Artificial Intelligence

Foundations of Machine Learning III

Prof. Ignacio Olmeda

THE IMPORTANCE OF DATA PRE-PROCESSING AND FEATURE ENGINEERING

- As we have seen, ML algorithms use data to develop predictive models.

- Before using such algorithms, the developer has to follow the ETL process that consist on *E*xtracting information from reliable sources, *T*ransform such information so that it is usable and *L*oad the information in the system that we will use to feed the algorithm.

- Many researchers suggest that the ELT process consumes more than 80% of the developing time in a ML project.

- Regarding *Extraction* and *Loading*, there is a number of solutions that can be employed to manage the massive amount of data that characterize ML applications, examples of such commercial solutions/systems/technologies are Hadoop, Hive, Spark, AWS, etc.

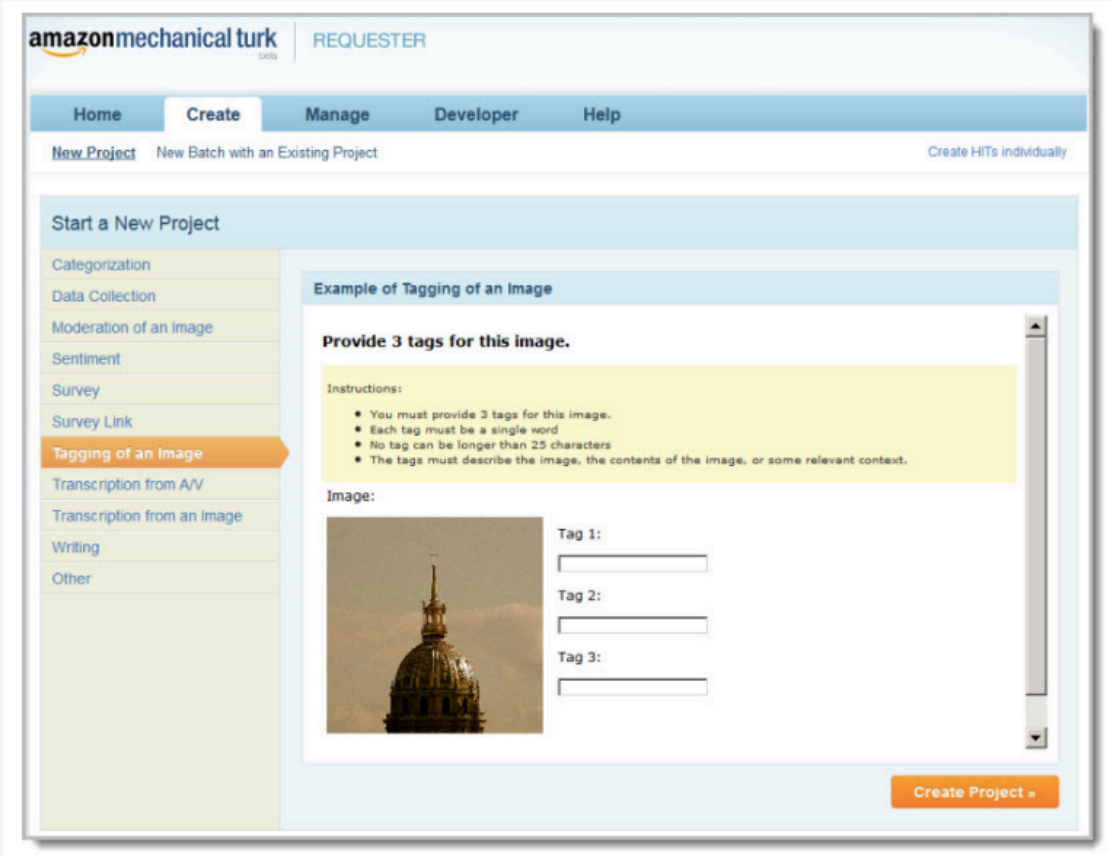
- Regarding data *Transformation* or processing, the task is highly application-dependent and requires a considerable effort of the developer.

- ML algorithms can, theoretically, find hidden patterns in *raw* data, this means that, in principle, it is not necessary to transform data to make it accesible.

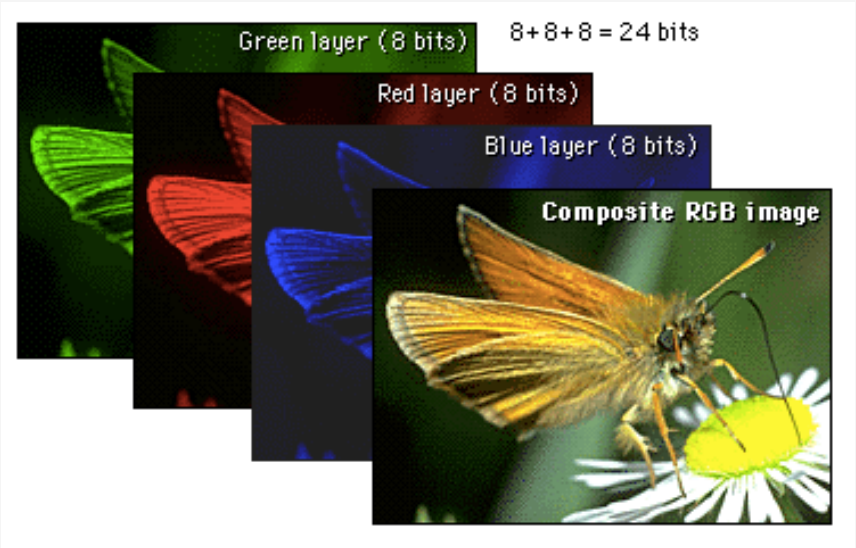
- Nevertheless, as humans, machine learning algorithms may process more efficiently data with a particular representation, representation or pre-processing provides some "hint" to the algorithm facilitating in most of the cases the learning process.

- Note also that some features of the instances we may want to employ may not be numerical (e.g. labels) and so they can not be treated analytically in an efficient manner.

- For example, think on the the case of Amazon's *Mechanical Turk* tasks:



- Finally, many algorithms require that the dimensionality of the input data is kept constant, for example, the same number of pixels in an image, so that raw data may need some sort of transformation.



- For these and other reasons, data pre-processing is an essential part of building ML models and it strongly conditions the result of the whole process.

Data quality is essential in ML since data is the raw material to build the models.

- As we mentioned, the attributes in ML are called *features*, and they can be interpreted as propeties of the object (measurable or not) .

- Usually it is required that features are independent (no relationship among them) and have discriminative power (so that they add something to the problem at hand) but even to detect such simple propertis in most of the cases is unfeasible.

- For example, features may look pairwise independent but there can a a latent variable that may affect two aparently independent variables.

- Also, some variable may seem not to have discriminative power but taken together with other variable its power may increase radically, e.g.

$$Y = X_1 X_2$$

- *Feature engineering* is the area of ML that tries to obtain the best possible results from a predictive model by transforming raw data into features that better represent the underlying problem to the predictive models.

- Note that features can be properties directly observable (e.g. Birth date), must need to be estimated using a model or aparatus (temperature) or may be subjective.

DATA INTUITION

- To detect whether data at hand is consistent, the first step is to have an intuitive view, visualization and descriptive statistics allow to understand the nature of data.

- Regarding statistics, some useful statistical measures are the *mean*, the *median* (the central point of the distribution) and the *mode* (the most frequent value).

4	8	3	5	6	9	2	3	1	mean	4.6
									median	4
									mode	3

- It is also relevant to evaluate some measures of spread such as the *range* (the distance between the maximum and minimum value), or the *variance*.

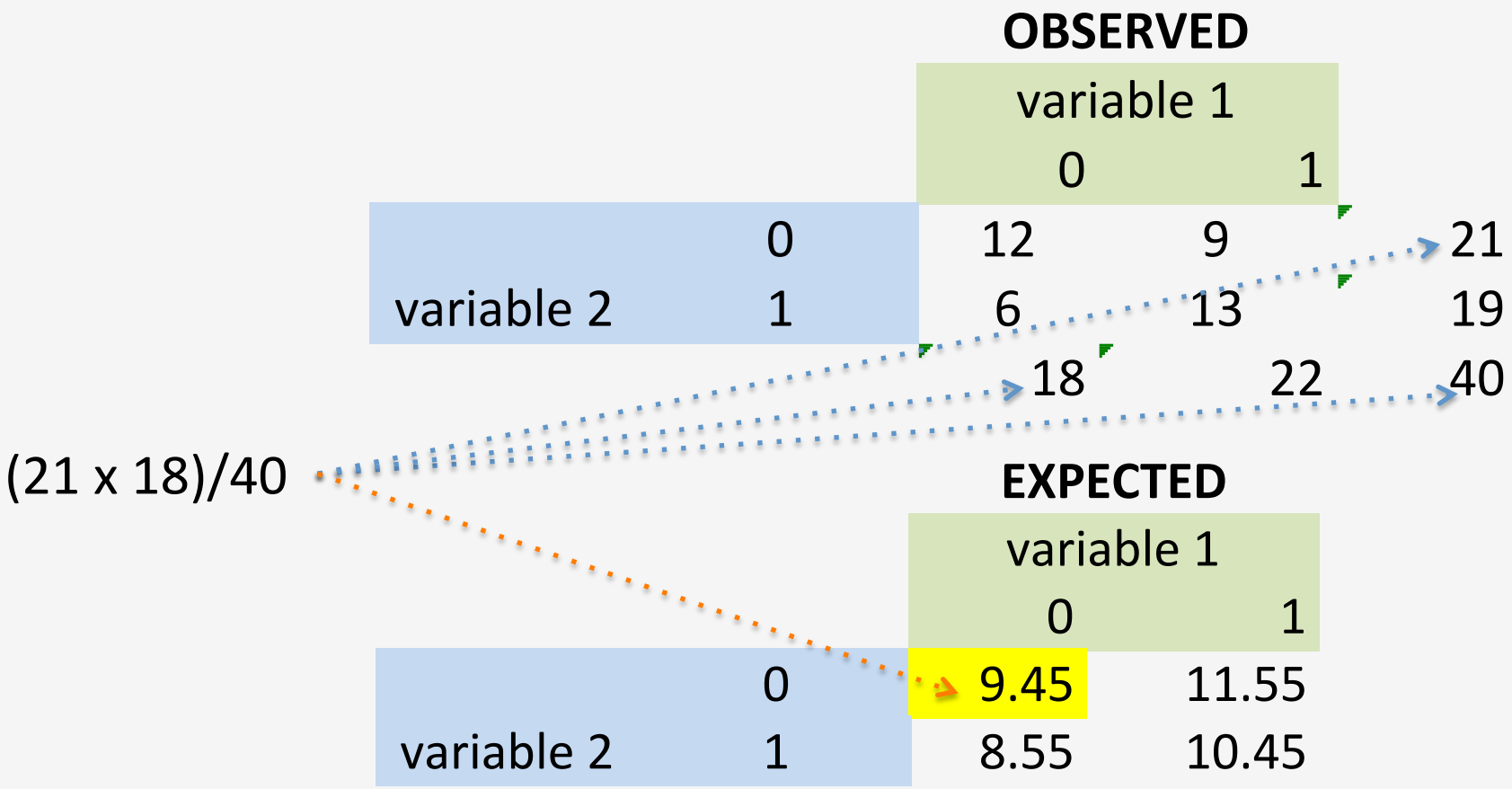
It can be also useful to analyse some measures of co-movement such as the *correlation*.

$$\rho_{X_1X_2} = \frac{\sum_{i=1}^{n}(X_1^i - \bar{X}_1)(X_2^i - \bar{X}_2)}{\sqrt{\sum_{i=1}^{n}(X_1^i - \bar{X}_1)^2(X_2^i - \bar{X}_2)^2}} \in [-1,1]$$

Another possibility is to use the *Chi-square test* for independence between the variables,

$$test = \frac{\sum_{i,j}^{n}(observed_{i,j} - expected_{i,j})^2}{expected_{i,j}} \approx \chi_k^2, \quad k = (r-1)*(c-1)$$

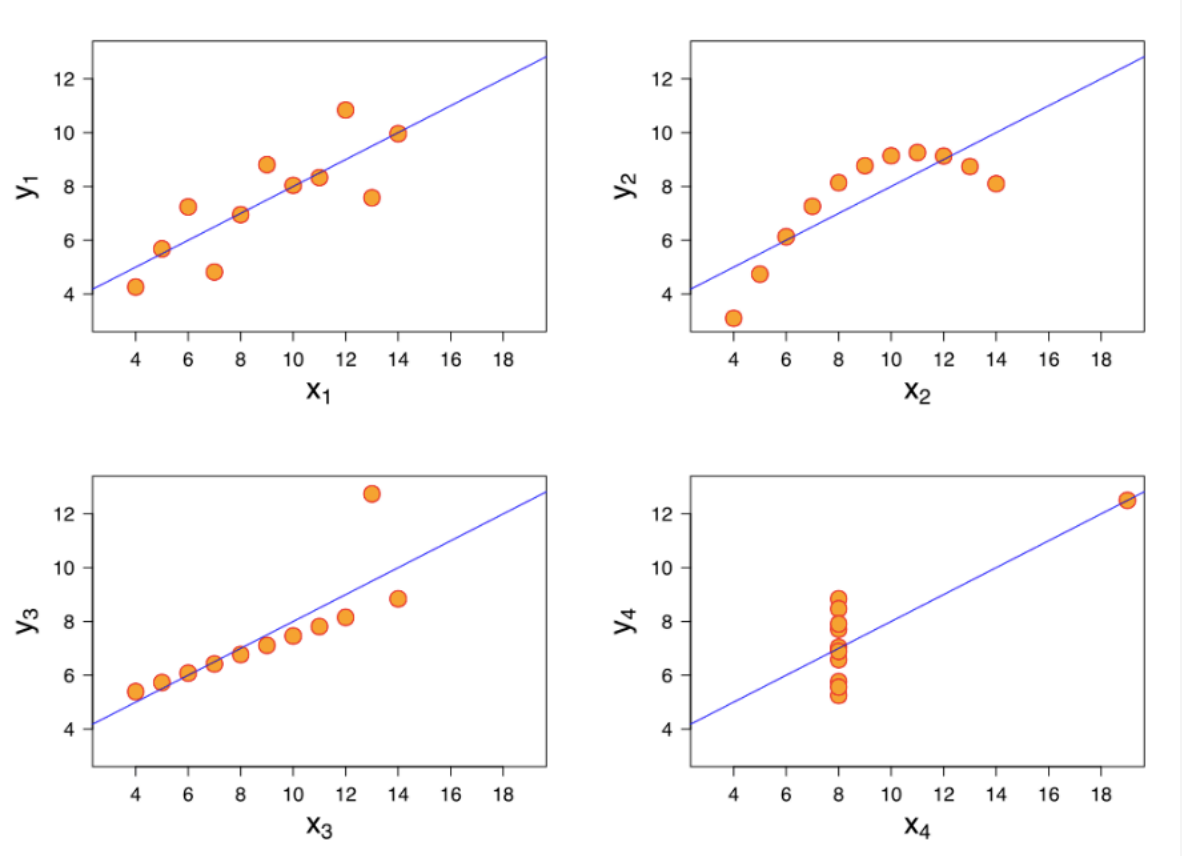
$$H_0 : Independence$$



	0.69	0.56	
chi-sq	0.76	0.62	
			2.63
critical (df=1)			3.84

- Nevertheless such statistics provide, in some cases, poor descripiptions of the data.

- A paramount example is the *Ascombe quartet*, four data sets with nearly identical simple descriptive statistics but which are very different when plotted

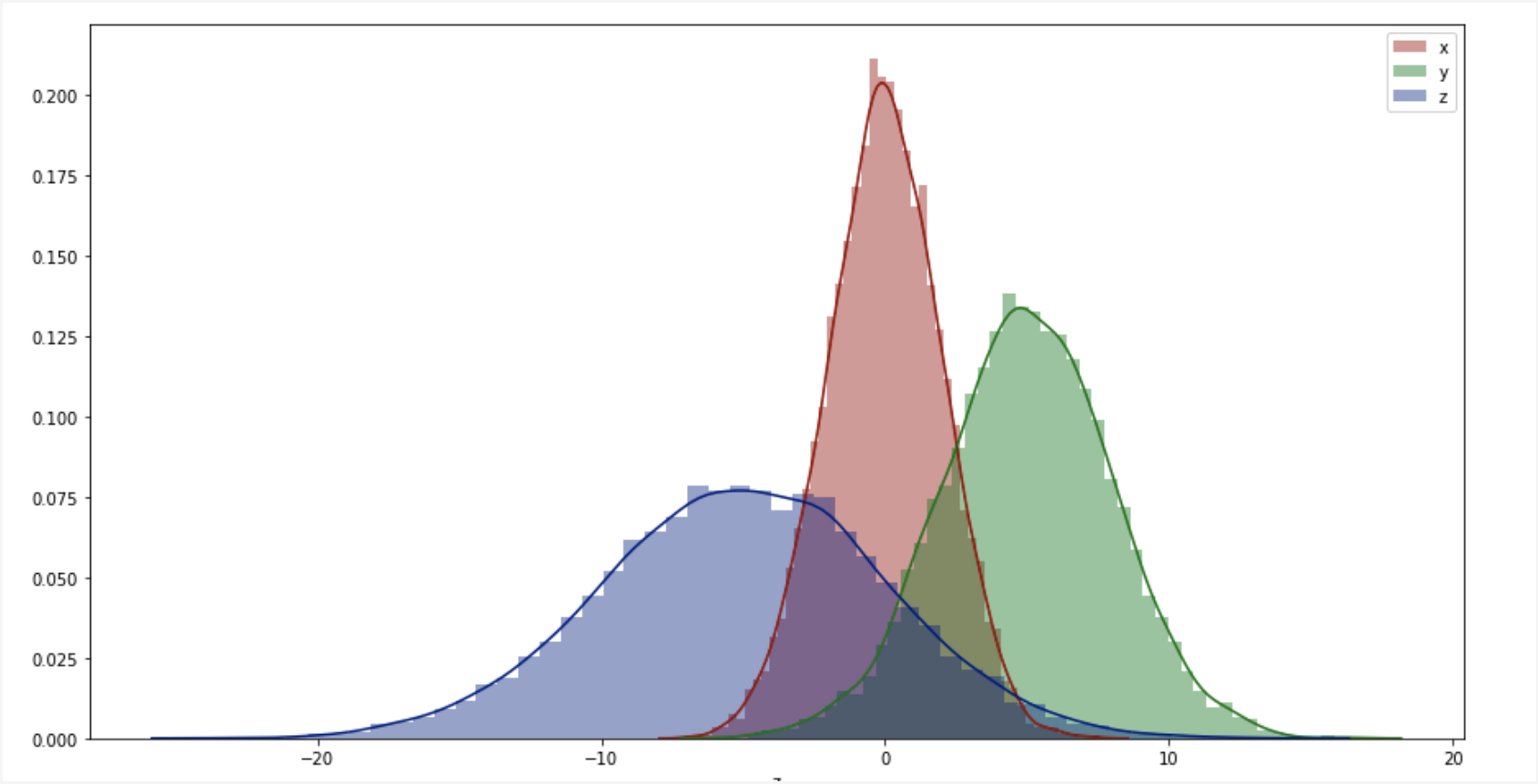


- For this reason, when possible, it is generally useful to have an intuitive view of the data in the form of some kind of graphic.

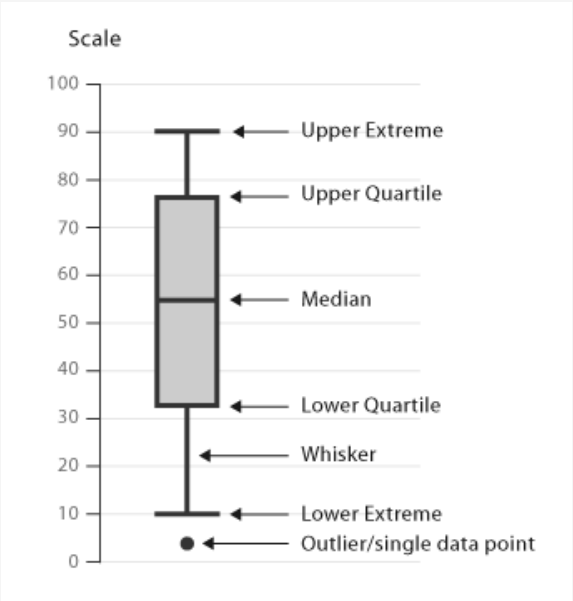
- The problem is that relationships between variables could be highly nonlinear, affected by noise and also have a high dimensionality making, in most of the cases, very difficult to produce such plots.

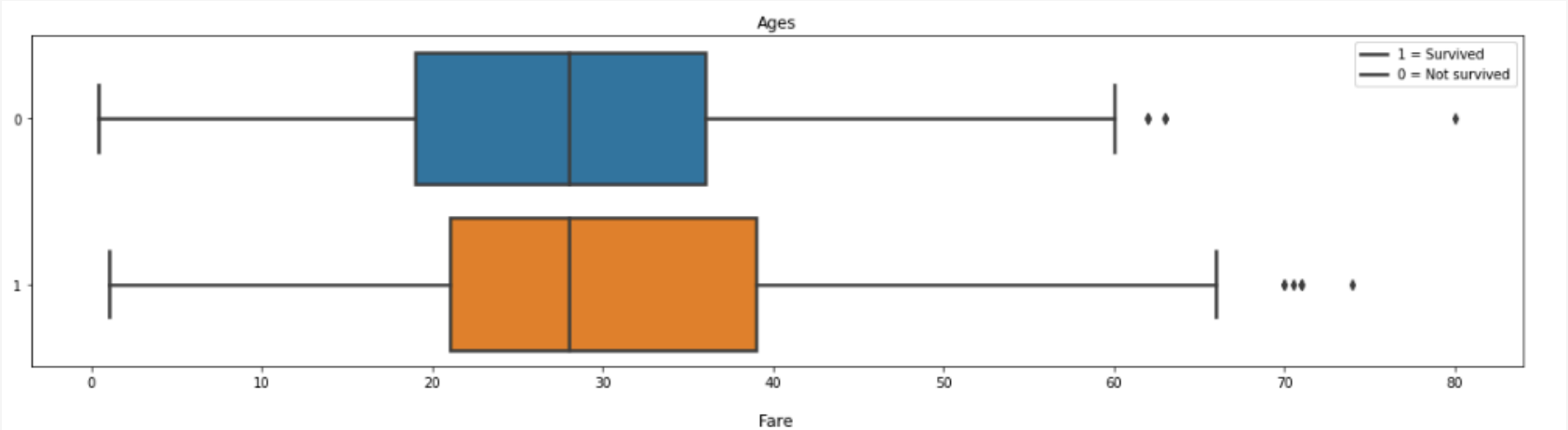
- The range of possibilities that visualization offers is enormous, it is a very active area of research which is producing many interesting results.

- Among the plots that can be used to have an intuitive view of the data probably the most bvious is the *histogram* which show show spread and peaky is the distribution and whether it is skewed.

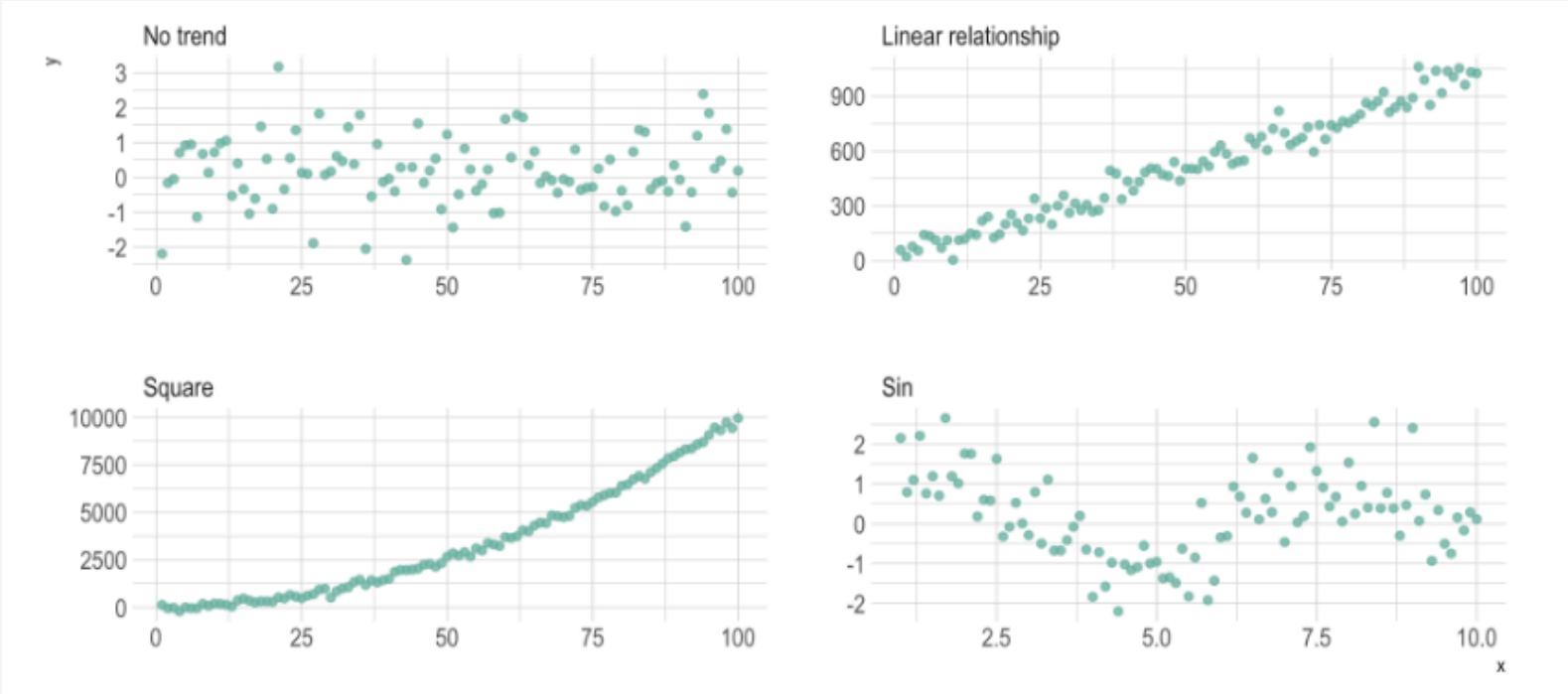


- The *whisker box plot* is also very convenient to understand the variability between quartiles of the distribution.



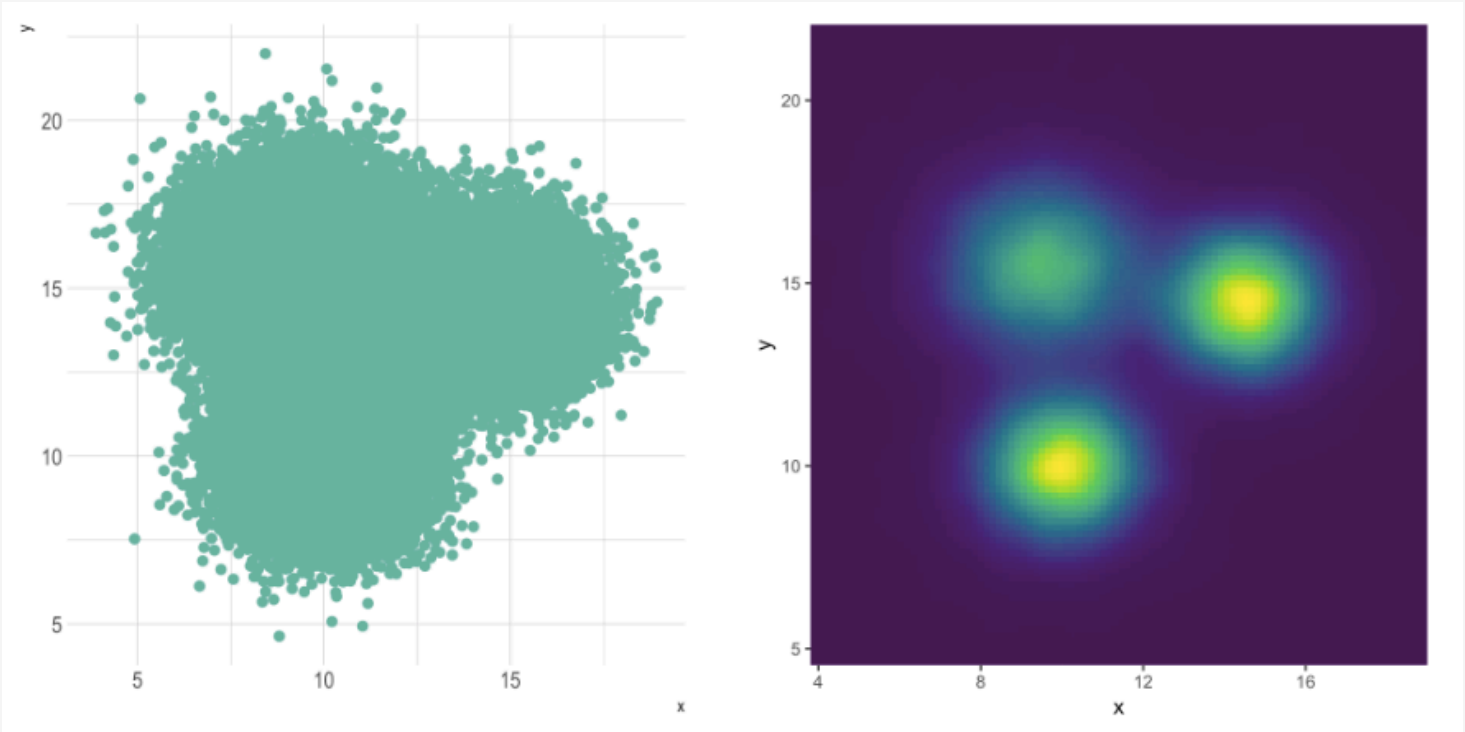


- *Scatter plots* give an intuition on the pairwise relationship between the variables at hand:

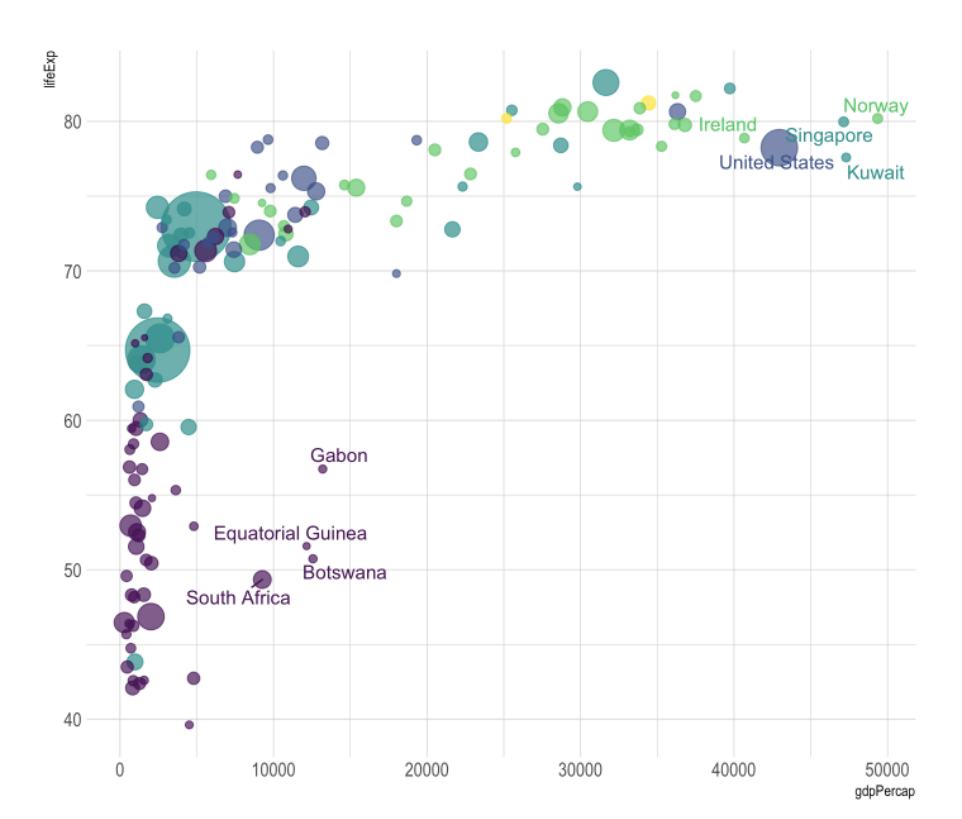


- One problem of scatterplots is that they might be too dense, making it impossible to detect any pattern in the data.

- An alternative is to employ *2D density plots* which count the number of observations within a particular area of the 2D space.



- *Bubble plots* extend two dimensions to a third by showing in a cartesian axes the third one using the size of the dots.



- Data visualization and descriptive statistics are poweful tools that may be helpful in understanding the data at hand and diagnosing problems with it.

DATA PRE-PROCESSING AND FEATURE ENGINEERING

- An intuitive view of data is not enough to build successful ML applications.
- There are a number of issues that need to be addressed so that we can be sure that data will ease the process of model construction, in particular we have to consider problems related to:
  - Missing Data
  - Transforming Data
  - Data Representation
  - Incomplete Data
  - Imbalanced Data
  - Data Reduction

- *Missing data* happens when one or more features of some particular example are not available.

- This can be due for several reasons and each one of them needs to be properly addressed:

  - The data exists but it is unknown: for example, for an individual we might not know her birthday

  - We do not know whether the data exist or not: for example in "floor" we do not know whether the individual resides in a flat or in a house, so we do not know if the attribute is appliable

- Missing data has different consequences depending on the algorithm applied, in some cases the algorithm may perfectly handle missing data (some *decision trees*, for example) while in some other cases the data must be completed so that the algorithm works (linear models, for example).

- Missing data are commonly referred as:
  - *Missing Completely at Random* (MCAR): Missed data are completely random, there is no relationship between whether a data point is missing and any values in the data set, missing or observed.
  - *Missing at Random* (MAR): the cause of the missing data is unrelated to the missing values but may be related to the observed values of other variables
  - *Missing not at Random* (MNAR): Missing value depends on the hypothetical value of the variable .

- To deal with missing data one may employ a number of alternatives.
- The most drastic one is to completely eliminate the example that has one or more uncomplete features (*listwise deletion*).
- This alternative may be reasonable when there is plenty of data but in most of the cases it is not feasible.
- Moreover, the loss of some characteristics may reveal problems in data acquisition and the elimination of such examples may induce a bias in data.

	X				
	ATTRIBUTE 1	ATTRIBUTE 2	ATTRIBUTE 3	ATTRIBUTE 4	ATTRIBUTE 5
pattern A	1	4	3	0	2
pattern B	4	6	na	1	2
pattern C	7	6	na	1	2
pattern D	1	5	3	6	8
pattern E	8	3	7	1	2
pattern F	1	6	na	2	2

- Another alternative is to eliminate the feature that has too many missing values, this is called *dropping features*:

- Note that this alternative it is also very dramatic since the feature eliminated may have a hgh predictive value on the dependent variable.

	X				
	ATTRIBUTE 1	ATTRIBUTE 2	ATTRIBUTE 3	ATTRIBUTE 4	ATTRIBUTE 5
pattern A	1	4	3	0	2
pattern B	4	6	na	1	2
pattern C	7	6	na	1	2
pattern D	1	5	3	6	8
pattern E	8	3	7	1	2
pattern F	1	6	na	2	2

- The third possibility is to replace the value of the missing attribute by some estimate of the "expected" value of the variable, this procedure is called *data imputation*.

- "Expected" has, in general, no clear interpretation, for example we could employ the mean, the median or the mode or even the value that is more common for similar patterns.

	X				
	ATTRIBUTE 1	ATTRIBUTE 2	ATTRIBUTE 3	ATTRIBUTE 4	ATTRIBUTE 5
pattern A	1	4	3	0	2
pattern B	4	4	na	1	2
pattern C	5	4	2	1	2
pattern D	1	5	3	6	8
pattern E	8	3	7	1	2
pattern F	1	6	4	2	2

Mode	3
Mean	3.8
Closest (pattern C)	2

- More sophisticated methods can be also applied, for example, ome may try to adjust a linear model using the missing characteristic as the independet variable and use examples taht are complete to perform the regresion and then yo use the estimated coefficients to forecats the value of the missing variable:

$$X^k = \alpha + \hat{\beta}_1 X_1 + ... + \hat{\beta}_{k-1} X_{k-1} + \hat{\beta}_{k+1} X_{k+1} + ... + \hat{\beta}_n X_n$$

- In the case of time series, there is a diversity of methods that can be employed, the simplest ones are simply using the last data point:

$$X^n_{t+1} = X^n_t$$

- Or an interpolation, e.g.

$$X^n_t = \frac{X^n_{t-1} + X^n_{t+1}}{2}$$

- Or a more sophisticated method such as estimating a model to predict the unknown value using the existig ones.

- *Transforming data* refers to the process of modifying data to make it more adequate for our machine learning algorithms.

- It involves a number of possible modifications of raw data or even the elimination of examples or features that may distort the process of finding hidden patterns in the data.

- This process is also called *cleaning data*.

- In Statistics it is very common to employ variables which follow a normal distribution $N(\mu,\sigma)$ and to transform it to a *standardized* N(0,1) distribution:

$$X' = \frac{X - \mu}{\sigma} \approx N(0,1)$$

- This is done because the statistical properties of the standardized normal distribution are very well known and one may calculate statistics with a known distribution.

- Such standardization (and others) is also commonly applied in Machine Learning, even though the purposes are not inferential but more of a practical nature.

- Since variables may be quite different expressed in very different scales this might affect the learning process, note that even in trivial models such a multinomial model

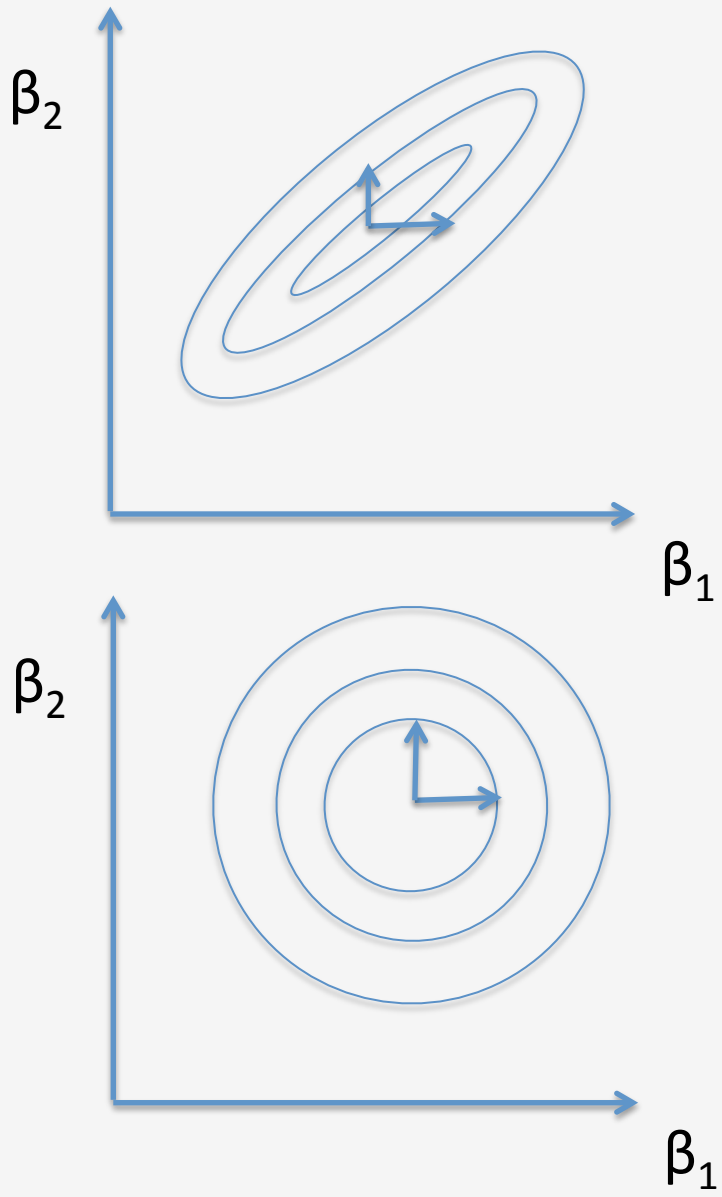
$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n$$

- The parameters β may have very different scale, so that e.g. a randomization on the interval [0,1] for β_1 could be innapropriate for β_n if it moves on e.g. [100.000,200.000].

- Another possibility is to employ

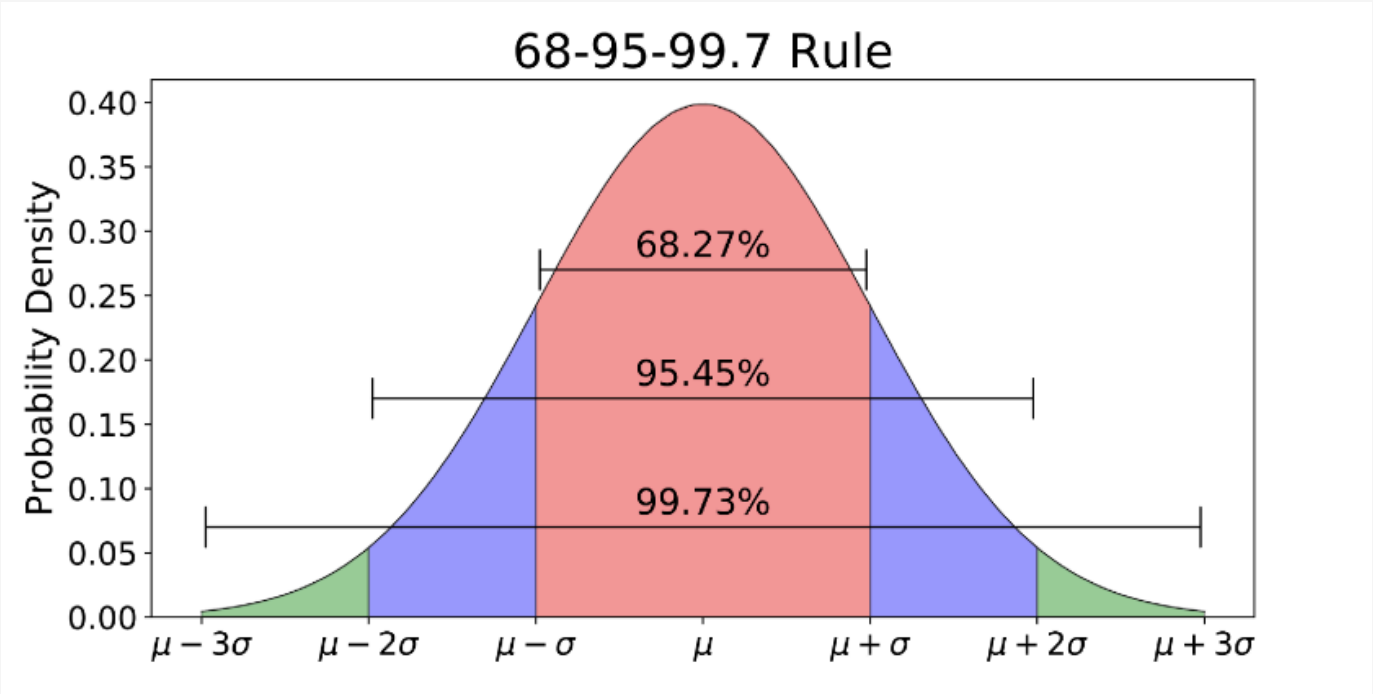
$$X' = \frac{X - \min(X)}{\max(X) - \min(X)} \in [0,1]$$

- The next figure provides some intuition of the normalization process



- For some ML algorithms it is particularly relevant to check for abnormal observations, it is almost impossible to define what does it mean "abnormal": observations may appear different, may be different or, simply, may be caused by errors collecting the data ("fat finger" problems).

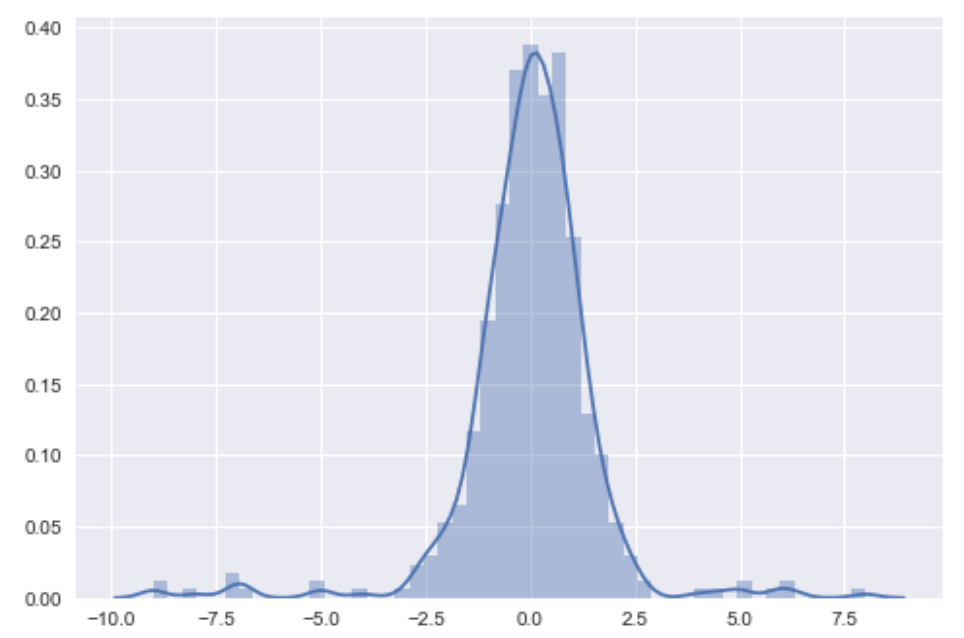
- One property of the normal distribution is that between the mean and 2 times the standard deviation we can find 95% of the observations

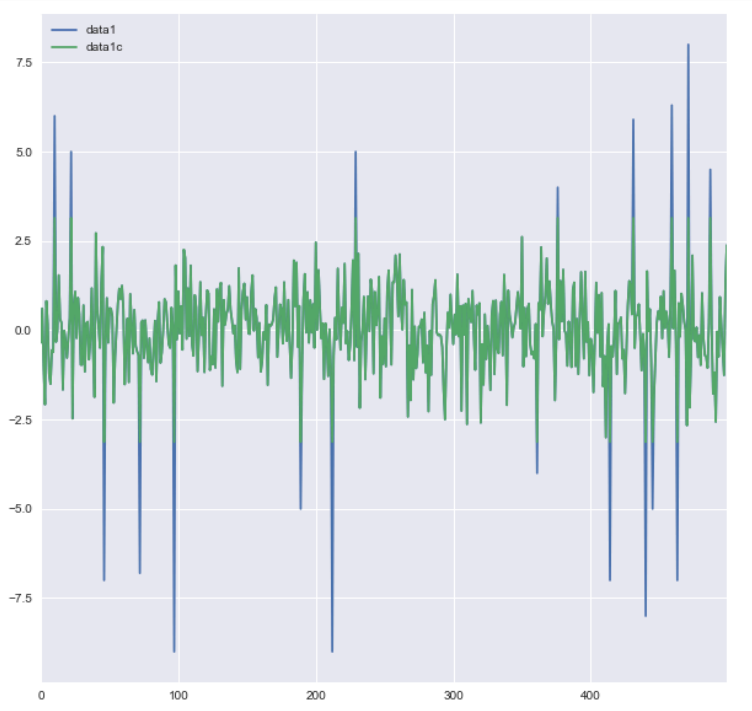


- An *outlier* is "an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different statistical mechanism" (Hawkins, 1980).

- One possibility is to consider that an abnormal oservation is one that is "extremelly rare" (e.g. 5%) and then to truncate variables which are far apart from the "normal" behavior, e.g.

$$x = \begin{cases} \min(x, \mu + 2\sigma), \text{if } x > 0 \\ \max(x, \mu - 2\sigma), \text{if } x \leq 0 \end{cases}$$





outlier.ipynb

- Another possibility is simply to calculate the *z-score* of the observations:

$$z - score = \frac{X - \mu}{\sigma}$$

- And then to eliminate observations which have e.g. Z-score higher than 3, 4…

- Finally one may use more sophisticated methods such as Isolation Forests, Density-based spatial clustering of applications with noise (DBSCAN), Local Outlier Factor (LOF) score, etc.

- *Data representation* refers to the procedure on finding the optimal alphabet to express the data so that it is adequate for the machine learning algorithm at hand.

- In some cases data representation is relatively clear, for example, in regression models it seems obvious to use real numbers.

- Nevertheless, in most of the cases, the decision is not trivial, assume that we have one fature that represents four classes A,B, C,D
  - One possibility is to consider them ordered and to use e.g. {1,2,3,4}.

  - Another possibility is to consider them un-ordered and use a binary representation e.g. {00,01,10,11}.

  - Finally we can consider the relative distance among the classes, considering real numbers e.g. {1,1.5,2.5,4}.

- In the case of categorical data where ordering has no sense, one possibility is to employ *one hot encoding*, that simply consists on creating "dummy" binary variable with the same number of bits as the number of classes in the original dataset, e.g.

- 4 classes: A, B, C D

A	1	0	0	0
B	0	1	0	0
C	0	0	1	0
D	0	0	0	1

- 7 classes: A, B, C, D, E, F, G

A	1	0	0	0	0	0	0
B	0	1	0	0	0	0	0
C	0	0	1	0	0	0	0
D	0	0	0	1	0	0	0
E	0	0	0	0	1	0	0
F	0	0	0	0	0	1	0
G	0	0	0	0	0	0	1

- *Incomplete data* is somewhat different to missing data, in this case we refer to the fact that some data might be simply non existent and must be created 'ad-hoc', for example using simulated data.

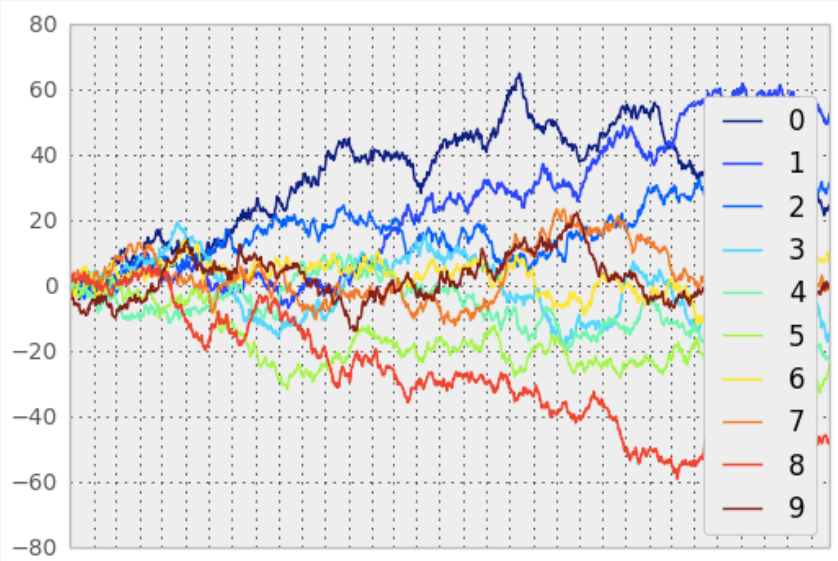
- *Simulated Data* refers to the observations that are created artificially to complete a data set or even as to be used as the single database.

- In many cases, simulated data is the only alternative when data is non-existent or scarce and there exists a model for data creation.

- For example, let us assume that we are building a ML algorithm and one of the inputs is the arrival of orders to some hub, assume that orders arrive randomly following a discrete random walk:

$$O_t = O_{t-1} + \eta_t, \eta_t \approx N(0,1)$$

- Assume that we need many examples of the evolution of the process for the first 1000 observations starting at $O_1=0$, we could simply employ the known model to generate such data and then employ it as an input to our algorithm.



- This is also the basis of *Montecarlo methods* which are widely employed in ML to evaluate alternative algorithms, to measure the sensitivity against features and many other applications besides data generation.

- *Imbalanced Data* refers to the problem that some data examples might be underrepresented in the database.

- This intrinsecally induces a bias in the learning algorithm which will pay more attention to the data that it is over represented.

- As an example, assume that some database includes two classes, class A with 95% of the examples and class B with 5% of the remaining ones.

- A "lazy" classifier may predict just class A and still attain a 95% of accuracy giving the wrong impression of an excellent behavior.

- The problem of imbalanced data is extremelly common in ML, not only in classification problems but also in regression, for example when we record returns of some stock most of the observations are close to zero so the model is naturally biased not to consider extreme observations.

- Imbalanced data must be treated properly if one wants to avoid biases in data which are frequently incorrectly labeled as “algorithmic biases”.

- When the database is imbalanced we may adopt four strategies:

  - Oversample the under-represented class by e.g. duplicating observations until the dataset is balanced

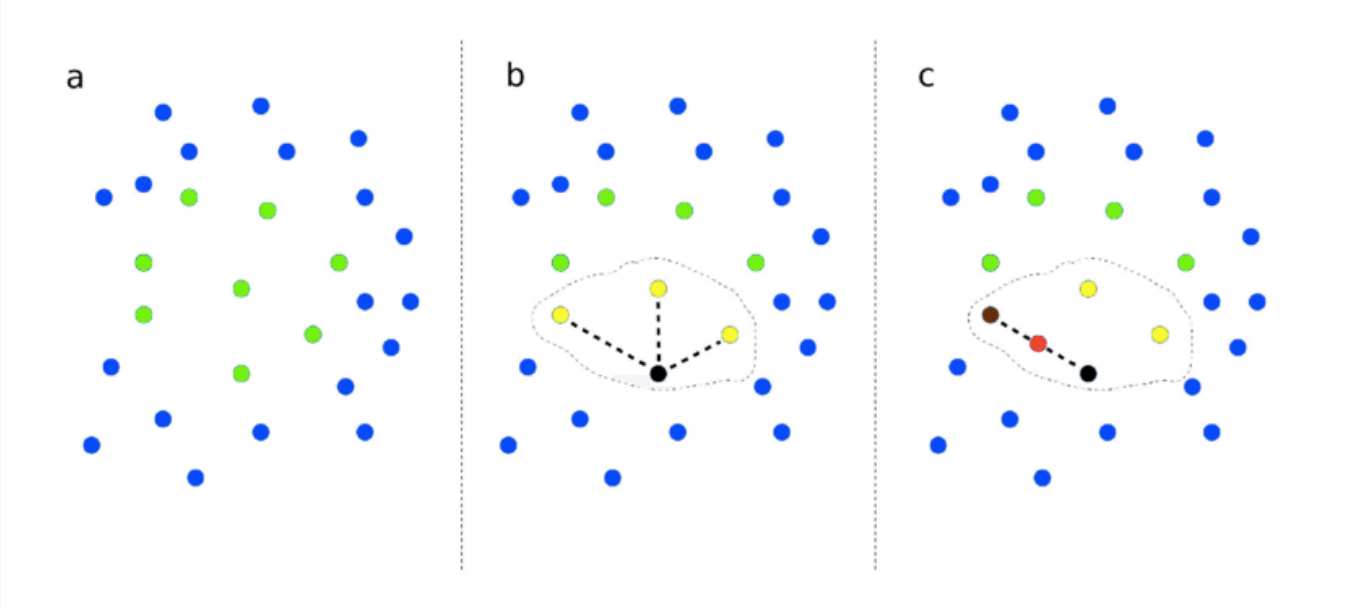
  - Undersample the over-represented class by removing observations until the dataset is balanced

  - Create data synthetically, in a similar manner as we mentioned before

  - Modify the cost function, giving more weight to the class/ data that it is under represented

- Regarding data creation for imbalanced data, one popular algorithm is *SMOTE* (*Synthetic Minority Oversampling Technique*) which follows the steps:

    1. For each example from a minority set, choose the *k* nearest neighbors
    2. Select randomly one instance from the nearest neighbors
    3. Create a new instance with features as a convex combination of the features of the original instance and the nearest neighbor



- The synthetic observation is created as

$$x_i^{'} = x_i + \lambda(x_j - x_i)$$

where λ is a random number in [0,1].

- A very similar technique is called *adaptive synthetic sampling method* (*ADASYN*).

- In ADASYN, the number of synthetic examples generated is proportional to the number of examples in the group of neigbors which are not from the minority class.

- Note that in this case more synthetic examples are generated in the area where the examples of the minority class are rare.

- Each one of these alternatives have its drawbacks:

  - Oversample may lead to overfitting
  - Undersample leads to the sample reduction
  - Synthetic data may have low quality or may difficult to replicate
  - Modified cost functions it is difficult to calibrate it and the learning algorithm needs to be re-written.

- *Data reduction* refers to the procedure os "compressing" features to reduce the complexity of the problem.

- Compressing can be interpreted in two ways: first, we may eliminate some features that may not have predictive power, in such case we properly talk about *feature selection*.

- In other cases, we may think that the granualrity of the problem is too high and we may be interested in combining attributes to create new (fewer) ones, in this case we say we are performing *dimensionality reduction*.

- Notice that, in both cases, the effectve number of features is reduced so that the complexity of the problem wil be also reduced, making it more easy to build and depurate ML models.

- Regarding complexity reduction a well known principle in Science is the *Occams's razor* that argues that among competing hypotheses the one with the fewest assumptions is probably the most correct.

- In the context of model building this means that models with fewer parameters and features may provide better explanations of data and also better forecasts.

- The process of feature selection can be done under several approaches.

- In the first place, we can make a distinction depending on the choice of the method to eliminate unuseful features, in this case, we can distinguish among:

  - *filtering methods*: try to eliminate ex-ante redundant or unuseful attributes

  - *wrapper methods*: consider feature selection as a search problem

  - *regularization methods:* directly interviene on the model to make it to discard low value features.

- In the second place, the choice of features can be:
  - *incremental*: adding one more feature to the model so that the performance increases most
  - *decremental*: removing the feature that degrades less the performace of the model.
- Notice that, in fact, the underlying model is modified and the effective number of parameters an model complexity are changed.

LOSS AND PERFORMANCE MEASURES

- As we have seen, the definition of Machine Learning involves some idea of improving some particular measure when performing a task.

- This measure must be well defined, easy to compute and understandable, moreover, it should be consistent with the algorithm employed to improve it so that learning efficiently takes place.

- Such measure is called the *loss*

- The choice of the loss function is crucial when developing any ML application: a poor choice of the loss will lead to a poor model, even when data is good and sufficient.

- Generically we can define the problem of learning as

$$\min_W L(W, X) = \frac{1}{n} \sum_{i=1}^{n} L(f(W, x_i), y_i)$$

- Which is: finding the optimal parameters so that the loss computed as the "distance" between the predicted value and the real one is minimized.

- As we see, learning can be formally defined in this way and so the queality of learning crucially depends on the choice of the loss function.

- Notice that parameters W will be optimal only under such loss, so that it needs to be consistent with the problem at hand.

- There must must be also possible to optimize the loss function in an efficient way.

- After a model has been built, some other measure or measures can be employed to evaluate it, such measures are called *performance measures* (also *metrics*).

- Performance measures need not to be the same as loss functions employed when building the algorithm.

- Note that, in principle, this is somehow an inconsistency since the model is optimized using a function and then evaluated using another one.

- This is done for several reasons:

    1. In the first place, loss and performance functions can be closely related, the most obvious situation is the mae and the mse when there are no big discrepancies in the scale of data, to the extent that errors are small one measure will lead to similar results than the other so performance and loss will be consistent.

2. In second place, very efficient algorithms exist for particular loss functions (for example, the backpropagation algorithm, to minimize the mse)

$$\nabla_w L(W, X)$$

- These algorithms may not exist or it may be costly to devise them for arbitrary performance functions.

3. In the third case, performance functions may be some sort of statistical tests that can be used to e.g. compare alternative models.

4. Finally, using a performance function different to the loss function may avoid some overfitting problems, in some cases the models successfully "exploits" properties of the performance function allowing to be be close, in some statistical sense, but far away in a geometrical sense (remember the *Ascombe quartet*, for example).

- The use of different performance/loss functions is still an area of debate beween theorists and practitioners

- Regarding loss functions, for example, when we introduced linear models in regression we mentioned that the forecasts can be evaluated using the *mean squared error*,

$$m.s.e. = g(Y, \hat{Y}) = (Y - \hat{Y})^2$$

- Up to now, we have used a compact notation to simplify, but notice that in reality what we want to do (in the preceeding case) is to compute the m.s.e. along all the observations y_1, y_2,… y_n, that is:

$$m.s.e. = g(Y, \hat{Y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- As mentioned, in this setting, learning can now be interpreted as the process to minimize some loss, that is, to find the optimal parameters β_1, β_2,…, β_n so that (e.g.) the m.s.e. error is minimized, i.e.

$$\min_{\beta 0,\beta 1,\beta 2,\ldots \beta n} g(Y,\hat{Y}_\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 =$$

$$= \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 X_i^1 - \beta_2 X_i^2 - .... - \beta_n X_i^n)^2$$

- Notice that the mse has a number of characteristics that are not apparent but must be taken into account, for example:
  - The mse will be biased if some forecast is particularly bad
  - The mse is expressed in squared units, not in the same units of the variable we want to forecast
  - The mse is sensible to the units measure
  - The mse does not consider proportionality 4 is double than 2 and 8 double than 4 but in the first case the mse is 4 and in the second case 16
- This, among other reasons, make the mse not a perfect loss in any situation.

- The ubiquity of the mse in many ML applications is due to the fact that there exists efficient learning algorithms since, as mentioned, many of them use *gradient descent*.

- Nevertheless, in other settings, we may preffer to employ other measures.

- For example we might employ the *root mean squared error* (rmse) which corrects the effect of using squared units:

$$r.m.s.e. = g(Y, \hat{Y}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- Or the *mean absolute error* (*mae*) that penalize equally along the range of Y

$$m.a.e. = g(Y, \hat{Y}) = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right|$$

- A generalization to both the mse and the mae is the l_k *norm*:

$$\ell_k = g(Y, \hat{Y}) = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right|^k$$

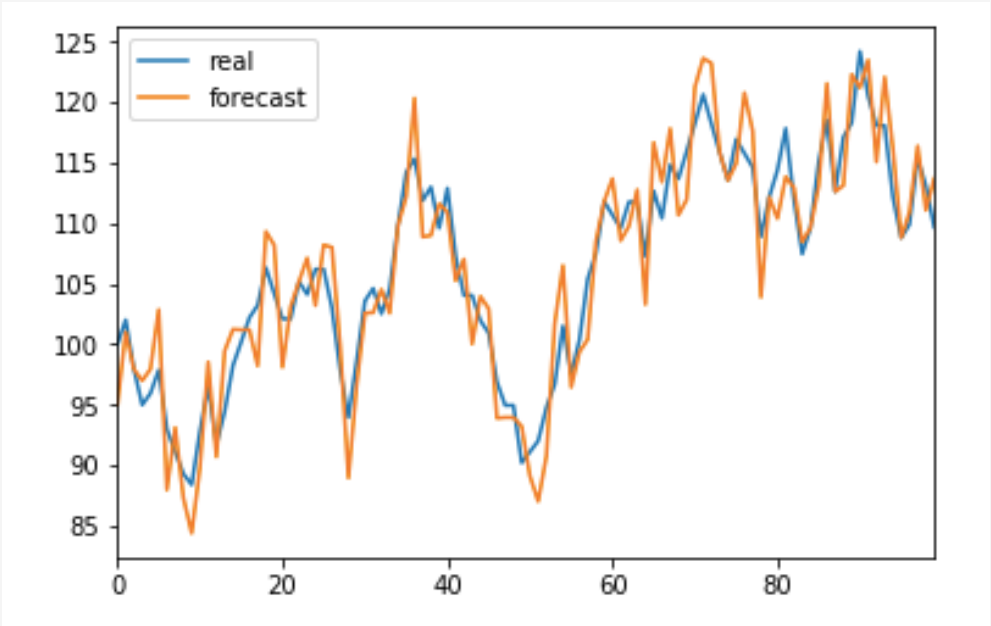
- Note thak for k=1 we have the mae and for k=2 the mse.

- The higher the norm index (k) the more it weights large deviations and the less it concentrates on small deviations.

- To demonstrate the sensitivity of these measures against outliers you can run *measures.ipynb*.

- Finally, it is common to express the error in percentual terms, in such case we can employ the *mean absolute percentage error*:

$$m.a.p.e. = g(Y, \hat{Y}) = \sum_{i=1}^{n}\frac{\left|y_i - \hat{y}_i\right|}{y_i}$$

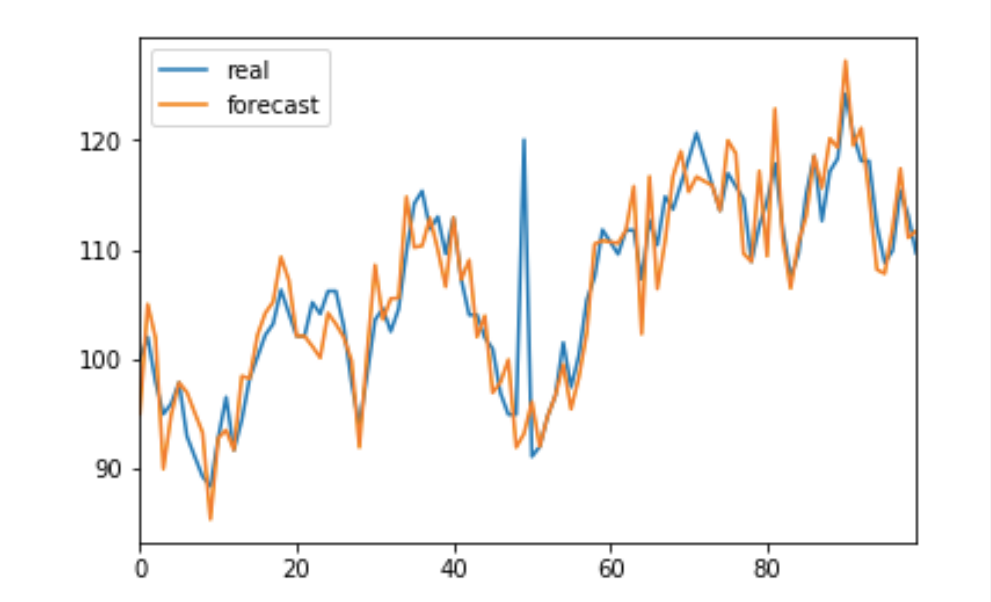



```
mae
2.6

mse
9.16

rmse
3.026549190084311

lknorm (beta=2)
9.16
```




```
mae
2.678

mse
15.672399999999998

rmse
3.958838213415648

lknorm (beta=2)
15.672399999999998
```

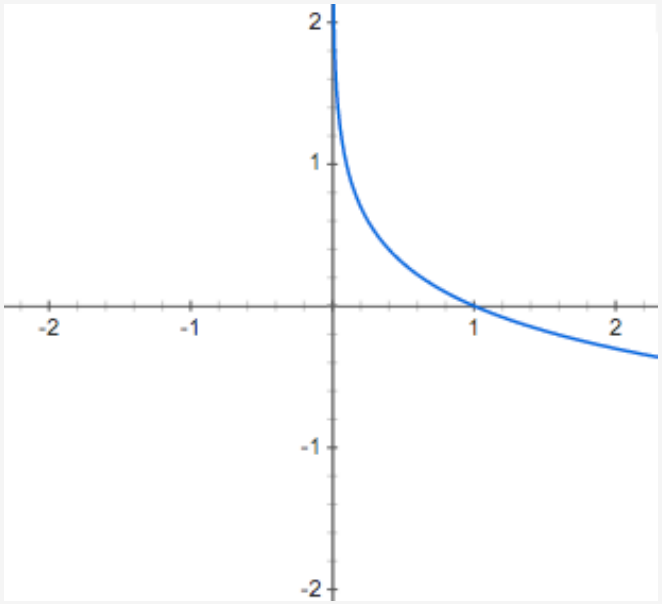
- In the case of binary classification, measures such as the mse do not seem to conform with the target we want to attain which is to minimize the number of misclassifications.

- The *cross-entropy* has been proposed as an approppriate loss function in the case of binary classification:

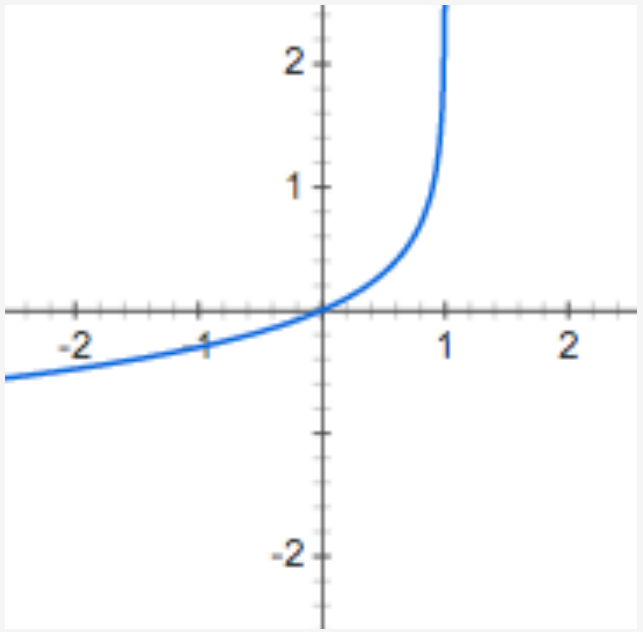
$$c.e. = \sum_{i=1}^{n} -y_i \log(\hat{y}_i) - (1 - y_i)\log(1 - \hat{y}_i)$$

- Notice that

$-log(y)$



$-log(1-y)$



$$y_i = 0, \hat{y}_i = 0 \Rightarrow -y_i \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i) = 0 - 0 = 0$$
$$y_i = 1, \hat{y}_i = 1 \Rightarrow -y_i \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i) = 0 - 0 = 0$$
$$y_i = 0, \hat{y}_i = 1 \Rightarrow -y_i \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i) = 0 + \infty = \infty$$
$$y_i = 1, \hat{y}_i = 0 \Rightarrow -y_i \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i) = \infty + 0 = \infty$$

- So, the value of cross-entropy goes to zero, in case all the predictions are correct and approaches infinity when the predictions are wrong.

- An alternative to cross-entropy, in binary classification is the *hinge loss function*

$$h.l. = \max(0, 1 - y_i \hat{y}_i),\ y_i, \hat{y}_i \in \{-1, 1\}$$

- In the case of multiple classes, the cross entropy can be also extended to:

$$c.e. = \sum_{C=1}^{K} \sum_{i=1}^{n} -(y_i^C) log(\hat{y}_i^C)$$

- Regarding performance functions, in the case of regression, we can use some statistic such as the *R-squared*:

$$R^2 = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2 - \sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \in [0,1]$$

- The R-squared measures what fraction of the variation of the dependent variable is explained by the model.

- Good models have a high value of the R-squared (close to one), and bad models have a low value (close to zero).

- In the case of binary classification (e.g. Y={0,1}, Y={-1,1}) one of the most employed performance measures is the *confusion matrix.*

- It simply records the number of times that the model correctly predicts class 0, the number of times that the model correctly predicts class 1 and the errors when predicting each of the classes.

- Calling 1 to the "positive" class and 0 to the "negative" class we have

		REAL	
		Positive	**Negative**
PREDICTION	**Positive**	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

- Notice that correct predictions (in green) are located in the main diagonal while wrong predictions or errors (in red) in the inverse diagonal of the table.

- The different terms used in a confusion matrix are the following:
  - *True Positive* (TP): These are the cases in which we predicted class "positive" and in fact the class was "positive".
  - *True Negative* (TN): These are the cases in which we predicted class "negative" and in fact the class was "positive".
  - *False Positive* (FP): We predicted class "positive" but the class was "negative".
  - *False Negative* (FN): We predicted class "negative" but the class was "positive".

- One measure of the overall *accuracy* of the algorithm is:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- The *misclassification rate* measures how often the algorithm made a wrong prediction, it is simply:

$$misclassification\_rate = 1 - accuracy = \frac{FP + FN}{TP + TN + FP + FN}$$

- Ideally, one would like the sum of the main diagonal be the total number of predictions

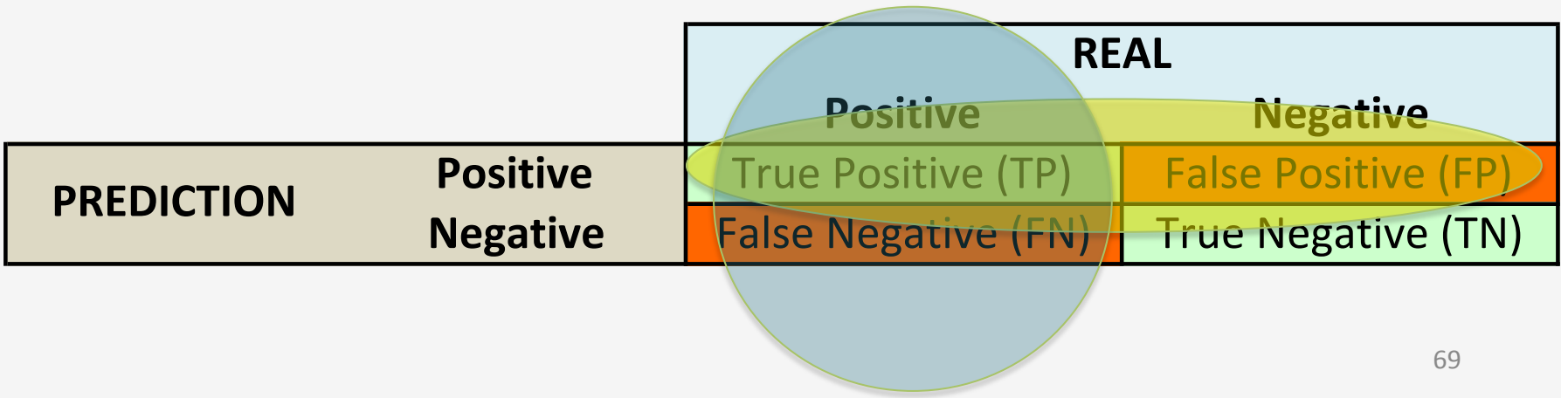
- Most of the times the algorithm will make incorrect predictions by predicting positive when it is negative or the opposite.

- Based on the confusion matrix several measures have been proposed, for example the *precision*:

$$precision = \frac{TP}{TP + FP}$$

- Another measure is the *recall*:

$$recall = \frac{TP}{TP + FN}$$



- Intuitively, let us assume that class “positive” represents good candidates for a job and class “negative” bad candidates.

- Precision tries to answer the question: out of the candidates that we considered good, in which percentage they were truly good?

- Recall tries to answer the question: out of the candidates that are good, in which percentage we correctly detected that they were good?

- Of course one would like to increase both measures at the same time but it can be demonstrated that there is a tradeoff between the two: when we increase one of the measures the other will decrease.

- These two measures are particularly useful when classes are imbalanced

- It is possible to combine precision and recall into a single metric called the *F1 score*, which can be used to compare two or more classifiers with different precision/recall.

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2 \; x \; recall \; x \; precision}{recall + precision}$$

- The F1 score is the harmonic mean of precision and recall and, contrary to the regular mean, it treats all values equally giving much more weight to low values.

- The F1 can vary between 0 and 1 and the score will be high if both recall and precision are high.

- Another popular measure of accuracy *is Matthews correlation coefficient*

$$F1 = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(FN + TN)(FP + TN)(TP + FN)}} \in [-1, 1]$$

- A coefficient equal to 1 corresponds to a perfect classifier while a coefficient equal to -1 corresponds to a completely wrong classifier.

- It has the advantage that can be used even when classes are imbalanced.

- The confusion matrix can be extended to an arbitrary number of classes, e.g. we could consider three classes: positive, neutral and negative:

		REAL		
		Positive	**Neutral**	**Negative**
PREDICTION	**Positive**			
	Neutral			
	Negative			

- In these cases, the performance measurement is more complicated since one has to consider an explosive number of misclassification types.

- Still, it is possible to define measures such as precision and recall for each one of the different classes:

		REAL			
		A	**B**	**C**	**D**
PREDICTION	**A**	TP_A	FN_BA	FN_CA	FN_DA
	B	FN_AB	TP_B	FN_CB	FN_DB
	C	FN_AC	FN_BC	TP_C	FN_DC
	D	FN_AD	FN_BD	FN_CD	TP_D

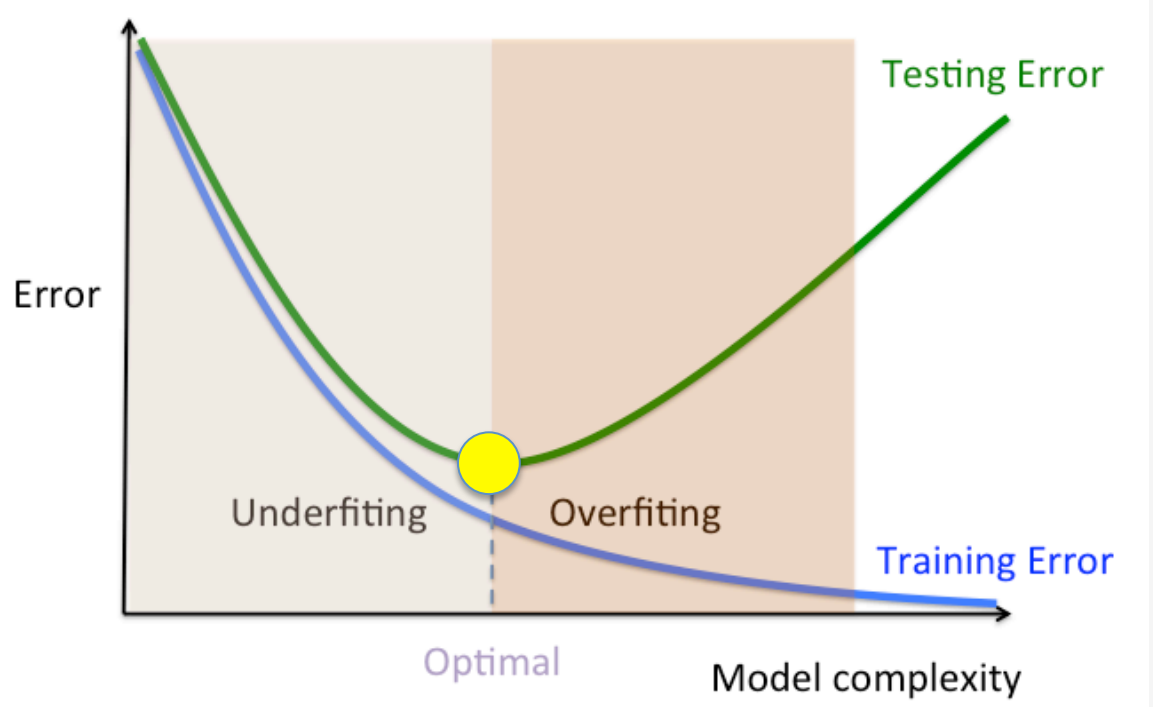
$$precision\_A = \frac{TP\_A}{TP\_A + FN_{BA} + FN_{CA} + FN_{DA}}$$

$$recall\_A = \frac{TP\_A}{TP\_A + FN_{AB} + FN_{AC} + FN_{AD}}$$

DATA SPLITTING

- As mentioned, nonparametric models have the ability to approximate arbitraryfunctions but have the drawback of possibly overfitting the noise.

- In the learning curve it seems obvious to choose the optimal complexity of the model: since it will be used for forecasting purposes the model should be the one which minimizes the testing error.



- The problem is that, in general, we will have no access to the testing set because it precisely consists on observations that we do not know because they still may not have happend or, at least, our predictions could not be tested because we do not know the value of the variable under study.

- For these reasons we have to devise methods that allow us to approximate the testing error using just observations of the training dataset.

- Most of the methods fall into what are called *bootstraping*, *the bootstrap* consists on sampling along the dataset we have at hand and calculating some particular measure of it (a *statistic*

- The mean of the measure is computed and is taken as an approximation to the real value.

- The first approach that has been proposed is to divide the trainig data into three components: pure training (or *training set* from now), *testing set* and *validation set* or *developing set*.

- The training dataset is used just to calibrate the model

- The objective along this dataset would be to reduce, as much as possible, the training error.

- After reaching the optimal calibration, the second set of observations is used to calculate the error on unseen observations.

- If the error rate along this set is similar (formal testing could be employed) to the one in the training set we may conclude that model complexity is adequate and then to use the model in a real setting.

- The error on the validation set is then calculated and it should be similar to the errors in the training and testing sets.

- If the error along the testing set is “significantly” higher than that of the training set this is a indicator of overfitting, the model may have memorized training data losing its capability of generalizing.

- The size of the training, testing and validation sets depend on the availability of data.

- Several years ago it was very common to have 60%/40% or 80%/20% splittings in training/testing (no validation).

- Nowadays, some applications make use of huge amounts of data so that 90%/5%/5% is reasonable or even 98%/1%/1%.

- To avoid the problem of an adverse selection of the testing set and to obtain a more reliable estimate of the validation error, the *cross-validation method* has been proposed, based on the metioned idea of bootstrapping..

- The idea is very simple: the data is splitted into *k* of non-overlapping slices, *k-1* of them are used for training and the remaining one is used to compute the testing error, then another set of *k-1* slices are used and the testing error on the remaining slice is calculated, the procedure continues until all the slices have been used.

$$CVE = \frac{1}{k}\sum_{j=1}^{k} testing\_error_j$$

- It can be demonstrated that the *k*-fold cross-validation error approaches the testing error when the number of observations goes to infinity.

	ITERATION 1		ITERATION 2		ITERATION 3		....	ITERATION 10	
Original Data	Training Data	Testing Data	Training Data	Testing Data	Training Data	Testing Data		Training Data	Testing Data
Slice 1	Slice 1		Slice 1		Slice 1				Slice 1
Slice 2	Slice 2		Slice 2		Slice 2			Slice 2	
Slice 3	Slice 3		Slice 3		Slice 3			Slice 3	
Slice 4	Slice 4		Slice 4		Slice 4			Slice 4	
Slice 5	Slice 5		Slice 5		Slice 5			Slice 5	
Slice 6	Slice 6		Slice 6		Slice 6			Slice 6	
Slice 7	Slice 7		Slice 7		Slice 7			Slice 7	
Slice 8	Slice 8		Slice 8			Slice 8		Slice 8	
Slice 9	Slice 9			Slice 9	Slice 9			Slice 9	
Slice 10		Slice 10	Slice 10		Slice 10			Slice 10	

- There is no guide for the choice of *k* but *k=10* is a very popular alternative.
- The choice of *k* must allow an evenly split of the data so that all the errors have the same value.
- As k increases, the computational effort increases, in the limit we can set *k* equal to the number of observations *n* so that at each round *n-1* examples are chosen for training and the remaining example is predicted.
- This is called *leave-one-out* cross validation or also the *jacknife*.
- Note also that *k=2* is just a "normal" splitting in training and testing set with reversing.
- The choice of examples is generelly done at random but in the case of small samples it is crucial that examples are re shuffled and sampling is stratified to ensure the same proportion of observations of each of the classes.

REGULARIZATION

- As we have mentioned, overfitting is an important problem in nonparametric ML models.

- Remember that ML models try to be operational, in the sense that they should provide good forecasts, a model that correctly fits the training data but fails to generalize is NOT a good model.

- To avoid overfitting, the complexity of the model should keep a balance between fitting the training data and generalizing over unseen examples.

- The problem is that, by definition, we do not know how complex the model should be, otherwise we would not need nonparametric models that increase complexity depending on the problem at hand.

- Only in the case of linear models there exist some formal procedures to select among competing models, for example, we can compute the *Akaike Information Criterion*

$$AIC(k) = e^{\frac{2K}{n}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Where K is the number of parameters of the model, and then select the model that minimizes the AIC.

- Note, however, that even in these cases the results are asymptotical and also they assume strong properties on the estimated model (for example, that has been correctly specified).

- Fortunately, in the context of Machine Learning, a number of techniques have been developed to control model complexity.

- For example, as we have seen before, it is possible to estimate the prediction error using resampling techniques such as cross-validation.

- The estimated error of several configurations of the algorithm, with varying degrees of complexity can be compared and then the model with the minimum expected error is selected.

- Note that this method consist on selecting the best model among the group of estimated models.

- Another possibility is to control, ex ante, the complexity of the model during the learning procedure.

- In principle, this would allow us to use a "single" configuration so that we would not need to estimate different models reducing the computational burdensome.

- The approaches that try to control the complexity of the model during the learning phase are known as *regularization*.

- Regularization techniques essentially consist on modifying the loss function to include a penalty for the complexity of the model.

- The learning algorithm must be conveniently modified so as to minimize this *regularized* function, the resulting model will keep a balance between accuracy over the training examples and complexity, ensuring generalization.

- In general terms, we may think that the loss function employed under regularization is:

$$\min_W L(W,X) = \frac{1}{n}\sum_{i=1}^{n} L(f(W,x_i),y_i) + \lambda\left\|W\right\|$$

- Where is a λ constant that modulates the balance between fitting and complexity penalization.

- When λ =0 then the models is not regularized and the loss function is the standard one, as λ increases the importace we give to model complexity increases.

- Note that as λ increases we reduce the model complexity, reducing the probability of overfitting while, as λ decreases, we increase models complexity reducing the probability of underfitting.

- The norm on W is a measure of model's complexity and, as we will see, there are several possible choices of it.

- An alternative view is to consider that one wants to optimize the model but subject to a complexity restriction (Θ):

$$\min_W L(W, X) = \frac{1}{n} \sum_{i=1}^{n} L(f(W, x_i), y_i)$$
$$\textit{subject to} \quad \|W\| \leq \Theta$$

- As mentioned, there are several posible choices for the regularization norm
  - When it is the L_2 norm we talk about *ridge regression* (also *Tikhonov regularization*)

$$\min_W L(W,X) = \frac{1}{n}\sum_{i=1}^{n} L(f(W,x_i),y_i) + \lambda W^2 =$$

$$\frac{1}{n}\sum_{i=1}^{n} L(f(W,x_i),y_i) + \lambda \sum_{j=1}^{K} w_j^2$$

  - When it is the L_1 norm we talk about *Lasso regression* (*least absolute shrinkage and selection operator*)

$$\min_W L(W,X) = \frac{1}{n}\sum_{i=1}^{n} L(f(W,x_i),y_i) + \lambda \left|W\right| =$$

$$\frac{1}{n}\sum_{i=1}^{n} L(f(W,x_i),y_i) + \lambda \sum_{j=1}^{K} \left|w_j\right|$$

- Even though both methods are very similar, Lasso regression has the advantange that it minimizes the number of effective parameters in the model while ridge regression just minimizes the size of the parameters but all of them remain active.

- It is very easy to understand this geometrically if we consider the methods expressed in the form of restrictions:

Ridge:

$$\min_W L(W,X) = \frac{1}{n}\sum_{i=1}^{n} L(f(W,x_i),y_i)$$

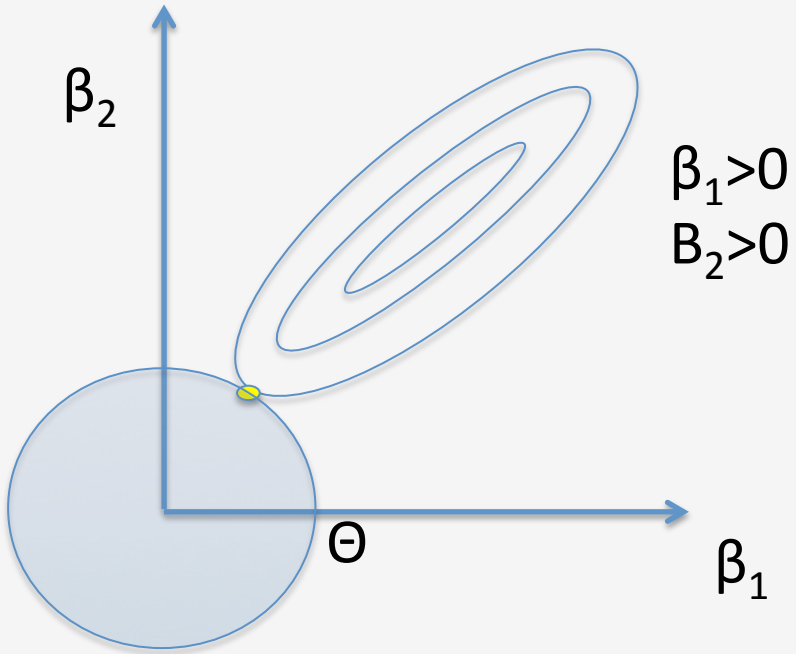
$$subject\ to \quad \|W\|^2 \leq \Theta$$

Lasso:

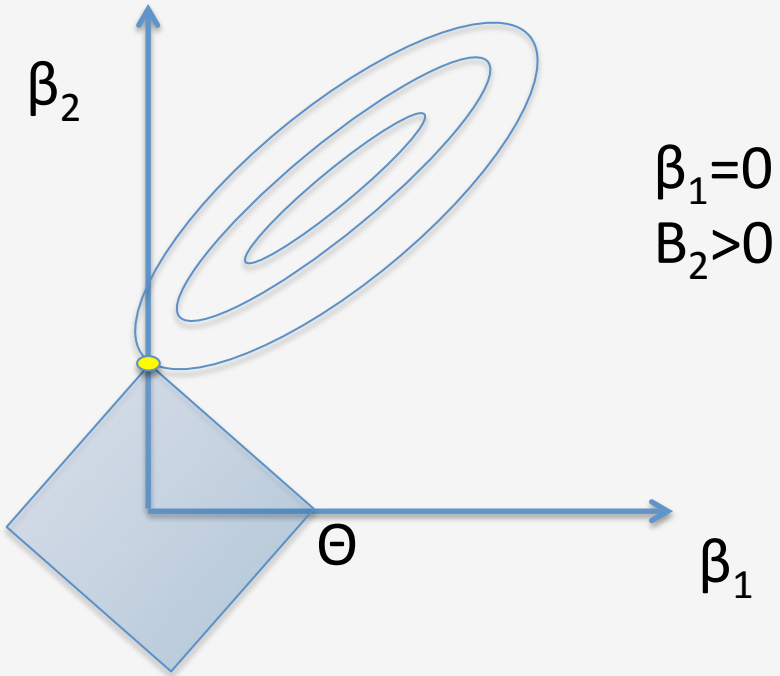
$$\min_W L(W,X) = \frac{1}{n}\sum_{i=1}^{n} L(f(W,x_i),y_i)$$

$$subject\ to \quad \|W\| \leq \Theta$$

Ridge:



Lasso:



- So, ridge regression has the effect of shrinking the coefficients of the model but keeping all of the actitive while Lasso regression also shrinks the coefficients but eliminates some of them, producing more sparse models and also (possibly) a natural way to eliminate variables.

- Regardless of the choice of the regularization method, one needs to modify conveniently the learning algorithm, since the loss function has changed.

- One advantage of the L_2 norm is that it is derivable, while L_1 is not, nevertheless some approximations can be used to calculate the gradients in an effective way.

- Finally, there is a mixture of both regularization methods that is called *ElasticNet regularization*.

- In ElasticNet we use both *Ridge* and *Lasso*:

$$\min_W L(W,X) = \frac{1}{n}\sum_{i=1}^{n} L(f(W,x_i),y_i) + \lambda_1 \|W\| + \lambda_2 \|W\|^2$$

- This kind of regularization forces both weight shrinkage and a global sparsity and its degree of intensity is controlled by the parameters λ_1 and λ_2.

- Another possibility to regularize models is to employ *early stopping* which essentially consists on stopping training with the results that it will not allow the model to overfit.

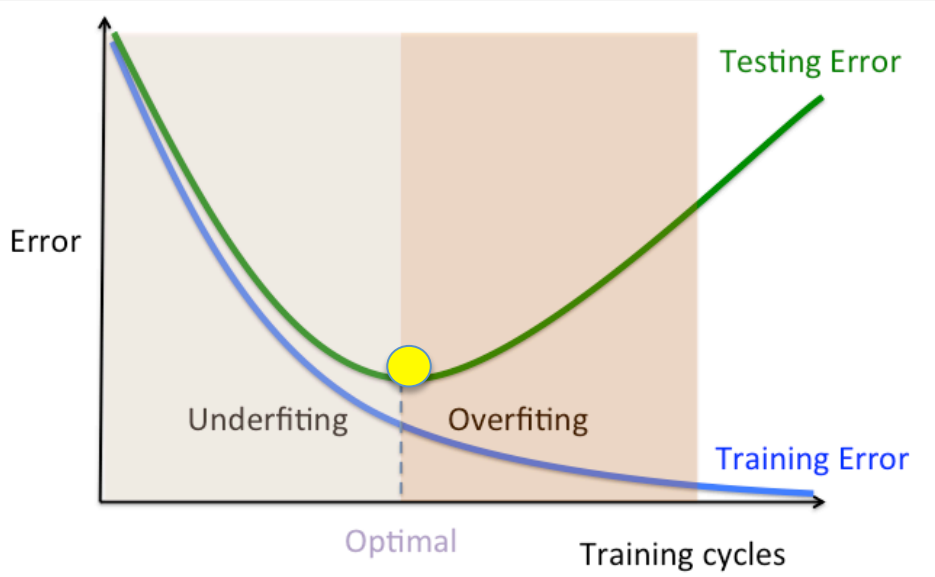
- Remind from the learning curve that the generalization curve has a U-shape, first decreasing and then increassing with model complexity

- It can be demonstrated that this pattern repeats if in the x-axis instead of considering the number of parameters we consider the training cycles because as learning progresses parameters are changed to increasingly fit the training data.

- Similar to humans, machines do need time to learn and "memorizing", in some sense, reduces creativity.

- At the begining of training, most of the parameters will be small (not effective) and as learning progresses they will increase its value, early stopping is as if we had regularized the network.

- If we stop training at the point where the error at the last iteration decreased but the error at the next iteration increases (yellow dot) then we can be sure that model complexity is optimal.



- For some authors early stopping provides and efficient way to control for complexity without the burden to modify the loss function and the training algorithm.

- For some others it "mixes" the phases of learning and model selection and will not be as effective as a two-step procedure.

- The price that we have to pay is to keep track of the parameters in the model in the preceeding phase so that those parameters are used in the case that error begins to increase, instead of the actual ones.

- Also, notice that we have to compute the error in the testing set at each iteration

- After the optimal model has been found one can employ the whole dataset (training + testing) to re-train the model.

- Notice that this has the drawback that we do not know how many cycles we will have to run and whether it is better to re-start using another set of initial parameters or to perform some cycles with the actual parameters .

- In the first case, it has been proposed to monitor the average loss function on the validation set, and continue training until it falls below the value of the training when we stopped.

- Other regularization methods have been also proposed, some of the most popular are:
  - Dropout
  - Injecting noise in the examples
  - Bagging and other ensemble methods
- All these proposals try to correct the important problem of overfitting in ML models.