓ_1 norm is used. Recall that the ℓ_2 norm of a vector w is defined as follows:

$$\|w\|_1 = |w_1| + \cdots + |w_d|$$

that is, the sum of the absolute values of each of its components. So the optimization problem for linear regression is formulated as follows:‘

$$\min_{\boldsymbol{w}} (\mathbf{y} - X\boldsymbol{w})^\top (\mathbf{y} - X\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1$$

Now, for this problem, unlike the unregularized, or the Tikhonov regularization, there is no closed form formula for the optimal $\boldsymbol{w}$. While the optimization problem remains a convex one, it has become non-differentiable (since $\|\boldsymbol{w}\|_1$ as a function of w_i is non-differentiable at zero.) Nevertheless, there are effective algorithms to solve this problem and find the optimal $\boldsymbol{w}$. The main interesting feature of the lasso approach to regularization is that the optimal solution tends to have many components w_i equal to zero. So, in a sense, this approach completely eliminates those features x_i which are not that important by setting their weight $w_i = 0$.

Again, the lasso approach can be used in any other method derived from the maximum likelihood approach, including logistic regression an neural networks.

There are many other forms of regularization techniques. Some of these methods are specific to a particular class of algorithms. For instance, for neural networks, there are regularization techniques such as early stopping in the optimization process, and *dropout* (to be covered later in the course,) and many others.