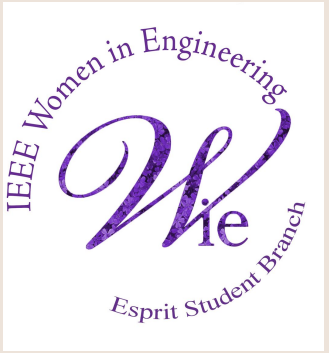
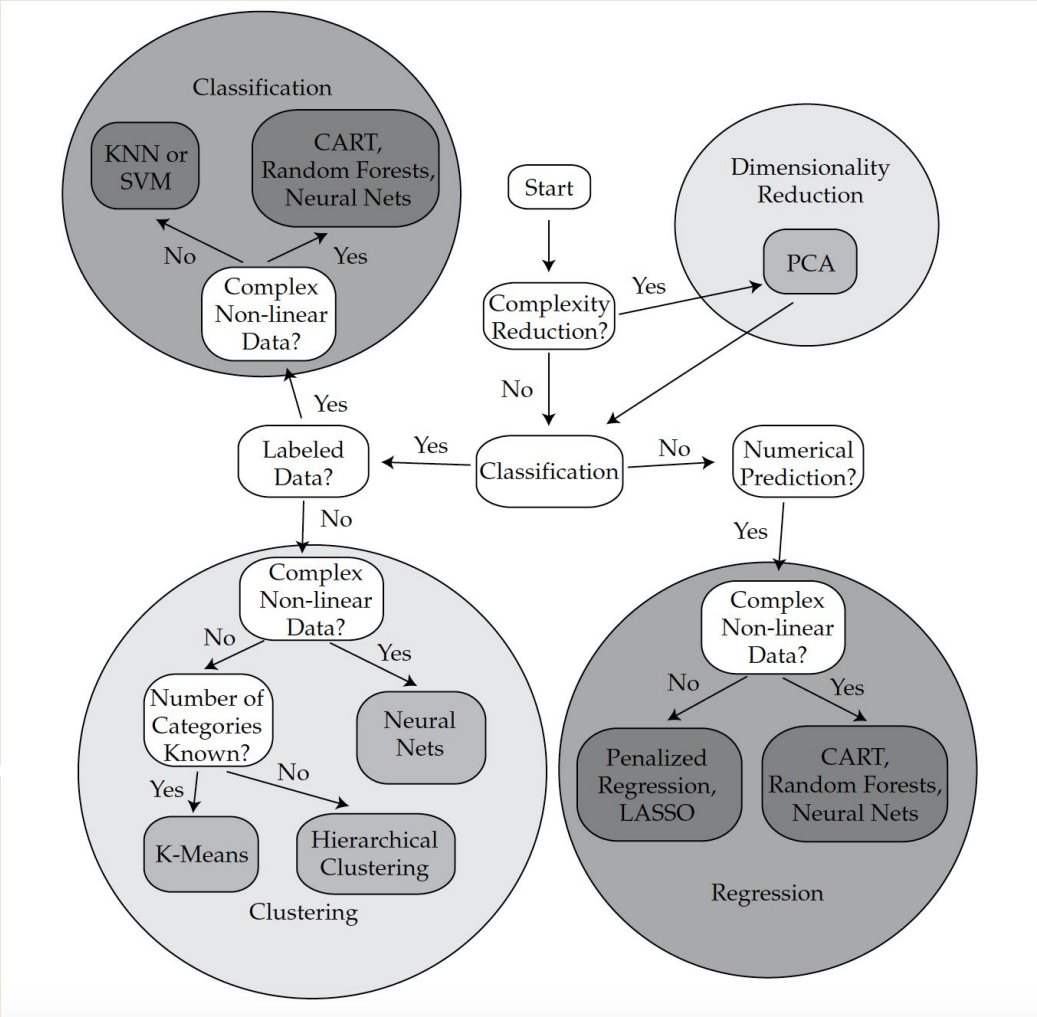


Machine Learning
A Hands-On Python Workshop

Vivek
Research Assistant
SPECS (Synthetic, Perceptive, Emotive and Cognitive Systems)
Instituto de Bioenginyeria de Cataluña (IBEC)
Av. Eduard Maristany, 16 08019 Barcelona
Campus Diagonal Besòs, Edifici C
Twitter | Linkedin


Start
Complexity Reduction?
Yes
No
Dimensionality Reduction
PCA
Classification
Labeled Data?
Yes
No
Numerical Prediction?
Yes
No
Classification
Complex Non-linear Data?
No
Yes
KNN or SVM
CART, Random Forests, Neural Nets
Complex Non-linear Data?
Yes
No
Number of Categories Known?
Yes
No
Neural Nets
K-Means
Hierarchical Clustering
Clustering
Complex Non-linear Data?
No
Yes
Penalized Regression, LASSO
CART, Random Forests, Neural Nets
Regression

Mean Squared Error

The **mean squared error** (MSE) tells you how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the "errors") and squaring them. The squaring is necessary to remove any negative signs. It also gives more weight to larger differences. It's called the mean squared error as you're finding the average of a set of errors. The lower the MSE, the better the forecast.

$$\hat{y}_i = \theta x_i$$

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Least-squares optimization

computing MSE at various values of $\hat{\theta}$ quickly got us to a good estimate, it still relied on evaluating the MSE value across a grid of hand-specified values. If we didn't pick a good range to begin with, or with enough granularity, we might miss the best possible estimator. Let's go one step further, and instead of finding the minimum MSE from a set of candidate estimates, let's solve for it analytically.

We can do this by minimizing the cost function. Mean squared error is a convex objective function, therefore we can compute its minimum using calculus.

$$\hat{\theta} = \frac{\mathbf{x}^\top \mathbf{y}}{\mathbf{x}^\top \mathbf{x}}$$

Where **x** and **y** are vectors of data points.

Probabilistic Models

Now that we have a model of our noise component ϵ as random variable, how do we incorporate this back into our original linear model from before? Consider again our simplified model $y=\theta x+\epsilon$ where the noise has zero mean and unit variance $\epsilon \sim N(0,1)$. We can now also treat y as a random variable drawn from a Gaussian distribution where $\mu=\theta x$ and $\sigma 2=1$:

$$y \sim \mathcal{N}(\theta x, 1)$$

which is to say that the probability of observing y given x and parameter θ is

$$p(y|x,\theta) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y-\theta x)^2}$$

Note that in this and the following sections, we will focus on a single data point (a single pairing of x and y). We have dropped the subscript i just for simplicity (that is, we use x for a single data point, instead of xi).

Maximum likelihood estimation

Maximum likelihood estimation is a method that determines values for the parameters of a model. The parameter values are found such that they maximise the likelihood that the process described by the model produced the data that were actually observed.

$$\mathcal{L}(\theta|x,y) = p(y|x,\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y-\theta x)^2}$$

Bootstrapping

Bootstrapping is a widely applicable method to assess confidence/uncertainty about estimated parameters, it was originally proposed by Bradley Efron. The idea is to generate many new synthetic datasets from the initial true dataset by randomly sampling from it, then finding estimators for each one of these new datasets, and finally looking at the distribution of all these estimators to quantify our confidence.

Note that each new resampled datasets will be the same size as our original one, with the new data points sampled with replacement i.e. we can repeat the same data point multiple times. Also note that in practice we need a lot of resampled datasets.

To explore this idea, we will start again with our noisy samples along the line $y_i = 1.2x_i + \epsilon_i$, but this time only use half the data points as last time (15 instead of 30).

Confidence Interval

A confidence interval displays the probability that a parameter will fall between a pair of values around the mean.
Confidence intervals measure the degree of uncertainty or certainty in a sampling method.
They are most often constructed using confidence levels of 95% or 99%.

Multiple linear regression (MLR)

- Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable.
- Multiple regression is an extension of linear (OLS) regression that uses just one explanatory variable.

Polynomial Regression

polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial in x. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y |x). Although polynomial regression fits a nonlinear model to the data, as a statistical estimation problem it is linear, in the sense that the regression function E(y | x) is linear in the unknown parameters that are estimated from the data. For this reason, polynomial regression is considered to be a special case of multiple linear regression.

Design matrix for polynomial regression

Now we have the basic idea of polynomial regression and some noisy data, let's begin! The key difference between fitting a linear regression model and a polynomial regression model lies in how we structure the input variables.

Let's go back to one feature for each data point. For linear regression, we used $\mathbf{X} = \mathbf{x}$ as the input data, where $\mathbf{x}$ is a vector where each element is the input for a single data point. To add a constant bias (a y-intercept in a 2-D plot), we use $\mathbf{X} = \left[\mathbf{1}, \mathbf{x}\right]$, where $\mathbf{1}$ is a column of ones. When fitting, we learn a weight for each column of this matrix. So we learn a weight that multiples with column 1 - in this case that column is all ones so we gain the bias parameter ($+\theta_0$).

This matrix $\mathbf{X}$ that we use for our inputs is known as a **design matrix**. We want to create our design matrix so we learn weights for $\mathbf{x}^2$, $\mathbf{x}^3$, etc. Thus, we want to build our design matrix X for polynomial regression of order k as:

$$\mathbf{X} = \left[\mathbf{1}, \mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^k\right],$$

where $\mathbf{1}$ is the vector the same length as $\mathbf{x}$ consisting of of all ones, and $\mathbf{x}^p$ is the vector $\mathbf{x}$ with all elements raised to the power p. Note that $\mathbf{1} = \mathbf{x}^0$ and $\mathbf{x}^1 = \mathbf{x}$.

If we have inputs with more than one feature, we can use a similar design matrix but include all features raised to each power. Imagine that we have two features per data point: $\mathbf{x}_m$ is a vector of one feature per data point and $\mathbf{x}_n$ is another. Our design matrix for a polynomial regression would be:

$$\mathbf{X} = \left[\mathbf{1}, \mathbf{x}_m^1, \mathbf{x}_n^1, \mathbf{x}_m^2, \mathbf{x}_n^2 \ldots, \mathbf{x}_m^k, \mathbf{x}_n^k\right],$$

References

- A special thanks to Neuromatch Academy for providing the codes and explanatory colaboratory notebooks (https://compneuro.neuromatch.io/tutorials/intro.html).

Time for Hands-On

Python Notebook