

Advanced Regression and Prediction

Javier Nogales

www.est.uc3m.es/nogales

@fjnogales

MS in Statistics for Data Science

2020

Objectives

- Relax some of the assumptions in classical linear regression (normality, loss functions, etc.)
- Estimate efficiently the parameters of the models
- Deal with the curse of dimensionality in high-dimensional problems
- Know the main tools in advanced predictive modeling, including machine learning
- Handle the R language for advanced regression (including caret package)

Organization

- Subject organized in two main topics: statistical-based models and machine-learning tools (roughly 60% + 40%)
- 7 weeks
- Practical course: 50% basic concepts + 50% computer labs (using R)
- **Online evaluation:** one final project divided in two parts

First part: prepare the input and statistical-learning tools

Second part: machine-learning tools and prepare the output

- Based on **advanced predictive modeling** using real data

Motivation

Predictive Modeling Framework

- Usual framework in [Machine Learning](#) / [Statistics](#):

$$\text{Data} = \text{Model} + \text{Noise}$$

- When we focus on one variable predicted by others: [supervised learning](#)

$$y = g(x_1, \dots, x_k) + \text{Noise}$$

- Statistical (linear) approximation: $y = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p + \epsilon$
- Machine learning approximation: $y = \text{map}(x_1, \dots, x_p) + \epsilon$
- Note the variables in the true model, g , may be different in the approximations
- Two main categories:
 - [Classification](#): y is categorical
 - [Regression](#): y is numeric (usually continuous)

Some questions answered by predictive models (regression)

- What will the electricity price be in Spain tomorrow? (depending on demand, wind, etc.)
- How can we determine the credit scoring of a bank client? (income, consumption, age, etc.)
- What will the fourth quarter sales be for a given company? (?)
- How many new followers will I get next week? (?)
- How can I determine the value of my home? (?)
- What is the beta of a stock? (?)

Note to answer these questions we need additional information:
how those variables are related with another ones

Linear regression: a brief review

One of the main tools in [Statistics](#) and [Machine Learning](#)

On the basis of a dataset, some questions to answer:

- Are there relationships between one variable and others?
Analyze [evidence](#) of association
- How strong are those relationships?
Analyze the [strength](#) of association
- Can we use those relationships to [predict better](#) a variable?
- How [accurate](#) is that prediction?
- How to assess whether a model is [valid](#)? What to do if it is not valid?

Linear regression: a brief review

- The target, y , is a continuous variable that depends linearly on predictors
- Although the linear assumption may seem simplistic, it is extremely useful in practice
- Linear regression can deal with interactions and non-linearities
- Estimation is based on **least squares**: the loss function is the sum of residual squares (training prediction error)
- But what happens if the input comes from Big Data (high dimension)?

Linear regression: a brief review

- For the model $y_i = \beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip} + \epsilon_i$

- Assumptions

① $E(\epsilon_i) = 0 \quad \forall i$

② $\text{Var}(\epsilon_i) = \sigma^2 \quad \forall i$

③ $E(\epsilon_i \epsilon_j) = 0 \quad \forall i \neq j$

④ $\epsilon_i \sim \text{Normal} \quad \forall i$

① $E(y_i) = \beta_0 + \beta_1 x_{i1} + \dots + \beta_k x_{ip} \quad \forall i$

② $\text{Var}(y_i) = \sigma^2 \quad \forall i$

③ $\text{Cov}(y_i, y_j) = 0 \quad \forall i \neq j$

④ $y_i \sim \text{Normal} \quad \forall i$

- Note (3) and (4) imply the observations are independent
- We do not need (4) for estimation, but for inference

What happens if some assumptions do not hold?

- In practice, the output may not be continuous
- The errors may not be normal
- The relation between the output and the predictors may not be linear

Generalized Linear Models (GLM): try to extend linear models by relaxing previous assumptions in an unified way

Non-linear regression: specify a parametric non-linear (in parameters) relation and estimate the parameters using an optimization solver

Regression in high-dimension: OLS has high variance in high dimension, advanced tools try to reduce the variance while increasing the bias

What is the meaning of linear?

- Are these models linear?

$$y_i = \beta_0 + \beta_1 x_{i1}^2 + \beta_2 x_{i2}^2 + \beta_3 x_{i1} x_{i2} + \epsilon_i$$
$$\log(y_i) = \beta_0 + \beta_1 x_{i1} + \epsilon_i$$

- Are these models linear?

$$y_i = \beta_0 + \beta_1 x_{i1}^{\beta_2} + \epsilon_i$$
$$y_i = \frac{\beta_0}{1 + \frac{x_{i1}}{\beta_1^2}} + \beta_2 x_{i2} + \epsilon_i$$

GLM vs Transformations

- When the assumptions of linear regression are not met, we can always try to transform the output to make it more linear respect to the predictors, or to make its variance more homogeneous, etc.
- So, when is GLM better?
- The link function transforms the mean, not the output

Example: if y is a price and we take the logarithm, then we can model $E(\log(y)) = \beta'x$

By GLM, we model $\log(E(y)) = \beta'x$, which avoids difficulties when $y = 0$

- Transforming the mean often allows the results to be more easily interpreted
- GLMs are more flexible: they allow for unequal variance, non-linear relationships, binary/categorical/count outcomes, skewed distributions, etc.

Regression in high dimension: an advance

- In high dimension, or p/n large, the dataset may contain redundant information about the response because some predictors may be correlated (**collinearity**)
- In other words: more information (more variables) is not necessary better, because this new information will come with much more noise in high dimension

Hence, how to improve linear models when the dimension is high?

- **Statistical Learning**: improve interpretability while attaining good predictive performance, replace least squares with some alternative fitting tools
- **Machine Learning**: focus on prediction accuracy, deals better with large scale applications

Today, much overlap between both views

Some Complementary References

- James, G., Witten, D., Hastie, T., Tibshirani, R. (2013). An Introduction to Statistical Learning with Applications in R. Springer
- Chatterjee, S. and A. S. Hadi (2012). Regression analysis by example. John Wiley & Sons
- Sheather, S. J. (2009). A modern approach to regression with R. Springer

Advanced Regression and Prediction

- 1 Advanced Estimation for Least-Squares
- 2 Extending Linear Models
- 3 Statistical Learning Tools
- 4 Machine Learning tools

1. Advanced Estimation for Least-Squares

Estimation tools: Least-Squares

- Ordinary least-squares approach (OLS):

$$\text{minimize}_{\beta} \quad \frac{1}{2} \sum_i (y_i - x_i^T \beta)^2$$

- Explicit solution: $\beta^* = (X^T X)^{-1} X^T y$
- Can deal with non-linearities: $x_i = \phi(z_i)$ or $x_i = z_j \cdot z_k$, etc.

- Nonlinear least-squares approach:

$$\text{minimize}_{\beta} \quad \frac{1}{2} \sum_i (y_i - F(\beta; x_i))^2$$

- No explicit solution: nonlinear unconstrained problem
- An efficient optimization solver is needed

Efficient estimation for OLS: QR factorization

- Because in OLS: $\beta^* = (X^T X)^{-1} X^T y$, the solution may be ill-conditioned
- Condition number: $\kappa(X^T X) = \kappa(X)^2$
- This is the classical approach ([Cholesky factorization](#)): $(X^T X)\beta = X^T y$
- But in high dimension, matrix is close to singular: need more numerically stable algorithms [QR factorization](#) and [SVD decomposition](#)

Efficient estimation for OLS: QR factorization

For a given data matrix X with n rows and p columns

- **QR**: orthogonalization algorithm that deals directly with X instead of $X^T X$
- Decomposition: $X = QR$
where Q is a $n \times n$ orthogonal matrix and R is a $n \times p$ right triangular one
- Apply to each column in X the Householder projection
- Then, $Q = (H_p \times \cdots \times H_2 \times H_1)^T$ and $R = Q^T X$
- Now, instead of solving $X\beta \simeq y$, solve $R\beta = z$, where $z = Q^T y$
- In practice, we do not need to compute Q
- This procedure is numerically more stable (**less condition number**) but more computationally expensive (not too much) than Cholesky

QR factorization for OLS: Example

X =				y =
1.0000	-1.0000	1.0000	-1.0000	75.9950
1.0000	-0.8182	0.6694	-0.5477	91.9720
1.0000	-0.6364	0.4050	-0.2577	105.7110
1.0000	-0.4545	0.2066	-0.0939	123.2030
1.0000	-0.2727	0.0744	-0.0203	131.6690
1.0000	-0.0909	0.0083	-0.0008	150.6970
1.0000	0.0909	0.0083	0.0008	179.3230
1.0000	0.2727	0.0744	0.0203	203.2120
1.0000	0.4545	0.2066	0.0939	226.5050
1.0000	0.6364	0.4050	0.2577	249.6330
1.0000	0.8182	0.6694	0.5477	281.4220
1.0000	1.0000	1.0000	1.0000	308.7480

Sequential Householder reflections are applied to X and y

QR factorization for OLS: Example

- First step: apply Householder reflection for the first column in X to introduces zeros below the first element; the same Householder reflection is applied to y

$$R = H_1 X, \quad z = H_1 y$$

- Second step: apply other Householder reflection for the second column in X to introduces zeros below the second element; the same Householder reflection is applied to y

$$R = H_2 H_1 X, \quad z = H_2 H_1 y$$

- Repeat this process p times

$$R = H_p \cdots H_2 H_1 X, \quad z = H_p \cdots H_2 H_1 y$$

QR factorization for OLS: Example

R =				z =
-3.4641	0.0000	-1.3646	-0.0000	-614.3267
0	2.1742	-0.0000	1.5274	252.2337
0	0	1.2077	0.0000	33.2463
0	0	0	-0.6469	0.7263
0	0	0	0	-5.9906
0	0	0	0	-5.8013
0	0	0	0	2.2826
0	0	0	0	3.8110
0	0	0	0	2.8095
0	0	0	0	-0.4061
0	0	0	0	2.8749
0	0	0	0	-0.5868

LS solution with QR is $R(1 : 4, 1 : 4)\beta = z(1 : 4)$

Efficient estimation for OLS: SVD decomposition

For a given data matrix X with n rows and p columns, and an output y

$$\text{minimize } \|y - X\beta\|_2^2$$

- If $r(X) = p$, then unique solution
- If $r(X) < p$, then infinitely many solutions
 - Perform the SVD decomposition of X : $X = U\Sigma V^T$
Use the pseudo-inverse $X^+ = V\Sigma^+U^T$, and $\beta = X^+y$ is the minimum-norm solution, where Σ^+ is computing by replacing every non-zero element by its reciprocal and transposing the resulting matrix
 - Use the solution with fewest possible nonzero components (based on QR factorization)

SVD for OLS: Example

Let

$$X = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \\ 13 & 14 & 15 \end{pmatrix} \quad y = \begin{pmatrix} 16 \\ 17 \\ 18 \\ 19 \\ 20 \end{pmatrix}$$

Note in this case $\Sigma = \begin{pmatrix} 35.18 & 0 & 0 \\ 0 & 1.48 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$. That implies the rank of X is 2

$$\text{Note } \Sigma^+ = \begin{pmatrix} 0.029 & 0 & 0 & 0 & 0 \\ 0 & 0.678 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad X^+ = \begin{pmatrix} -0.389 & -0.244 & -0.100 & 0.044 & 0.189 \\ -0.022 & -0.011 & 0.000 & 0.011 & 0.022 \\ 0.344 & 0.222 & 0.100 & -0.022 & -0.144 \end{pmatrix}$$

SVD for OLS: Example

- If we compute β as $\beta = X^+y$ where $X^+ = V\Sigma^+U^T$,

$$\beta = \begin{pmatrix} -7.5556 \\ 0.1111 \\ 7.7778 \end{pmatrix}, \text{ and it has the lowest norm}$$

- If we compute β as $R\beta \simeq Q^Ty$ where $X = QR$,

$$\beta = \begin{pmatrix} -7.5 \\ 0 \\ 7.8333 \end{pmatrix}, \text{ and it has at most 2 non-zero elements}$$

Matrix Decompositions for OLS: Summary

- Cholesky is less stable than QR, which is less stable than SVD
- But SVD is more expensive than QR, which is more expensive than Cholesky
- When the problem is degenerate, $r(X) < p$, use SVD decomposition
- If the problem is well-conditioned, use classical OLS (Cholesky or normal equations)
- In problems in between well-conditioned and degenerate
 - Use QR if we do not want bias in the solution
 - Use ridge or lasso regression if we allow for bias (reducing then the variance)
- In R, `lm(y ~ x)` uses QR. For SVD try: `ginv(x) %*% y`
- In Python, SVD is used by default (either in numpy or scikit-learn), but QR is also a choice

Advanced Regression and Prediction

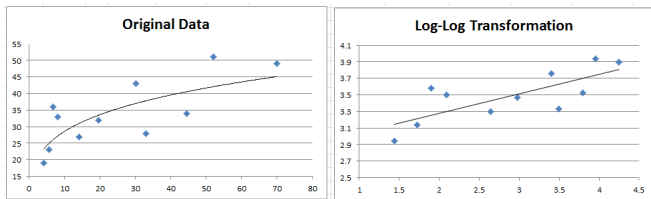
- 1 Advanced Estimation for Least-Squares
- 2 Extending Linear Models
- 3 Statistical Learning Tools
- 4 Machine Learning tools

2. Extending Linear Models

- Feature Engineering: transformations
- Non-linear regression
- GLM: non-linear transformation of the mean
- Local regression: non-parametric, kernels
- Robustness

Feature Engineering: transformations

- The model, or predictor function, is linear in the features but not the data
- Hence $x' = \log(x)$ is still a linear model
- A dummy variable is still a linear model
- $x' = x_1 \cdot x_2$ is still a linear model
- $\beta_1 x + \beta_2 x^2$ is still a linear model



Feature Engineering: transformations

- Usual transformations:
 - If $y \simeq \beta_0 e^{\beta_1 x}$, then $y' = \log(y)$
 - If $y \simeq \beta_0 x^{\beta_1}$, then: $y' = \log(y), x' = \log(x)$
 - For Poisson counts or areas, use $y' = \sqrt{y}$
 - For ratios, use $y' = \log(y)$
 - For rates, use $y' = \frac{1}{y}$
- Do not forget to transform back the predictions to the original scale for interpretation
- The scatter plot is quite useful to decide the transformations

Feature Engineering: non-linearities

- Sometimes the linear approximation is not enough to capture the relations in

$$y = g(x_1, \dots, x_k) + \text{Noise}$$

- We can add higher terms in the linear model. For instance:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \epsilon$$

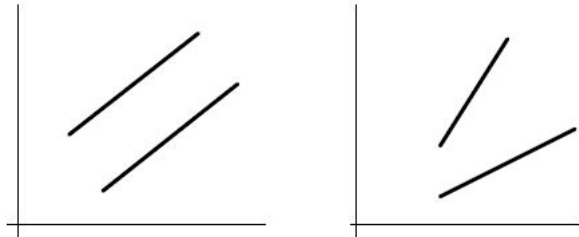
Note the model is still linear (in the estimation)

- Convenient to center the data previously
I.e. use $x_1 - \bar{x}_1$ instead of x_1 to reduce collinearity (estimation noise)
(because x_1 will be correlated with x_1^2)

Feature Engineering: Interactions

- When two features have a combined effect, this is known as an **interaction**: $x_1 \cdot x_2$
- Specially useful when one variable is numeric, x , and the other categorical, z

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 z_i + \beta_3 x_i z_i + u_i$$



In the first case, no interaction, $\beta_3 = 0$

In the second case, relevant interaction, $\beta_3 \neq 0$

- Interactions should be included in a model with each of the interacting variables

Feature Engineering: polynomial regression

- Polynomial approximation between the response and one predictor:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_h x_i^h + u_i$$

where h is the degree of the polynomial

- Of course, other variables (predictors) can be added. And also with interactions
- Note polynomial regression is still considered linear regression!
- Take care: the higher h , the bigger collinearity and overfitting
- Convenient to center the data previously
- Take care: do not extrapolate beyond the limits of your observed values

Non-linear Regression

- Are these models linear?

$$y_i = \beta_0 + \beta_1 x_{i1}^2 + \beta_2 x_{i2}^2 + \beta_3 x_{i1} x_{i2} + \epsilon_i$$
$$\log(y_i) = \beta_0 + \beta_1 x_{i1} + \epsilon_i$$

- Are these models linear?

$$y_i = \beta_0 + \beta_1 x_{i1}^{\beta_2} + \epsilon_i$$
$$y_i = \frac{\beta_0}{1 + \frac{x_{i1}}{\beta_1^2}} + \beta_2 x_{i2} + \epsilon_i$$

Non-linear Regression

- Analyze non-linear relationships in the form:

$$y = F(\beta; x) + \text{Noise}$$

- We assume the parametric function F is known, except for the parameters β
- Estimation is done through **nonlinear least-squares**:

$$\text{minimize}_{\beta} \quad \frac{1}{2} \sum_i (y_i - F(\beta; x_i))^2$$

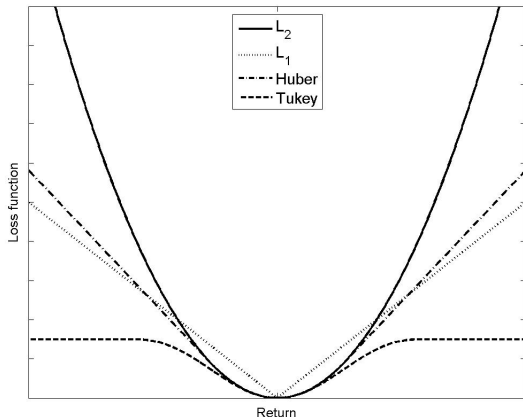
- Optimization algorithms are needed (gradients and Hessian matrix)
- Logistic regression can be viewed as a particular case, and also it is a particular case of GLM

Robustness

- **Outliers** may affect regression estimation: a simple outlier can dramatically alter the outcome of a regression
- How can deal with them in an automatic way?
- I.e. without trying to detect them previously and removing them
- **Robustness**: a way to achieve more robust outcomes by changing OLS objective function

Robustness: General Loss Functions

- Residuals in OLS can be written as $\sum_i (y_i - x_i^T \beta)^2 = \|y - X\beta\|_2^2$
- Hence, other versions if we change the norm (loss function)
- For instance, loss function in OLS is $\rho(x) = x^2$



Robustness: Least-absolute or L_1 regression

- The loss function is $\rho(x) = |x|$
- It gives less weight to large errors (outliers):

$$\text{minimize}_{\beta} \quad \|y - X\beta\|_1 = \sum_i |y_i - x_i^T \beta|$$

No explicit solution

- This approach is **robust** against outliers: the absolute value gives less weight to points far from the center than OLS

Robustness: Chebyshev regression

- Minimax approach:

$$\text{minimize}_{\beta} \quad ||y - X\beta||_{\infty} = \min \max |y_i - x_i^T \beta|$$

No explicit solution

- This approach is also robust but in a minimax sense: it minimizes the worst residual
- Very conservative approach

Robustness: Robust regression

- A general robust framework:

$$\text{minimize}_{\beta} \quad \sum_i \rho\left(\frac{y_i - F_i(\beta; x_i)}{\sigma_i}\right),$$

where σ_i is a scale parameter (perhaps not included) and ρ is a loss (error) measure

Robust regression: particular cases

- Least squares: $\rho(x) = \frac{x^2}{2}$
- L_p estimators: $\rho(x) = |x|^p$, with $1 < p < 2$
- Lorentz estimator: $\rho(x) = \log(1 + \frac{1}{2}x^2)$
- Truncated least-squares estimator:

$$\rho(x, k) = \begin{cases} x^2, & |x| < \sqrt{k}, \\ k, & \text{otherwise} \end{cases}$$

- Huber function, etc.

$$\rho(x, k) = \begin{cases} \frac{1}{2}x^2, & |x| < k, \\ k|x| - \frac{1}{2}k^2, & \text{otherwise} \end{cases}$$

- Others

Advanced Regression and Prediction

- 1 Advanced Estimation for Least-Squares
- 2 Extending Linear Models
- 3 Statistical Learning Tools
- 4 Machine Learning tools

3. Statistical Learning Tools

- Model Selection
- Regularization Methods
- Dimension Reduction

Advanced Regression

How to improve simple linear models when the dimension is high?

- Try to improve the interpretability while attaining good predictive performance
- Need to replace least squares with some alternative fitting tools
- Need to balance prediction accuracy versus model interpretability (feature selection)

Collinearity

- Phenomenon due to redundant information about the response because some predictors in the model are correlated
- Usually, more information (more variables) is not necessary better, because adding more variables will result in inability to visualize that information
- That implies a close-to-singularity matrix $X'X$ (large condition number)
- That implies beta's are estimated with high noise
- That implies confusing and misleading results about the effects: non-reliable t-tests for parameters but reliable F-test, non-significant parameters in multiple regression but significant ones in simple regressions, etc.

Overfitting

- In high dimension (p/n is large), collinearity appears almost sure (redundant information)
- May imply **overfitting**: more parameters than needed, bad generalization (out-of-sample prediction)
- Estimation by OLS is not reliable (bad explanation):

$$E(\text{Var}(\hat{\beta})) = \sigma^2 \text{trace}((X^T X)^{-1})/p$$

- Prediction of output is reliable within the range of historical data (in-sample):

$$E(\text{Var}(\hat{y})) = E(\text{Var}(X\hat{\beta})) = \sigma^2 \text{trace}(X(X^T X)^{-1}X^T) = \sigma^2 p/n$$

- But maybe not reliable if predictors are outside that range:

$$\text{Var}(\hat{y}_0) = \text{Var}(x_0'\hat{\beta}) = \sigma^2 (x_0'(X^T X)^{-1}x_0)$$

- Hence, how can we estimate with some accuracy β ? **To explain**
- And, are all the p variables really needed to predict y ? **To predict better**

To explain or to predict?

In regression/classification, there are three sources of uncertainty:

- The error in the coefficients when the linear approximation is true (**estimation error**)
- The error in the linear approximation when the true model is non-linear, or contains other variables (**model bias**)
- The noise in the DGP: $\text{Data} = \text{Model} + \text{Error}$ (**irreducible error**)
- Error decomposition: $\text{Prediction Error} = \text{Data} - \widehat{\text{Model}}$

$$(\text{Prediction Error})^2 = \sigma^2 + \text{Bias}^2 + \text{Var}$$

To explain or to predict?

$$(\text{Prediction Error})^2 = \sigma^2 + \text{Bias}^2 + \text{Var}$$

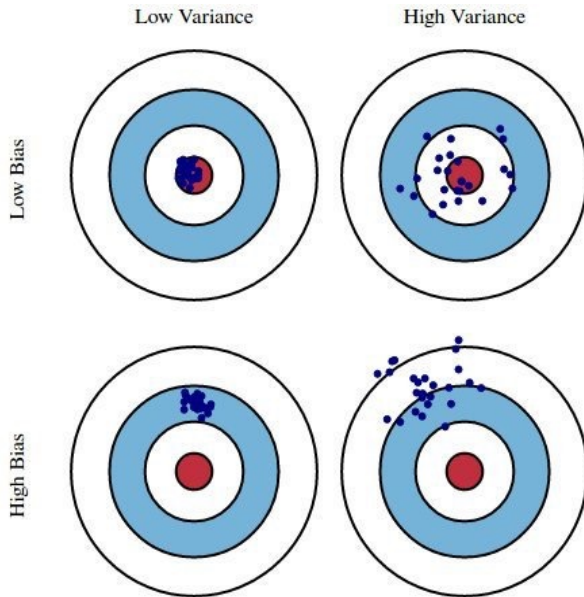
- **Statistics:**

- Focus on minimizing Bias (by assuming knowledge about population, DGP)
- Hence, able to obtain formulas for Var that **provides explanation** (inference, effects of predictors on response)
- The Var can be large in practice

- **Machine Learning:**

- Focus on minimizing $\text{Bias}^2 + \text{Var}$
- No assumptions needed (discover knowledge), hence no formulas and no explanation
- But **good prediction** performance

Predictions: Bias vs Variance



Prediction error of linear models

- General decomposition: $(\text{Prediction Error})^2 = \sigma^2 + \text{Bias}^2 + \text{Var}$
- For a linear model $y = x^T \beta + \epsilon$ with p variables (features) and n observations
- Assume no bias: we are using the same variables as in the population
- Prediction error in training set:

$$(\text{Prediction Error})^2 = E((y - \hat{y})^2 | x) = \sigma^2 + 0 + \sigma^2 \frac{p}{n}$$

It does not depend on collinearity, but increases with p

- Prediction error in testing set:

$$(\text{Prediction Error})^2 = \sigma^2 + 0 + \sigma^2 (x_{\text{test}}^T (X_{\text{train}}^T X_{\text{train}})^{-1} x_{\text{test}})$$

That implies reliable predictions within the range of historical data ($x_{\text{test}} \in X_{\text{train}}$)

Overfitting and underfitting in practice

- Consider the following true data generating process (DGP):

$$y = x_1\beta_1 + \cdots + x_p\beta_p + \epsilon = x'_{\text{true}}\beta + \epsilon$$

where $E(\epsilon) = 0$ and $\text{Var}(\epsilon) = \sigma^2$. The corresponding OLS estimator is denoted by $\hat{\beta}_{\text{true}}$

- Overfitting:** the model is estimated with more variables than needed ($q > p$ variables):

$$y = x_1\beta_1 + \cdots + x_q\beta_q + \epsilon = x'_{\text{over}}\beta_{\text{over}} + \epsilon$$

If $\hat{\beta}_{\text{over}}$ denotes the OLS estimator, then the prediction is unbiased but with larger variance:

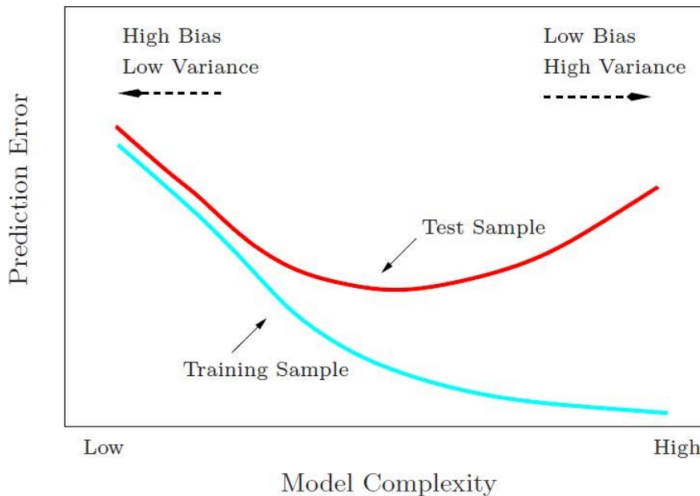
$$E(x'_{\text{over}}\hat{\beta}_{\text{over}}) = x'_{\text{true}}\beta, \quad \text{Var}(x'_{\text{over}}\hat{\beta}_{\text{over}}) \geq \text{Var}(x'_{\text{true}}\hat{\beta}_{\text{true}})$$

- Underfitting:** the model is estimated with less variables than needed ($q < p$ variables). Then, the prediction is biased but with smaller variance, i.e.

$$E(x'_{\text{under}}\hat{\beta}_{\text{under}}) \neq x'_{\text{true}}\beta, \quad \text{Var}(x'_{\text{under}}\hat{\beta}_{\text{under}}) \leq \text{Var}(x'_{\text{true}}\hat{\beta}_{\text{true}})$$

Overfitting

When a model fits or predict very well the training data but bad the testing data
(p is large compared with n)



- Model Selection

Model Selection

Assume we have a high-dimensional regression problem:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \varepsilon = \mathbf{X}\beta + \varepsilon$$

where p/n is large

Two main questions:

- How can we estimate with some accuracy β ?
- Are all the p variables needed to predict y ?

Model Selection

- Strong objective: identify **the** subset of the p variables that is **really associated** to the response
This may be prohibitive: if $p = 50$, then more than 10^{15} models to try!
- Weak objective: identify **a** subset of the p variables that is **relevant** to the response
But, what is the meaning of relevant?
- Some quality measures: adjusted R^2 , Akaike information criterion (AIC), Bayesian information criterion (BIC), based on cross-validation, based on fitting in testing set,...

Model Selection: AIC

- For a given estimate $\hat{\beta}_d$ of the model $y = X_d\beta_d + \varepsilon_d$, where X_d represent a (column) subset of X , let RSS_d be the residual sum of squares:

$$RSS_d = \|y - X_d\hat{\beta}_d\|^2$$

and $\hat{\sigma}_d^2 = \text{var}(\varepsilon_d)$ is the residual variance

- We need to penalize RSS to avoid overfitting
- Hence, instead of focusing on RSS, we focus on

$$AIC_d = \frac{1}{n\hat{\sigma}_d^2} (RSS_d + 2 d \hat{\sigma}_d^2)$$

- Choose model with smallest AIC_d

Model Selection: BIC

- The BIC has a heavier penalty, hence select a smaller number of variables

$$\text{BIC}_d = \frac{1}{n}(\text{RSS}_d + \log(n) d \hat{\sigma}_d^2)$$

- Choose model with smallest BIC_d

Model Selection: Cross-validation

- Leave-one-out cross-validation:
 - split the sample in two parts, one with one observation (test) and the other with the rest (train)
 - fit the model with the training set, validate the model with the test set
 - repeat the process and compute the validation error using test observations
- k -Fold Cross-Validation: similar, but randomly divide the sample in k groups, where the training set uses $k - 1$ folds and the test set uses the remaining one

$k = 5$ or $k = 10$ is typically used

- Fewer assumptions than AIC or BIC. Indeed, AIC is asymptotically similar to leave-one-out CV
- But more computational demanding

Model Selection: testing set

- Similar to cross-validation:
 - **randomly** split the sample in two parts: one with (let's say) 80% observations (training set) and the other with the rest (testing set)
 - fit the model with the training set, validate the model with the testing set
 - repeat the process many times, computing each time fitting measures (like R^2) in the testing set
- Computational demanding

Model Selection: Subset Selection

Best subset selection:

- Fit a regression for each possible combination of the p predictors: 2^p combinations
- Fix a relevant criterion (AIC, BIC, cross-validation, ...)
- Choose the combination with smallest criterion

Model Selection: Stepwise Selection

- Best subset selection is computationally infeasible for $p > 40$
- Forward Stepwise Selection:
 - Start with a model with no predictors, with the corresponding quality measure (AIC, BIC, cross-validation, ...)
 - Add one predictor each time, and select that with best fit. Compute the corresponding criterion for the best fit (AIC, BIC, cross-validation, ...)
 - Add one (remaining) predictor each time and repeat the process
 - Select the model with smallest criterion
- This is an heuristic: only $1 + p(p + 1)/2$ models considered (quadratic instead of exponential)

Model Selection: Stepwise Selection

- **Backward Stepwise Selection:**
 - Start with the full model, with the corresponding quality measure (AIC, BIC, cross-validation, ...)
 - Remove one predictor each time, and select that with best fit. Compute the corresponding criterion for the best fit (AIC, BIC, cross-validation, ...)
 - Remove one (remaining) predictor each time and repeat the process
 - Select the model with smallest criterion
- This is again an heuristic: only $1 + p(p + 1)/2$ models considered
- Not feasible when $n < p$
- **Hybrid approaches:** variables are added and removed to the model sequentially (stepwise regression)
- Same ideas can be used to other regression models like logistic regression (where the deviance plays the role of RSS)

- Regularization Methods

Regularization Methods

- How can we estimate with some accuracy β ?
- Main idea: penalize coefficient estimates, i.e. shrink them to 0
- With this framework, no need to select variables previously
- Main tools: Ridge, Lasso, Elastic Net, etc.

Ridge Regression

- **Ridge regression**: used in high dimension to mitigate overfitting (even if $p > n$)
- Also known as Tikhonov regularization: ill-conditioned problems

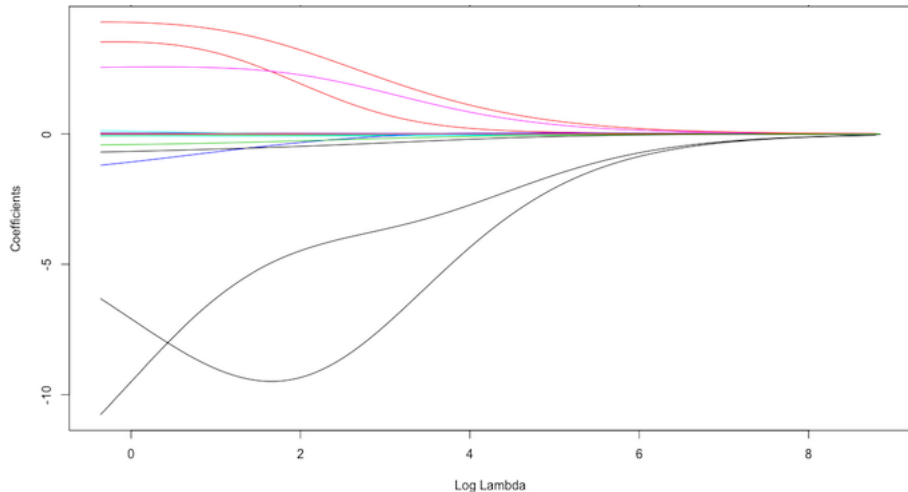
$$\text{minimize} \quad ||y - X\beta||_2^2 + \rho ||\beta||_2^2$$

where ρ is a tuning parameter, to be calibrated separately

- Explicit solution: $\hat{\beta} = (X^T X + \rho I)^{-1} X^T y$
- It adds some bias to the estimation to reduce a lot the variance: better MSE than OLS
- It is better the data matrix X is centered previously (no estimation of β_0 , we do not want to shrink it). Then, $\hat{\beta}_0 = \bar{y}$
- It is also better to standardize the data, in order to make the estimation scale-invariant
- Low computational cost, good prediction accuracy, but dense solution (no variable selection)

Ridge Regression

- Ridge regression coefficients as $\log \rho$ increases



- Note ridge tends to OLS when ρ is small, and tends to 0 when ρ is large

Ridge Regression: some theoretical results

- It is a biased estimator:

$$\text{Bias}^2(\rho) = \|E(\hat{\beta}_{RR}(\rho)) - \beta^*\|^2 = \|((X^T X + \rho I)^{-1}(X^T X) - I)\beta^*\|^2 \neq 0$$

- It has better variance than OLS:

$$\text{Var}(\rho) = \sigma^2 \text{trace} (X^T X + \rho I)^{-1}(X^T X)(X^T X + \rho I)^{-1} \leq \text{Var}(\hat{\beta}_{OLS}) = \sigma^2 \text{trace} (X^T X)^{-1}$$

- MSE decomposition: $\text{MSE}(\hat{\beta}_{RR}(\rho)) = \text{Bias}^2(\rho) + \text{Var}(\rho)$
- Theorem (Theobald, 1974): there exists $\rho > 0$ such that

$$\text{MSE}(\hat{\beta}_{RR}(\rho)) < \text{MSE}(\hat{\beta}_{OLS})$$

Ridge Regression: a Bayesian view

- Up to now, β was fixed but unknown
- **Bayesian view:** treat it as a random quantity
- Before observing X and y , we assume a prior distribution for β , $p(\beta)$
- After observing X and y , we can compute the posterior distribution for β as

$$p(\beta|X, y) \propto p(\beta) p(X, y|\beta)$$

- If $p(\beta) \sim \mathcal{N}_p(0, 1/(2\rho^2)I)$, then the mean of the posterior distribution is equivalent to ridge regression

The Lasso

- **Lasso regression**: used in high dimension to mitigate overfitting (even if $p > n$)
- L_1 regularization: **sparse solutions**

$$\text{minimize}_{\beta} \quad \frac{1}{2} \|y - X\beta\|_2^2 + \rho \|\beta\|_1$$

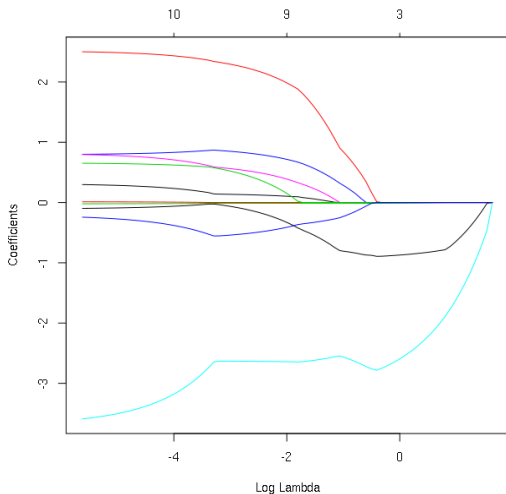
- No explicit solution: non-differentiable problem
- It adds some bias to the estimation to reduce a lot the variance: better MSE than OLS
- Attains sparsity (model selection at the same time)
- Again, it is convenient the data matrix X is centered previously (no estimation of β_0 , we do not want to shrink it). Then, $\hat{\beta}_0 = \bar{y}$
- State-of-the-art tool in **Big-Data Analytics**

The Lasso

- Many ways to estimate the lasso regression: solving a quadratic optimization problem, solving the original (but non-differentiable) problem, using the LARS, using coordinate descent, ...
- With non-prior information, ridge regression attains less variance than lasso (with similar bias)
- If real model is sparse, then lasso performs better
- Take care: if $p > n$, the Lasso selects at most n variables (which can be small)
- **Bayesian view:** if the prior for β is a Laplace distribution, then the posterior mean is equivalent to the Lasso

The Lasso

- Lasso regression coefficients as $\log \rho$ increases



- Note lasso tends to OLS when ρ is small, and tends to 0 when ρ is large, but in a sparse way

Cardinality constraints or L_0 regression

- Based on 0-norm penalty:

$$\text{minimize}_{\beta} \quad \frac{1}{2} \|y - X\beta\|_2^2 + \rho \|\beta\|_0$$

where $\|\beta\|_0$ is the number of non-zero elements in β

- $\|\cdot\|_0$ is not really a norm. . .
- This formulation is equivalent to best subset selection
- Can improve computational efficiency by using good MIP techniques (optimization)
- Not quite stable in practice

Regularization: Elastic Net

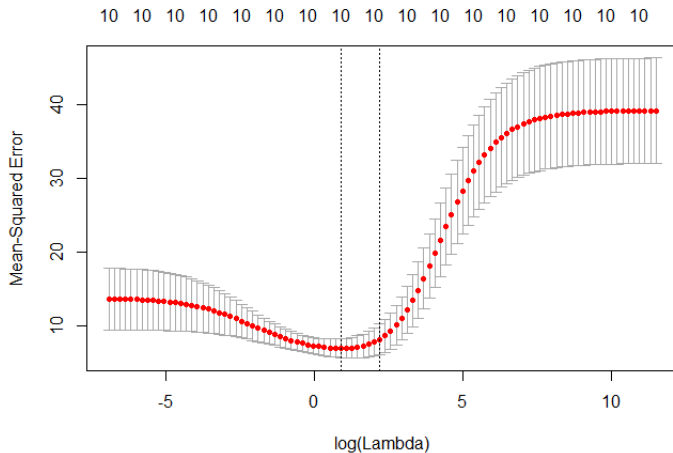
- Based on 1 and 2-norm penalties:

$$\text{minimize}_{\beta} \quad \frac{1}{2} \|y - X\beta\|_2^2 + \alpha \rho \|\beta\|_1 + \alpha(1 - \rho) \|\beta\|_2^2$$

- The 1-norm controls sparsity
- The 2-norm stabilizes the regularization path
- But we need to calibrate two parameters. . .
- Recommended packages for regularization tools: glmnet (in R), scikit.learn (in Python)

Regularization: Tuning

- How to optimize the hyper-parameters?
- Perform cross-validation and select the parameter by minimizing sum of squared residuals



- Dimension Reduction

Dimension Reduction Tools

- Principal Component Regression (**PCR**): reduce dimension in X using just a few PCs, and then perform the regression
- Partial Least Squares (**PLS**): similar in spirit to PCR but using (besides X) the information in y to reduce the dimension
Widely used in chemometrics, especially when $p \gg n$
- PLS, PCR, and Ridge regressions perform similarly
Ridge regression maybe preferred because simplicity

Principal Component Regression

- **Principal component regression:** instead of fitting a high-dimensional regression model:

$$y = X\beta + \varepsilon, \quad \text{where } \beta \in \mathcal{R}^p$$

apply PCA: $Z = XA_r$ with $r \ll p$ and fit

$$y = Z\beta + \varepsilon, \quad \text{where now } \beta \in \mathcal{R}^r$$

- Because matrix Z is orthogonal, PCR is a sum of univariate regressions
- We need to standardize previously X and y

Partial Least Squares

- **Partial Least Squares:** similar in spirit to PCR
- PLS also looks for linear combinations of the predictors, but using y in addition to X to find the optimal combination
- In other words, PCR identifies directions in an unsupervised way whereas PLS uses the response to supervise the identification of the principal components
- I.e. PLS tries to find directions that help explain both the response and the predictors
- In general, PLS has better prediction performance than PCR, but not always
- We need to standardize previously X and y

Partial Least Squares: Procedure

- After standardizing the p predictors, the weights for the first linear combination are computed through the slopes of the simple regressions between y and X_j (instead of using the eigenvectors, as in PCA)
- That implies the weights of the linear combination are proportional to the correlation between y and X_j
- Hence, PLS gives the highest weight to the most correlated variable with the response
- Subsequent directions are found by taking residuals and then repeating the above step

Statistical learning in high dimension: some conclusions

- Model selection tools are also dimension-reduction tools, but without changing the input (as in PCR or PLS). Usually they have smaller bias but larger variance
- Model selection focuses more on interpretation, less on prediction
- Regularization tools do not change the input, but change (bias) the estimator. Usually they have larger bias but smaller variance
- PCR and PLS perform roughly similar to Ridge regression, so the last one is usually preferable (much simpler and smoother)
- Lasso performs a bit worse than Ridge (in terms of prediction error) but provides model selection (better interpretation)

Optimization of hyper-parameters

Selecting the tuning parameter:

- How can we select the number of variables in step-wise regression, or the penalty parameters in regularization tools, or the number of directions in PCR or PLS?
- Cross-validation: choose a grid of those values, and compute the cross-validation error (quality measure) for each value of the grid
- Select the value with smallest cross-validation error

This is the usual way, but computationally expensive

Advanced Regression and Prediction

- 1 Advanced Estimation for Least-Squares
- 2 Extending Linear Models
- 3 Statistical Learning Tools
- 4 Machine Learning tools

4. Machine Learning tools

- Nearest Neighbors
- Support Vector Regression
- Regression trees and Random Forests
- Neural Networks

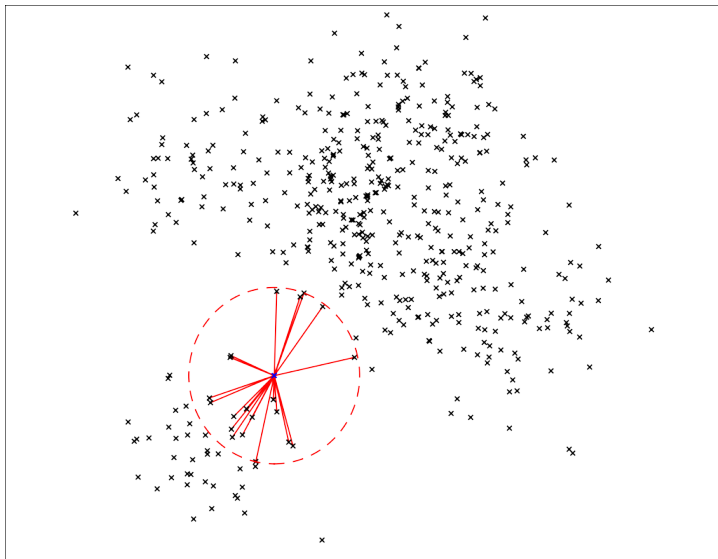
Machine Learning

- Take inputs x_1, x_2, \dots, x_k and map them to an output y
 - **Statistical learning** uses probability assumptions to find this map or function
 - **Machine learning** does not use probabilities, just the data
- To know whether the map is working correctly, we need some kind of metric to measure performance
 - **Loss function**: count number of times the model is working wrong (classification), or measure how close the prediction is to y (regression)
- The objective is to **minimize the loss**
- In regression, typically: $\text{Loss}(\hat{y}, y) = |y - \hat{y}|^h$, for some $h \geq 1$

k-Nearest Neighbors for regression

- Simple method with good performance for highly non-linear data and moderate p/n
- The **k-NN** algorithm is as follows:
 - Define a distance between observations (Euclidean, Mahalanobis, Manhattan, ...)
 - Compute the distance between the new observation to predict, x , and all the observations in the sample
 - Select the k closest observations to x and compute the average of all the responses in the k -neighborhood
 - Then, predict x with that average
- Recommended packages: FNN (in R), sklearn.neighbors module (in Python)

k-NN for regression

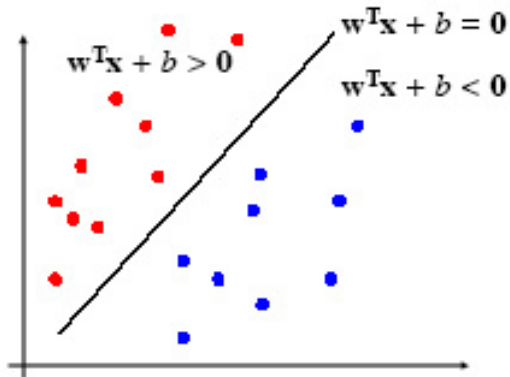


k-NN for regression

- This method is a data-mining tool: no assumptions needed
- But it cannot tell us which predictors are important
- Well-suited for highly non-linear data
- But bad performance if the dimension, p , is high
- The key element in k-NN is the selection of k (neighbors)
- Usually, cross-validation is used to select k with less prediction error (best bias-variance tradeoff)

Support Vector Regression

Remember SVM for classification:



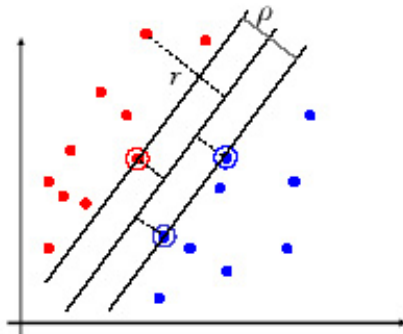
- Idea: try to maximize the gap between the two classes

Support Vector Regression

- Distance from a given point x to the linear classifier:

$$r = \frac{w^T x + b}{\|w\|}$$

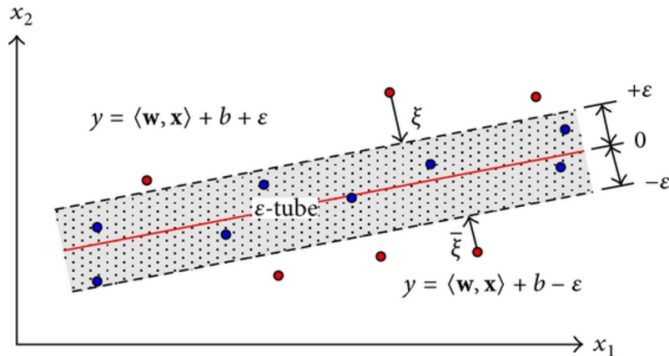
- The margin ρ associated to the classifier is the width between the two classes



- In regression, we will try to fit tube with width ε (instead of ρ)

Support Vector Regression

- The soft margin ε -insensitive loss function corresponds to a linear support-vector regression



- The **support vectors** are those points strictly outside the tube

Support Vector Regression

- In regression, the optimization problem is similar to that in classification, but not the same:

$$\begin{aligned} & \underset{w,b}{\text{minimize}} && \frac{1}{2} w^T w \\ & \text{subject to} && |y_i - (w^T x_i + b)| \leq \varepsilon \quad i = 1, \dots, n \end{aligned}$$

- Note now we do not have labels, y , but continuous variables
- Note the loss function is now the ε -insensitive loss:

$$\text{Loss}(\hat{y}, y) = |y - \hat{y}|_{\varepsilon} = \max(0, |y - \hat{y}| - \varepsilon)$$

Support Vector Regression

- With slack variables (to allow for noise):

$$\begin{aligned} \text{minimize}_{w,b,\zeta,\zeta'} \quad & \frac{1}{2}w^T w + C \sum_i (\zeta_i + \zeta'_i) \\ \text{subject to} \quad & y_i - (w^T x_i + b) \leq \varepsilon + \zeta_i \quad i = 1, \dots, n \\ & y_i - (w^T x_i + b) \geq -\varepsilon - \zeta'_i \quad i = 1, \dots, n \\ & \zeta, \zeta' \geq 0 \end{aligned}$$

- Note ε and C are hyper-parameters, usually optimized by cross-validation

Support Vector Regression: kernel-based tools

- As in classification, we can use **kernels** to deal with non-linear datasets
- The **kernel trick**: how to operate in a high-dimensional space by simply dealing with scalar products

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

- **Mercer's Theorem**: every symmetric and semi-definite positive function, K , is a kernel
- It helps introducing non-linearity into the model in a cheap way

► <https://www.youtube.com/watch?v=3liCbRZPrZA>

Support Vector Regression: kernel-based tools

- Kernel: from $(x'y)^2$ to $K(x, y)$
- Examples
 - Polynomial: $K(x, y) = (x'y + \lambda)^d$
 - Gaussian: $K(x, y) = \exp(-\sigma||x - y||^2)$
 - Much more. . .
- Note by using the kernel trick we can always compute distances in the higher space:

$$d(\phi(x_i), \phi(x_j))^2 = K(x_i, x_i) + K(x_j, x_j) - 2K(x_i, x_j)$$

- Again, we do not need ϕ , we just need the kernel

Support Vector Regression: kernel-based tools

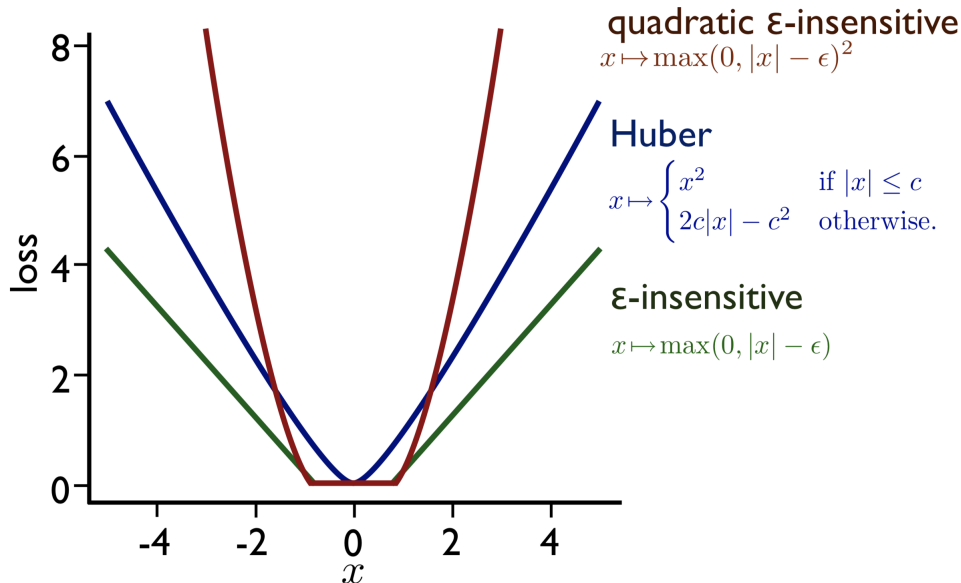
- Optimization problem:

$$\begin{aligned} \text{minimize}_{\alpha, \alpha'} \quad & \frac{1}{2} \sum_i \sum_j (\alpha_i - \alpha'_i)(\alpha_j - \alpha'_j) K(x_i, x_j) + \varepsilon \sum_i (\alpha_i + \alpha'_i) - \sum_i y_i (\alpha_i - \alpha'_i) \\ \text{subject to} \quad & \sum_i \alpha_i = \sum_i \alpha'_i, \quad i = 1, \dots, n, \\ & 0 \leq \alpha_i, \alpha'_i \leq C \quad i = 1, \dots, n \end{aligned}$$

- It can be solved by quadratic programming (convex)
- Efficient in high-dimensional spaces (columns), but inefficient with large number of observations (rows)

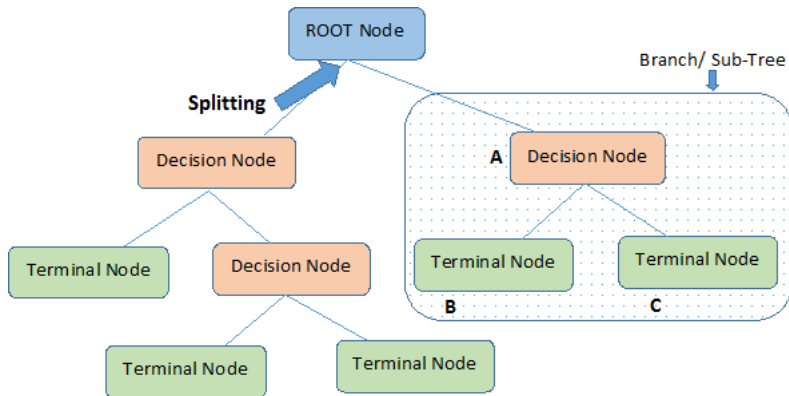
Support Vector Regression

- Other loss functions



Decision Trees

- The best visual introduction I know: [▶ Link](http://www.r2d3.us/visual-intro-to-machine-learning-part-1/)
<http://www.r2d3.us/visual-intro-to-machine-learning-part-1/>



Note:- A is parent node of B and C.

Decision trees

- In DTs, the sample is splitted first into two homogeneous and non-overlapping regions based on most relevant predictor. They are like high-dimensional boxes
- The best split is based on the variance for regression models
- Prediction is based on the average value of observations that reach the terminal node
- Regression trees, were introduced in the 1980s as part of the CART algorithm
- **Advantages:** interpretability, automatic feature selection, less data cleaning (NAs and outliers)
- **Disadvantages:** large variance or low predictive performance

Bagging

- DTs are the basis for more advanced tools: [Random Forests](#)
- DTs are fast and have small bias, but the variance is large
- [Random Forests](#) address this problem by bootstrap aggregation ([bagging](#)) for trees, reducing the variance and overfitting by using a model averaging approach
- Proposed by Leo Breiman: combine some models, instead of just one, for reducing the variance
- The reduction in the variance is related to the degree of independence (correlations) between the single models

Bagging

- Bagging uses the bootstrap to reuse the (single) training set
- For instance, if \hat{y} is our prediction model for a training set (x, y) , then the bagging prediction for B bootstrap replicates is

$$\hat{y}_{\text{Bag}} = \frac{1}{B} \sum_{b=1}^B \hat{y}_b^*(x_b^*, y_b^*)$$

where \hat{y}_b^* denotes the prediction of the b -th decision tree

- Note bagging uses equal weights and focuses on changing the training set
- **Out-of-bag error estimation:** Bagging automatically yields an estimate of the testing error using the out-of-bag observations

Out-of-bag observations

- How many values are left out of a bootstrap resample?
- Given a sample x_1, \dots, x_n and assuming all x_i are different, the probability that a particular value x_i is left out of a bootstrap resample is

$$P(x_{u_j} \neq x_i, 1 \leq j \leq n) = \left(1 - \frac{1}{n}\right)^n$$

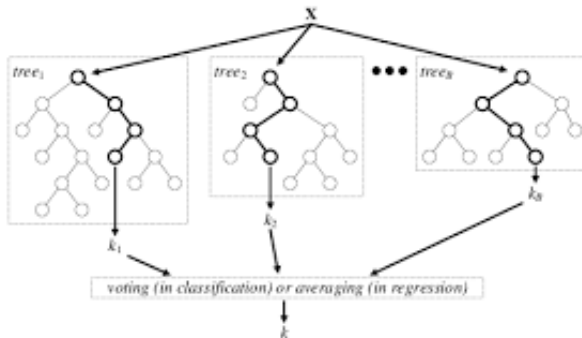
- This is because $P(x_{u_j} = x_i) = 1/n$
- When n is large, previous probability converges to $e^{-1} \simeq 0.37$
- I.e. 37% of the original sample values are usually left out
- We can use the out-of-bag observations (as a testing set) to compute the out-of-bag error

Random Forest

- Most-known bagging-learning tool
- Random Forest improves the predictive performance of DTs by averaging them
- It also reduces overfitting: for each node of the tree, consider only a random subset of predictors, $m < p$, for instance $m = \sqrt{p}$
- Approximately 2/3 of the total training data is used for growing each tree
- And the remaining 1/3 are left out and not used in the construction of each tree
- In regression, the prediction is based on the average prediction over all the trees in the forest

Random Forest: steps

- 1 Draw B bootstrap samples of size n : a decision tree for each bootstrap sample
- 2 Grow each tree, by selecting a random set of m (out of p) features at each node, and choosing the best feature to split on
- 3 Aggregate the predictions of the trees to produce the final prediction



Random Forest: final comments

- **Advantages:** more accurate, reduce variance and overfitting, easy to implement and parallelize, can easily handle categorical features and also non-linear relationships
- **Disadvantages:** less interpretation and understanding, computationally expensive, variance is not reduced enough if predictors are too correlated
- Recommended packages: randomForest (in R), sklearn.ensemble module (in Python)

Boosting

- **Boosting** = boost the performance of weak learners to attain the performance of stronger learners
- Ensemble proposed by Schapire and Freund to reduce mainly the bias (but also the variance) in a sequential way
- **Weak learner**: accuracy only a bit better than random guessing
- Boosting learns slowly (sequentially) in order to give more influence to observations that are incorrectly classified by previous learners. Hence, no bootstrap but modification of data depending on previous tree
- The two most important types of boosting algorithms in classification are the **Ada Boost** (Adaptive Boosting) algorithm (Freund, Schapire, 1997) and the **Arcing** algorithm (Breiman, 1996)

Boosting in regression: steps

- 1 Assign same weights to all observations in the training set, and train a weak learner
- 2 Change the observation weights giving more influence to observations that were incorrectly classified by previous learners, and repeat the process
- 3 At the end, choose a **weighted average** over all the weak learners to make final prediction

$$\hat{F}(x) = \sum_{b=1}^B \lambda \hat{F}_b(x),$$

where λ is the shrinkage parameter (small number)

There are mainly three hyper-parameters: d (number of nodes in the trees, usually small), B (number of trees), and λ (shrinkage)

Gradient Boosting

- Instead of using the error of each weak learner, define a **loss function** and use the **gradient descent** to minimize the loss by adding weak learners. In regression, the loss function can be the MSE
- I.e. instead of up-weighting observations that were misclassified before, identify large residuals from previous iterations
- The residuals are defined by the gradient of the loss function. And the prediction function is updated at iteration k as:

$$\hat{F}_{b+1}(x) = \hat{F}_b(x) - \alpha_b h_b(x)$$

where \hat{F} denotes the prediction function, h is a weak learner computed from gradient residuals, and α is the learning rate. For instance, $\hat{F}_0(x) = \bar{y}$

- It may overfit quickly. To avoid overfitting:
 - Tree Constraints (number of trees, tree depth, number of nodes, etc.)
 - Shrinkage (the learning rate)
 - Random sampling: stochastic gradient boosting
 - Penalized Learning: penalized values in terminal nodes, like **XGboost**

Gradient Boosting

- Advantages:

- Can handle any type of predictors

- Good predictive power

- Robustness to outliers if robust loss functions are used

- Disadvantages:

- Scalability, it is a sequential approach, hence difficult to parallelize

- Can overfit quickly, to avoid it we need to tune many hyper-parameters

Differences between Bagging and Boosting

- In boosting, each model is built on top of the previous ones whereas in bagging each model is built independently
- In bagging, the weak learners are aggregated equally whereas in boosting they are aggregated with different weights
- Bagging is a method of reducing variance while boosting focuses more on bias (but reduces also the variance)
- Bagging solves overfitting naturally while boosting needs advanced tools to deal with it
- Hence, Gradient Boosting have many hyper-parameters to tune, while Random Forest is practically tuning-free
- In general, Gradient Boosting \succ Boosting \succ Random Forests \succ Bagging \succ Decision Trees

Neural Networks

Neural Networks

The new hype: **Artificial Intelligence** and **Deep Learning**

Incredible success in applications like image recognition (diagnosis in medical imaging), natural language processing (virtual personal assistants, language translator), automated trading, autonomous cars, drug discoveries, recommendation systems, ...

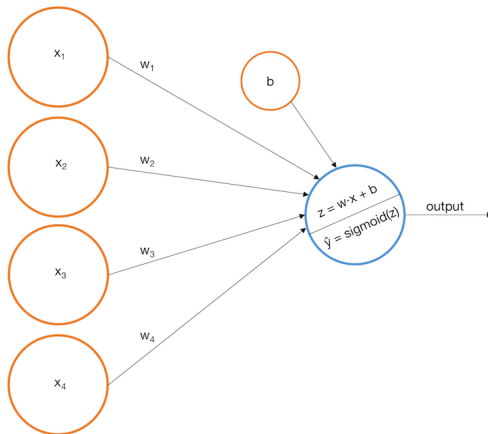
In classification, easy to understand if you know Logistic Regression:

- Artificial Neural Networks are like many logistic regressions interconnected through a network
- Each logistic regression is a **single layer perceptron** (one node in the network)
- The logistic function is now called sigmoid (a type of **activation function**)

$$\text{From } p = \frac{1}{1+\exp(-\beta_0-\beta^T x)} \text{ to sigmoid } = \frac{1}{1+\exp(-b-w^T x)}$$

In regression, the same with a linear activation function

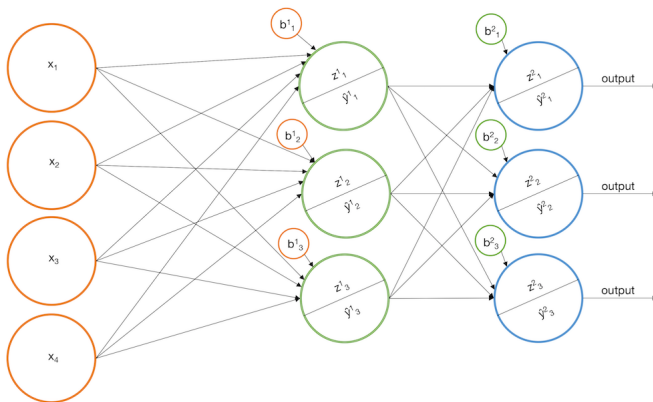
Single Layer Perceptron



Example with 4 predictors (plus constant 1) and one output (yes or no, binary classification)

Multi-Layer Perceptron

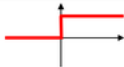
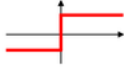
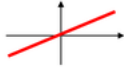


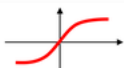
How to incorporate non-linearities and interactions? Add more layers and perceptrons (nodes)



Neural network: many layers (in the graph an input layer, a hidden layer, and an output layer with the scores of 3 classes or groups)

Alternative activation functions

The activation function, like the sigmoid, must be non-linear, otherwise the NN would be just a single layer perceptron. Its value is the input for next neurons

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer NN	

Neural Networks

Before training an ANN, we need to:

- Deal with categorical variables: one-hot-encoding (in Python language)
- Deal with missing values
- Deal with outliers
- Normalize data

In regression, use the linear (identity) activation function in output unit

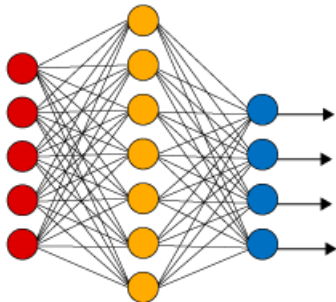
The loss function is usually the MSE (as in linear regression) or the hinge loss (as in SVM)

Deep Learning

- Can we add just more hidden layers to improve performance?

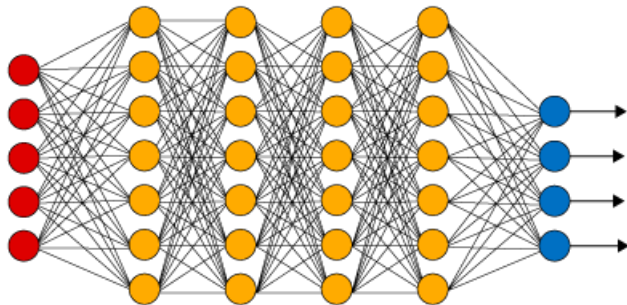
Yes! This is called **deep learning** = deep neural networks

Simple Neural Network



● Input Layer

Deep Learning Neural Network



● Hidden Layer

● Output Layer

Deep Learning

- Take care: the [Universal Approximation Theorem](#) says you can approximate any continuous function using a NN with just a single hidden layer
- Hence, in theory no need of more than one hidden layer. But DL fits a problem faster in practice
- DL estimates the weights in the NN by minimizing a differentiable loss function through the (mini-batch) gradient descent method, computing the derivatives using back-propagation (chain's rule). In practice, millions of parameters to estimate in a highly non-linear function
- In general, larger networks tend to predict better than smaller networks, but overfitting should be avoided through regularization
- Use TensorFlow in practice (an open-source DL flexible framework) to select network topology, optimization algorithm, activation function, etc.

Advanced Regression: final remarks

No-Free-Lunch-Theorem: there is no best algorithm that works best in all cases

- Linear regression is a very simple tool: no bias but large variance in high dimension. It provides good interpretation
- Decision trees are even simpler than LMs and provide simpler interpretation, but the prediction accuracy is worse
- Model selection tools add a bit bias but variance is still high
- Variance can be decreased a lot by regularization tools, at the price of increasing also the bias
- Elastic net is a good choice, but expensive to optimize the hyper-parameters

Advanced Regression: final remarks

No-Free-Lunch-Theorem: there is no best algorithm that works best in all cases

- SVM is especially useful when non-linear kernels are used (non-linear problems). Not suffer from collinearity but hard to interpret. Scales well with p but bad with n
- Random Forests: good performance in practice, good for distributing computing, deals well with interactions and non-linearities. Provides variable importance and scales well with n
- Gradient Boosting: tends to perform better than RF but harder to tune parameters. Prone to overfitting. Not good for parallel computing, but performs well in practice when n is large
- Neural Networks: nowadays, one of the most successful approaches, specially for non-structured data (non-tabular). They can handle non-linear data and high dimension, but computationally expensive and difficult to avoid overfitting. No need of feature engineering. A completely black-box tool but many open source implementations. Difficult to tune lots of hyper-parameters, non-linear optimization problem. Performs very well in practice when n is huge

The end... Attained objectives

- Relax some of the assumptions in classical linear regression (normality, loss functions, etc.)
- Estimate efficiently the parameters of the models
- Deal with the curse of dimensionality in high-dimensional problems
- Know the main tools in advanced predictive modeling, including machine learning
- Handle the R language for advanced regression (including caret package)

"Far better an approximate answer to the right question, which is often vague, than an exact answer to the wrong question, which can always be made precise."

John W. Tukey

The end...

Thanks for your attention!!

Javier Nogales
www.est.uc3m.es/Nogales
@fjnogales