Data prediction

1

Predictive analytics

Predictive analytics encompasses a variety of (statistical) techniques from data mining, predictive modelling, and machine learning, that analyze current and historical facts to make predictions about future or otherwise unknown events.

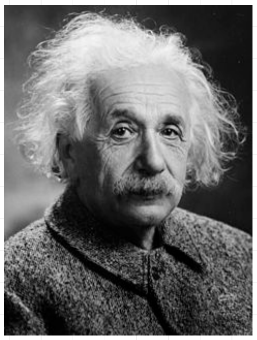
The core of predictive analytics relies on capturing relationships between explanatory variables and the predicted variables from past occurrences, and exploiting them to predict the unknown outcome.

https://en.wikipedia.org/wiki/Predictive_analytics

2

Prediction is very difficult, especially about the future

Everything should be made as simple as possible, but not simpler



Predictive analytics

Predictive

- Primary objective is the prediction of future events on the basis of data (time series or other) available *now*.

Analytics

- Forecasts are used to take decisions *now.*

Examples (management)

- Forecast of goods or services demand
- Forecast of workforce availability or need
- Forecast of material requirements and of warehouse availabilities.
- Forecast of revenues, profits and losses in order to define future investments
- . . .

Forecast horizons

short term (*days, weeks*)

- Resource requirements
- Workforce shifts
- Retail sales

medium term (*weeks, months*)

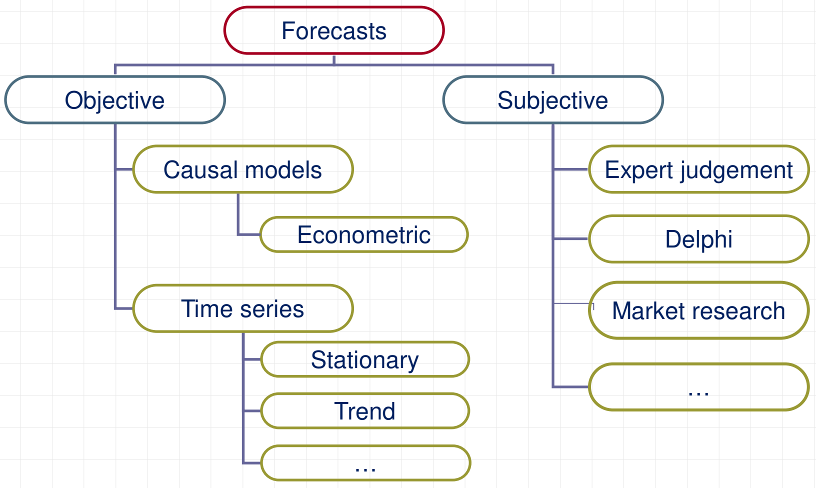
- Sales by product category
- Workforce needs
- Resource requirements

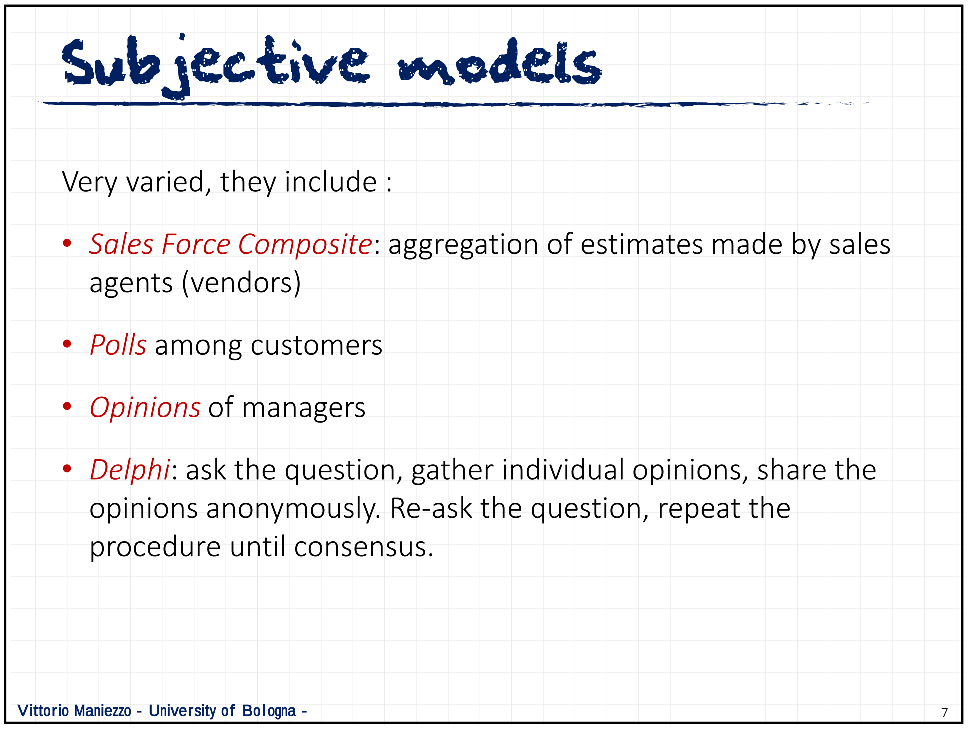
long term (*months, years*)

- Growth trends
- Storage needs
- Sales patterns, market trends

Prediction models

Several types of models for data forecast



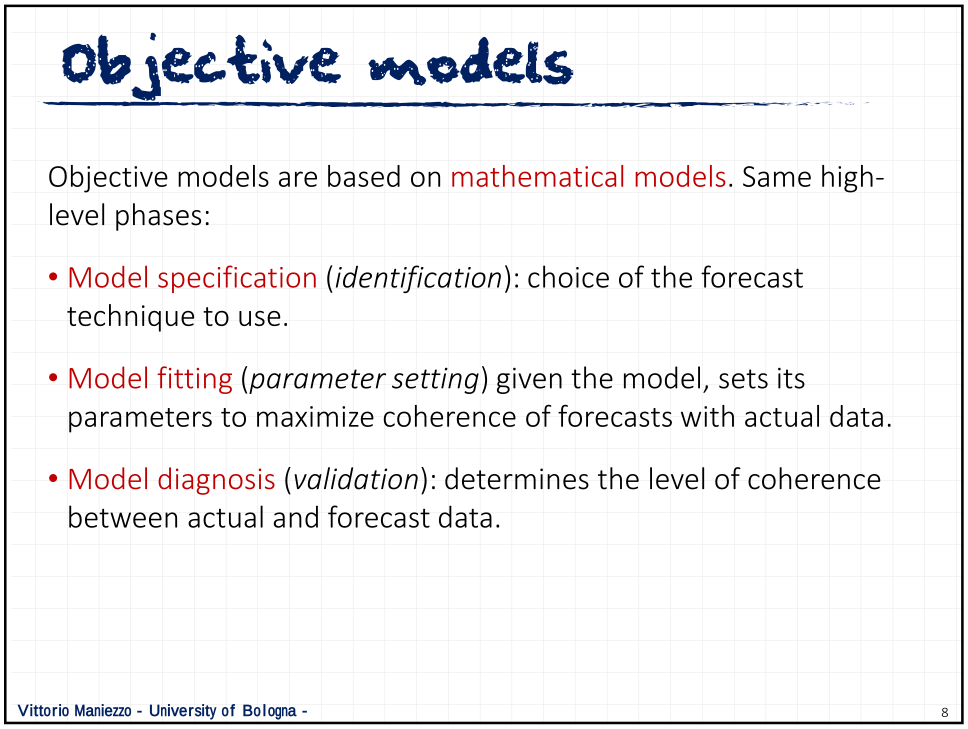


Subjective models

Very varied, they include :

- *Sales Force Composite*: aggregation of estimates made by sales agents (vendors)
- *Polls* among customers
- *Opinions* of managers
- *Delphi*: ask the question, gather individual opinions, share the opinions anonymously. Re-ask the question, repeat the procedure until consensus.

7



Objective models

Objective models are based on mathematical models. Same high-level phases:

- Model specification (*identification*): choice of the forecast technique to use.
- Model fitting (*parameter setting*) given the model, sets its parameters to maximize coherence of forecasts with actual data.
- Model diagnosis (*validation*): determines the level of coherence between actual and forecast data.

8

Econometric models

An econometric model yields a theory on how different economic factors interact among themselves.

The model tries to predict how a change of a variable affects the future values taken by other variables.

Typically defined along these phases:

1. Choice of the variables to include in the model;
2. Separation between endogenous and exogenous variables;
3. A priori definition of the causal relationships among the variables;
4. Specification of quantitative equations;
5. Parameter setting;
6. Forecast.

Econometric models, example

Relation between national income and consumption, as proposed by J.M. Keynes (*1936*)

$$c = \mu + \beta y$$

Where c is the consumer spending, y is the real disposable income and μ and β are parameters.

$\beta = {}^{dc}/_{dy}$ is called marginal propensity to consume, it is positive and less than 1, while μ, autonomous consumption, is strictly positive.

The model has (potentially) observable quantities, consumptions and income (the variables), and other non observable ones, the parameters μ and β.

Time series

A time series is simply a set of time-indexed values, and denotes the dynamics of a process over time.

In the model, we assume to have n observations coming from as many dependent random variables.

Objective of the study of time series is to find patterns in past data, which can be assumed to repeat in the future.

Time series, examples

Can be found in any economic sectors

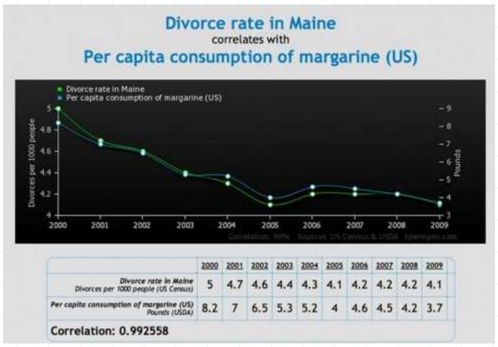
Organization	Series
Sales agent	Daily, weekly, monthly, … sales
Manufacturing company	Monthly labor costs (hours or euros)
Manufacturing company	Trimestral sales (units or euros)
Local government	Fiscal income (monthly)
Local government	Weekly building permits
Parish	Attendance at masses, weekly
University	Num. of students enrolled (absolute, percent)
Bank	Loans granted, weekly (number, euro)
agricultural company	Production per hectare, yearly
hospital	Man-days of hospitalization, monthly
Family	Telephone costs, bimestrial

Time series

A *time-series plot* (timeplot, time series graph) is a bidimensional chart of the time series.

The vertical axis is the variable of interest, the horizontal axis the time units.

Ex.



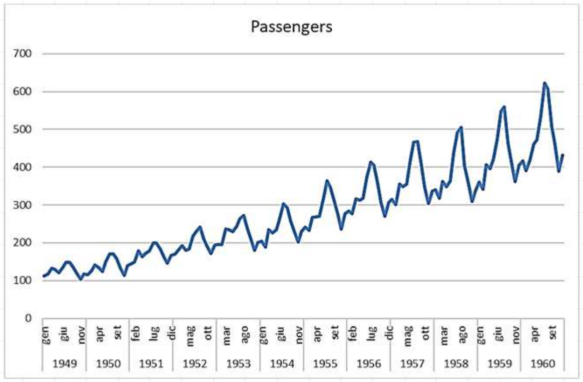
	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009
Divorce rate in Maine Divorces per 1000 people (US Census)	5	4.7	4.6	4.4	4.3	4.1	4.2	4.2	4.2	4.1
Per capita consumption of margarine (US) Pounds (USDA)	8.2	7	6.5	5.3	5.2	4	4.6	4.5	4.2	3.7

Correlation: 0.992558

http://www.tylervigen.com/spurious-correlations

THE time series

International airline passengers: monthly totals in thousands. Jan 49 – Dec 60 (G.E.P. Box, G.M. Jenkins, 1976).



Time series forecasts

Two main approaches (and much in-between):

1. *Statistic*. Based on a well-defined theoretical corpus, yields justified results.

2. *Neural*. More recent, often provides better results but with little justification of the applied models.

Time series forecasts

In statistical methods, after training on historical data, an equation presents the relationship between output and its corresponding factors.

In artificial intelligence methods, the human way of thinking would be copied. Many methods: expert system, evolutionary programming, fuzzy systems, etc. but by far the most effective ones are neural methods.

Recently, other effective approaches (e.g., ensemble methods) have appeared.

Regression analysis

Regression is the general term often used in the ML literature to refer to prediction tasks.

Regression analysis tries to estimate the relationships among variables, usually the relationship between a dependent variable and one or more independent variables (or *'predictors'*).

Regression analysis is not limited to prediction and forecasting, but more generally wants to determine the (average) value of the dependent variable when the independent variables are fixed.

It often takes the form of a function relating the independent variables to the dependent one.

Regression, forecast, prediction

Prediction tasks:

- *Forecasting*: out of sample observations,
- *Prediction*: in sample observations.

Predicted values are calculated for observations in the sample used to estimate the regression.

Forecast is made for some date *beyond* the data used to estimate the regression, the data of the forecasted variable *are not in the sample* used to estimate the regression.

Residuals: difference between the actual value of Y and its predicted value for observations in the sample.

Forecast error: difference between future value of Y, not contained in the estimation sample, and the forecast of the future value.

Time series and forecast

Forecasting a time series wants to predict the values a time series is going to take in the future.

Two types of time series forecasting:

- *Univariate* Time Series Forecasting: uses only the previous values of the series to predict its future values.
- *Multi Variate* Time Series Forecasting: use predictors other than the series (a.k.a. exogenous variables) to forecast.

Forecasting vs Regression

Often “forecasting” used for univariate and “regression” for multivariate (!).

Forecasting uses past data (ex. past sales amounts) to predict future homogeneous data (ex. future sales amount),

Regression uses exogenous data (ex. the attributes of a transaction) to predict the number of sales.

In regression, the independent attributes for future transactions must be knowable. F.e., a dairy producer may have regular customers, that put their orders in on fixed days. Attributes are the time the order is in, product price, planned promotions.

- If forecast involves prior knowledge on future events, f.e. product price change, then regression is ideal.
- If a future setting could not be known, f.e. the weather on a certain day, then forecasting could be better.

Forecasting vs Regression

Another key difference is time.

Regression can use time and even history to make a prediction, but they are not key or mandatory components. A regression task does not necessarily order input data from past to future.

Forecasting requires time and history as key components to make its prediction.

1. Regression predicts the value of a number in a hypothetical future scenario.
2. Forecasting uses a compiled timeline of data moments to tell how it will continue into the future along those trends.

Time series decomposition

Time series decomposition deconstructs a time series into several components, each representing one of the underlying categories of patterns.

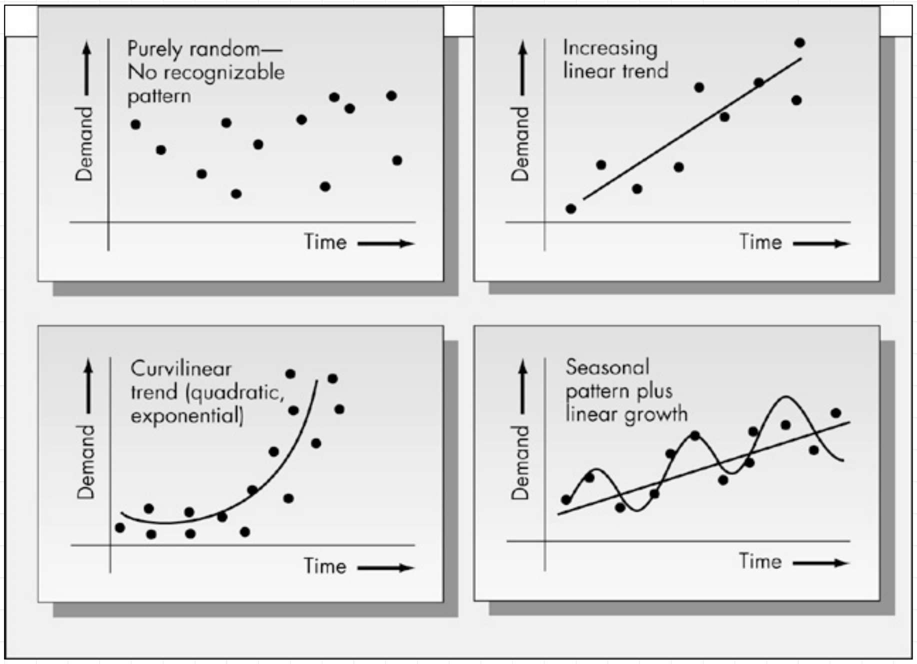
A common decomposition is into:

- Trend component
- Seasonal component
- Cyclical component
- Random component
- (occasional component)

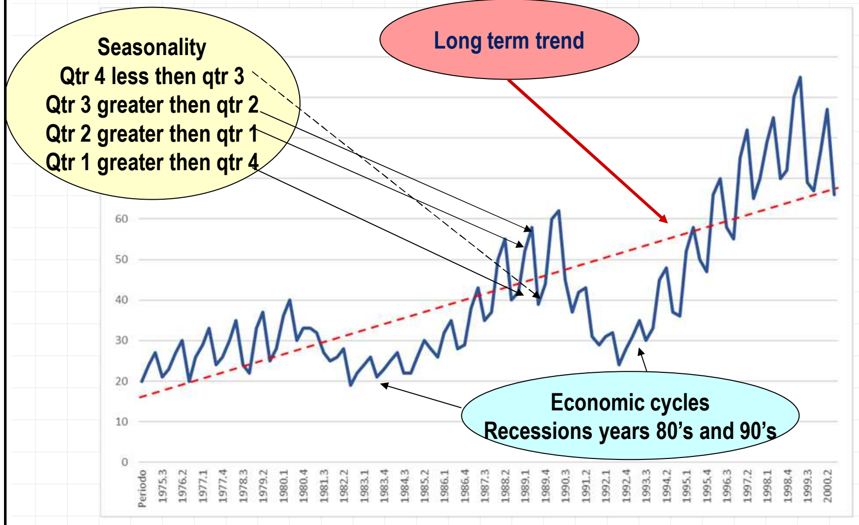
A common model is STL (seasonal decomposition of time series by Loess, where Loess is short for Local Regression), that decomposes the series into seasonal, trend and irregular, where the cyclical component is included in the trend component plot.

Time series decomposition

- *Trend component* is a persistent increasing or decreasing direction in the data. Usually modelled by a linear, quadratic or exponential function, but can be any analytical distribution.
- *Seasonal component* includes repetitive patterns, typically over periods of weeks, months or years.
- *Cyclical component* reflects repeated but non-periodic fluctuations (non seasonal) lasting at least two years, typically due to changes in economic conditions (recessions, expansions).
- *Random component* (irregular, noise): random, irregular influences
- *Occasional component*: rare, due to war events, important technological innovations, political crises, etc..



Camper sales, USA



Time series model

Hypothesis: the observed y_t time series values result from the combination of different components.

- Long term trend μ_t
- Seasonal component s_t (period L)
- cyclical component c_t
- Random residual, irregular r_t.

The components are composed into different types of models:

Additive models: $y_t = \mu_t + s_t + c_t + r_t$

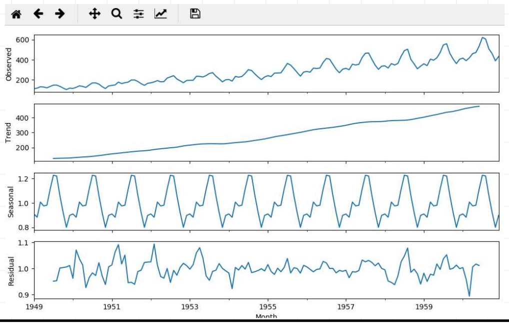
Multiplicative models: $y_t = \mu_t \times s_t \times c_t \times r_t$

Mixed models: $y_t = \mu_t \times s_t \times c_t + r_t$

An additive model is used when the variations around the trend do not vary with the level of the time series, a multiplicative model is used if the trend is proportional to the level of the time series.

Decomposition, python

```
import os, pandas as pd
from matplotlib import pyplot as plot
from statsmodels.tsa.seasonal import seasonal_decompose
if __name__ == "__main__":
   abspath = os.path.abspath(__file__)
   os.chdir(os.path.dirname(abspath))
   df = pd.read_csv('..\\rawAirlinesPassengers.csv',index_col=False,header=0);
   series = df.iloc[:,0] # convert to Series object (on this file not needed)
   result = seasonal_decompose(df.values, model='multiplicative', period=12)
   plot.rcParams['figure.figsize'] = (10.0, 6.0)
   result.plot()
   plot.show()
```



Stationary series

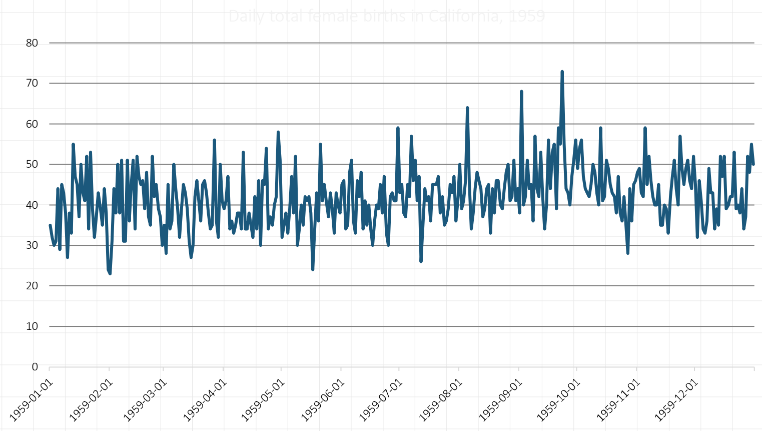
Stationarity conditions. A time series is stationary if:

1. Its mean is constant over time (stationarity in means);
2. Its variance is constant over time (stationarity in variance);
3. The ratio between values separated by k time periods depends only on k, not on time (stationarity in covariance).

Simply, a series is stationary if its distribution is invariant w.r.t. translations in time.

Stationary process

It has no trends (stationary in means and variance). No seasonality or cyclicity.

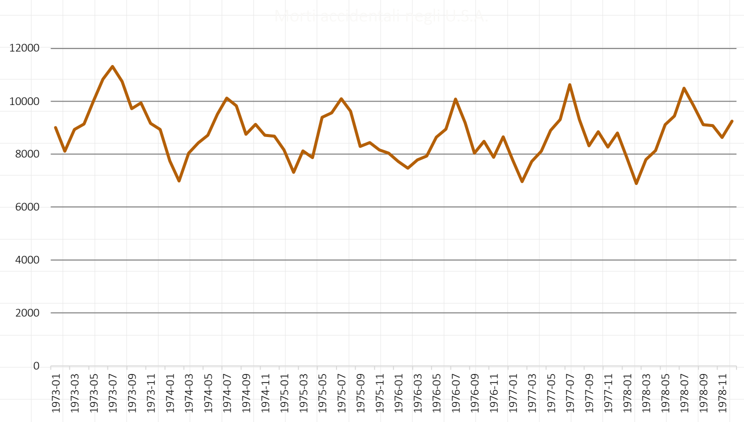


Vittorio Maniezzo - University of Bologna 29

29

Stationary with seasonality

Stationary in means and variance, but with seasonality.

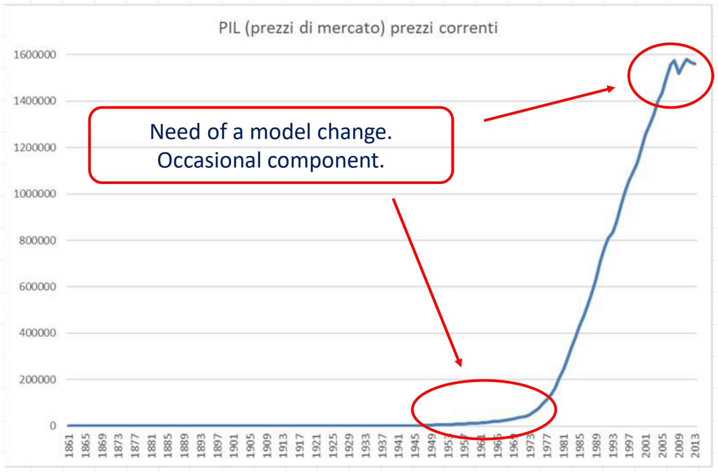


Vittorio Maniezzo - University of Bologna 30

30

Discontinuities

GNP Italy since 1861



Sampling

In any learning process, performance should be estimated with out-of-sample validation, withholding some data from model identification (training phase), then using the model to make forecasts for the hold-out data (test data)

Need to split the data into training and testing buckets. the testing dataset should never be used in the learning step

Data in the estimation are used to help select the model and to estimate its parameters.

k-fold cross validation repeats this process by repeatedly splitting the data into k groups, each given a chance to be a held out model.

Prediction on train values are called *fitted values* and the corresponding forecast errors are called *residuals*.

Train-Test Split

You must split your dataset into training and testing subsets.

Your model must be prepared on the training dataset and predictions made and evaluated for the test dataset.

This can be done by selecting an arbitrary split point in the ordered list of observations and creating two new datasets: you can use for example splits of 50-50, 70-30 and 90-10.

```
import os, pandas as pd
from matplotlib import pyplot
if __name__ == "__main__":
   abspath = os.path.abspath(__file__); os.chdir(os.path.dirname(abspath))
   df = pd.read_csv('..\\rawAirlinesPassengers.csv',index_col=False,header=0);
   X = df.values
   train_size = int(len(X) * 0.66)
   train, test = X[0:train_size], X[train_size:len(X)]
   print('Observations: {}'.format(len(X)))
   print('Training Observations: {}'.format(len(train)))
   print('Testing Observations: {}'.format(len(test)))
   pyplot.plot(train)
   pyplot.plot([None for i in train] + [x[0] for x in test])
   pyplot.show()
```

Forecasts accuracy measures

$X = (x_1, \dots, x_n)$, observations, $F = (f_1, \dots, f_n)$ forecasts

- BIAS:

  arithmetic average of errors $BIAS = \frac{\sum_{i=1}^{n}(x_i - f_i)}{n}$

- MAD (Mean Absolute Deviation):

  avg of absolute values of errors $MAD = \frac{\sum_{i=1}^{n}|x_i - f_i|}{n}$

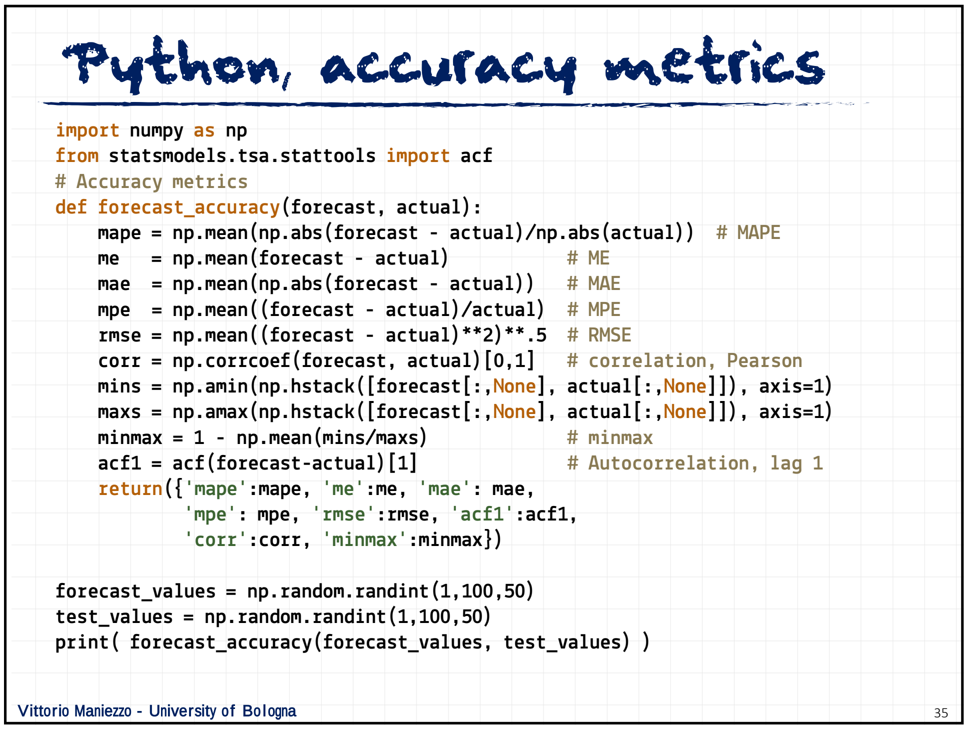
- MSE (Mean Squared Error)

  avg of squared errors $MSE = \frac{\sum_{i=1}^{n}(x_i - f_i)^2}{n}$

- Standard Error: $\sqrt{MSE}$

- MAPE (Mean Absolute Percent Error)

  avg of absolute values of errors in percent: $MAPE = \frac{\sum_{i=1}^{n}\frac{|x_i - f_i|}{f_i} * 100}{n}$

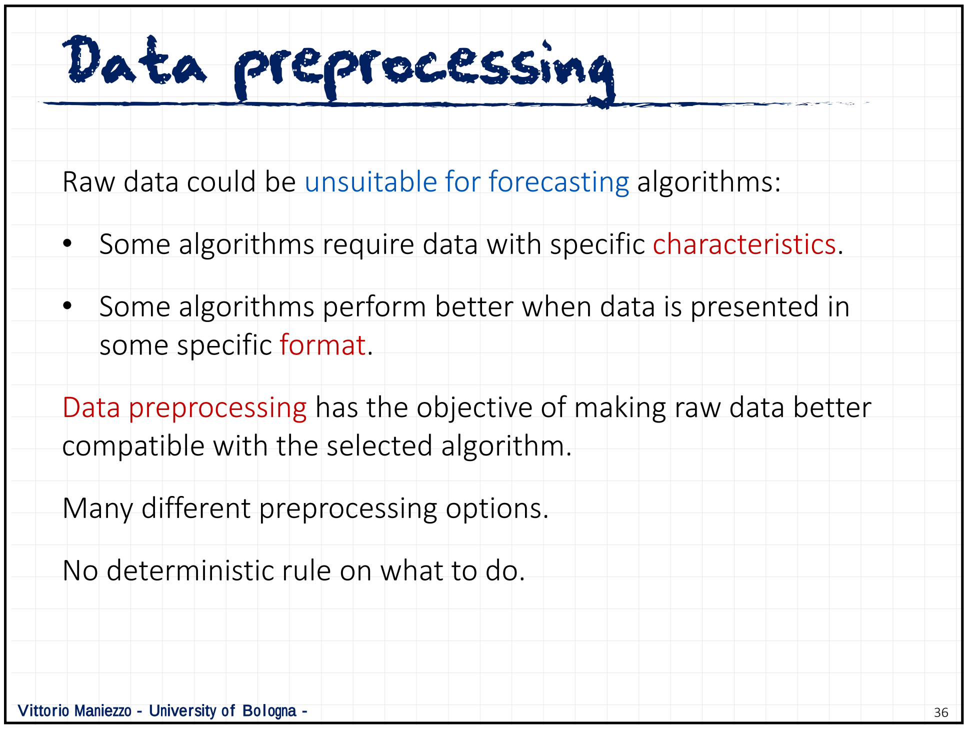


Python, accuracy metrics

```python
import numpy as np
from statsmodels.tsa.stattools import acf
# Accuracy metrics
def forecast_accuracy(forecast, actual):
    mape = np.mean(np.abs(forecast - actual)/np.abs(actual))  # MAPE
    me   = np.mean(forecast - actual)             # ME
    mae  = np.mean(np.abs(forecast - actual))    # MAE
    mpe  = np.mean((forecast - actual)/actual)   # MPE
    rmse = np.mean((forecast - actual)**2)**.5   # RMSE
    corr = np.corrcoef(forecast, actual)[0,1]    # correlation, Pearson
    mins = np.amin(np.hstack([forecast[:,None], actual[:,None]]), axis=1)
    maxs = np.amax(np.hstack([forecast[:,None], actual[:,None]]), axis=1)
    minmax = 1 - np.mean(mins/maxs)              # minmax
    acf1 = acf(forecast-actual)[1]               # Autocorrelation, lag 1
    return({'mape':mape, 'me':me, 'mae': mae,
            'mpe': mpe, 'rmse':rmse, 'acf1':acf1,
            'corr':corr, 'minmax':minmax})

forecast_values = np.random.randint(1,100,50)
test_values = np.random.randint(1,100,50)
print( forecast_accuracy(forecast_values, test_values) )
```

35



Data preprocessing

Raw data could be unsuitable for forecasting algorithms:

- Some algorithms require data with specific characteristics.
- Some algorithms perform better when data is presented in some specific format.

Data preprocessing has the objective of making raw data better compatible with the selected algorithm.

Many different preprocessing options.

No deterministic rule on what to do.

36

Basic transforms

Given a univariate time series dataset, there are some basic transforms often used prior to forecasting:

- Difference Transform
- Power Transform
- Standardization
- Normalization

Vittorio Maniezzo - University of Bologna 37

37

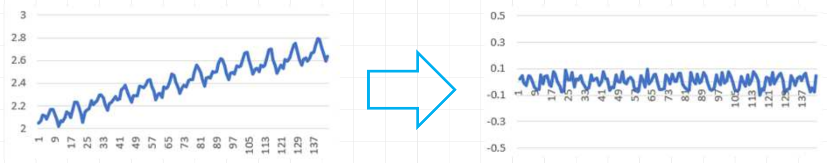
Difference Transform

Difference transform (*diff*) subtracts from each entry in the series the entry back-shifted of a constant time lag.

This is a simple way for removing a systematic structure from the series: e.g., a trend can be removed by subtracting the previous value from each value in the series (*first order differencing*).

The process can be repeated (e.g. difference the differenced series) to remove second order trends, and so on.

A seasonal structure can be removed by subtracting the observation from the prior season, e.g. 12 time steps before for monthly data with a yearly seasonal structure.



Vittorio Maniezzo - University of Bologna 38

38

Difference transform

```
# example of a difference transform (python)
# difference dataset
def difference(data, interval):
   return [data[i] - data[i - interval] for i in range(interval, len(data))]
# invert difference
def invert_difference(orig_data, diff_data, interval):
   return [diff_data[i-interval] + orig_data[i-interval] for i in
range(interval, len(orig_data))]
# define dataset
data = [x for x in range(1, 10)]
print(data)
# difference transform
transformed = difference(data, 1)
print(transformed)
# invert difference
inverted = invert_difference(data, transformed, 1)
print(inverted)
```

Note: *pandas* has the built-in function diff() that can be applied to data series:
`data = pd.Series(data).diff()`

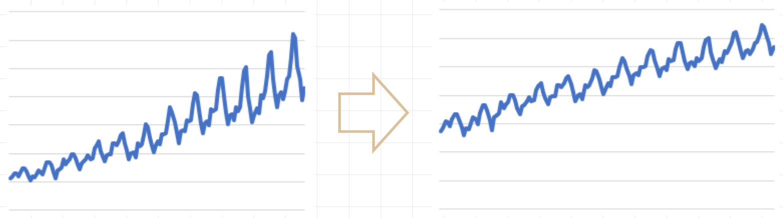
t	*series*	*diff*
1	5	
2	7	2
3	9	2
4	11	2
5	13	2
6	15	2
7	17	2
8	19	2
9	21	2
10	23	2

Power transform (log)

Power transform is a family of functions that create a monotonic transformation of data using power functions, used to stabilize variance, make the data more normal distribution-like.

On a time series dataset, this can remove a change in variance over time.

Popular examples are the log transform, the square root transform or generalized versions such as the Box-Cox transform.



Another transformation boxcox transform, transfoms a non-gaussian var into a gaussian one. Optimizes a parameter λ so that results are as gaussian as possible (*next slide*).

Power transform

```
# example of boxcox power transform and inversion (python)
from math import log
from math import exp
from scipy.stats import boxcox
# invert a boxcox transform for one value
def invert_boxcox(value, lam):
        # log case
        if lam == 0:
                return exp(value)
        # all other cases
        return exp(log(lam * value + 1) / lam)
# define dataset
data = [x for x in range(1, 10)]
print(data)
# power transform, gets optimal lambda
transformed, lmbda = boxcox(data)
print(transformed, lmbda)
# invert transform
inverted = [invert_boxcox(x, lmbda) for x in transformed]
print(inverted)
```

$$y_i^{(\lambda)} = \begin{cases} \dfrac{y_i^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln y_i & \text{if } \lambda = 0, \end{cases}$$

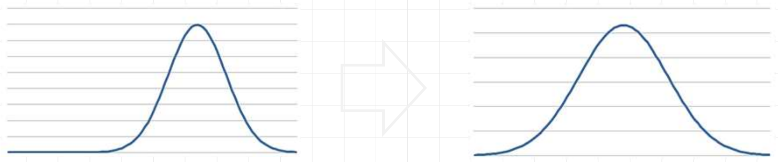
t	series	log
1	5	0.699
2	7	0.845
3	9	0.954
4	11	1.041
5	13	1.114
6	15	1.176
7	17	1.23
8	19	1.279
9	21	1.322
10	23	1.362

Standardization

Standardization is a transform *for data with a Gaussian distribution* resulting in a standard normal (*z score*).

It rescales the distribution of values so that the mean of observed values is 0 and the standard deviation is 1.

It subtracts the series mean $\bar{x}$ and divides the result by the standard deviation σ of the data sample: $z_i = \frac{x_i - \bar{x}}{\sigma}$



Standardization

```
# example of standardization
from sklearn.preprocessing import StandardScaler
from numpy import array
# define dataset
data = [x for x in range(1, 10)]          # list
data = array(data).reshape(len(data), 1) # np array, n rows 1 col
print(data)
# fit transform
transformer = StandardScaler()
transformer.fit(data)
# standard transform
transformed = transformer.transform(data)
print(transformed)
# invert difference
inverted = transformer.inverse_transform(transformed)
print(inverted)
```

t	series	z
1	5	-1.57
2	7	-1.22
3	9	-0.87
4	11	-0.52
5	13	-0.17
6	15	0.174
7	17	0.522
8	19	0.87
9	21	1.219
10	23	1.567

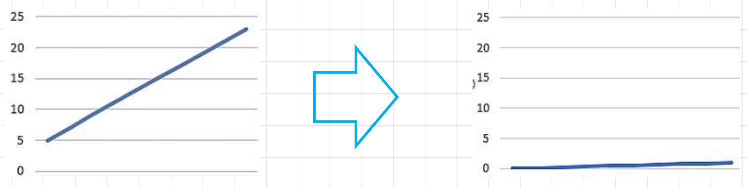
mean	14
stdev	5.745

Normalization

In the simplest case, normalization is a rescaling of data from the original range to a new range (Min-Max Feature scaling), usually between 0 and 1.

$$x_i' = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

In more complicated cases, normalization may refer to adjustments, where the intention is to bring the entire probability distribution of adjusted values into given alignment



Normalization

```
# example of normalization (python)
from sklearn.preprocessing import MinMaxScaler
from numpy import array
# define dataset
data = [x for x in range(1, 10)]
data = array(data).reshape(len(data), 1)
print(data)
# fit transform
transformer = MinMaxScaler()
transformer.fit(data)
# normalized transform
transformed = transformer.transform(data)
print(transformed)
# invert difference
inverted = transformer.inverse_transform(transformed)
print(inverted)
```

t	*series*	*z*
1	5	0
2	7	0.111
3	9	0.222
4	11	0.333
5	13	0.444
6	15	0.556
7	17	0.667
8	19	0.778
9	21	0.889
10	23	1

max	23
min	5

Advanced preprocessing

More sophisticated data preprocessing techniques include:

- Missing values insertions
- Outliers detection
- Feature extraction
- Dimensionality reduction
- Noise reduction
- Data augmentation

Correlograms

A correlogram, or autocorrelation plot, is a chart that represents the autocorrelation of a time series as a function of the lag used for computing the autocorrelation.

To define the correlogram, we consider the correlations between the series and itself shifted by each of K periods;

F.e., given Y = (y_1, ... , y_n) we construct the table below and we examine the *K* correlations between column y_t and each of the K columns y_{t-k}

Y_t	Y_{t-1}	Y_{t-2}	Y_{t-3}	$\cdots$	Y_{t-K}
Y_1					
Y_2	Y_1				
Y_3	Y_2	Y_1			
Y_4	Y_3	Y_2	Y_1		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Y_{T-2}	Y_{T-3}	Y_{T-4}	Y_{T-5}	$\vdots$	Y_{T-K-2}
Y_{T-1}	Y_{T-2}	Y_{T-3}	Y_{T-4}	$\vdots$	Y_{T-K-1}
Y_T	Y_{T-1}	Y_{T-2}	Y_{T-3}	$\vdots$	Y_{T-K}

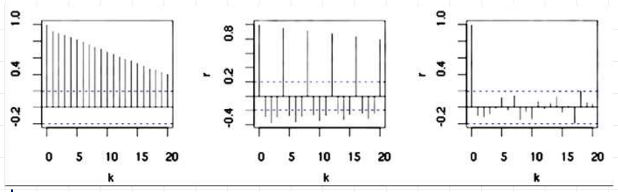
Correlograms

We start from the (K+1)-th row, to compare equal length series.

We compute a value for r_k letting k range from 1 to K and computing the correlation (here Pearson, but could be Spearman) r_k between column Y_t , average $\overline{Y}_t$, and the column of the lagged variable Y_{t-k}:

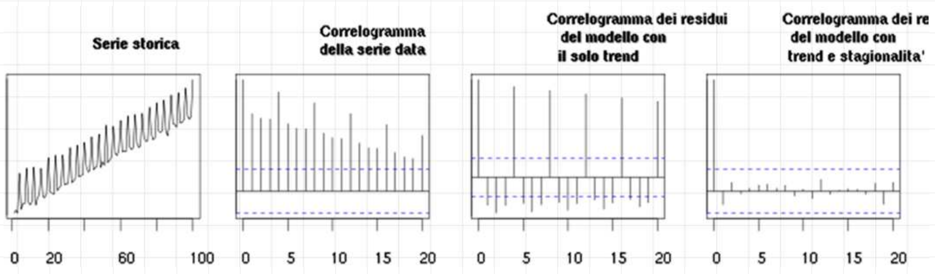
$$r_k = \frac{\sum_{t=k+1}^{T}(Y_t - \overline{Y}_t)(Y_{t-k} - \overline{Y}_t)}{\sum_{t=k+1}^{T}(Y_t - \overline{Y}_t)^2}$$

Pairs of values (k, r_k) are shown on a cartesian chart, with lags on the x-axis and the corresp. correlation on the y-axis:

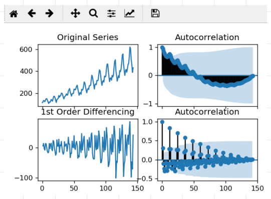


Correlograms

Correlograms can be widely different, but we have three typical cases:



Python, ACF



```
import os
import numpy as np, pandas as pd, matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
# Import data
if __name__ == "__main__":
   abspath = os.path.abspath(__file__)
   os.chdir(os.path.dirname(abspath))
   df = pd.read_csv('..\\rawAirlinesPassengers.csv', usecols=[0], names=['value'],
header=0)
   # Original Series
   plt.rcParams.update({'figure.figsize':(9,7), 'figure.dpi':120})
   fig, axes = plt.subplots(2, 2, sharex=True)
   axes[0, 0].plot(df.value); axes[0, 0].set_title('Original Series’)
   plot_acf(df.value, ax=axes[0, 1])
   # 1st Differencing
   axes[1, 0].plot(df.value.diff()); axes[1, 0].set_title('1st Order Differencing')
   plot_acf(df.value.diff().dropna(), ax=axes[1, 1])
   plt.show()
```

Autoregressive models AR(p)

In autoregressive models (of order p, AR(p)) the future value of a variable (added to a random component and to a constant) is assumed to be a linear combination of the last *p* observations

$$y_t = c + \sum_{i=1}^{p} \varphi_i y_{t-i} + \varepsilon_t$$

Feasible only for stationary processes! It is a linear dependency.
where:

y_t	value at time t
ε_t	random error at time t
φ_i	($i = 1, \dots, p$) model parameters
c	constant

Moving average models MA(q)

In moving average models (of order q, MA(q)) the future value of a variable is assumed to be equal to the average of the observations, μ, added to a linear combination of the last q errors:

$$y_t = \mu + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$$

Where:

μ	average of the series
θ_j	($j = 1, \dots, q$) model parameters

Random errors are usually assumed to have a normal distribution (mean 0, variance σ^2)

ARIMA models

ARMA models can be used only on stationary processes, without trend or seasonality.

ARIMA models (*AutoRegressive Integrated Moving Average*) are ARMA models that work on a diff time series (diff preproc).

ARIMA(p,d,q) =

$$y_t = c + \sum_{i=1}^{p} \varphi_i y'_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$$

p order of the autoregressive component

q order of the moving average component

d diff degree (usually d =1, if d=0 ARIMA=ARMA)

ARIMA model in words:

Predicted y_t = Constant
+ Linear combination lags of y (upto p lags)
+ linear combination of lagged forecast errors (upto q lags)

ARIMA

The key aspects of the model are:

- AR: Autoregression. The model uses a relationship between an observation and some number of lagged observations.
- I: Integrated. The model uses differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.
- MA: Moving Average. The model uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

ARIMA, parameters

Each of its components is explicitly specified in the model as a parameter ARIMA(p,d,q), defined as follows:

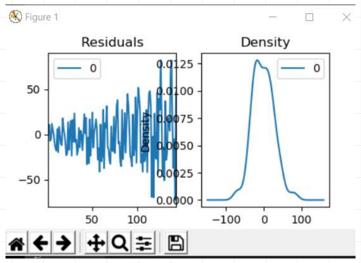
- p: The number of lag observations included in the model, also called the lag order.
- d: The number of times that the raw observations are differenced, also called the degree of differencing.
- q: The size of the moving average window, also called the order of moving average.

Python, ARIMA

```
import os
import pandas as pd, matplotlib.pyplot as plt
from statsmodels.tsa.arima_model import ARIMA
# Import data
if __name__ == "__main__":
   abspath = os.path.abspath(__file__)
   os.chdir(os.path.dirname(abspath))
   df = pd.read_csv('..\\rawAirlinesPassengers.csv', usecols=[0],
names=['value'], header=0)

   # 1,1,2 ARIMA Model
   model = ARIMA(df.value, order=(1,1,2))
   model_fit = model.fit(disp=0)
   print(model_fit.summary())

   # Plot residual errors
   residuals = pd.DataFrame(model_fit.resid)
   fig, ax = plt.subplots(1,2)
   residuals.plot(title="Residuals", ax=ax[0])
   residuals.plot(kind='kde', title='Density', ax=ax[1])
   plt.show()
```



SARIMA models

An extension, proposed by Box, Jenkins (1970), of ARIMA models which can be applied also to data with seasonality.

A seasonal diff is included.

Model identification can be complex.

Notationally, they are denoted by $ARIMA(p,d,q)(P,D,Q)_m$

where:

- p,d,q are ARIMA parameters referring to periods (ex., week),
- P,D,Q the same parameters referring to seasons (ex., trimester)
- m number of period in one season (ex., weeks in a trimester).

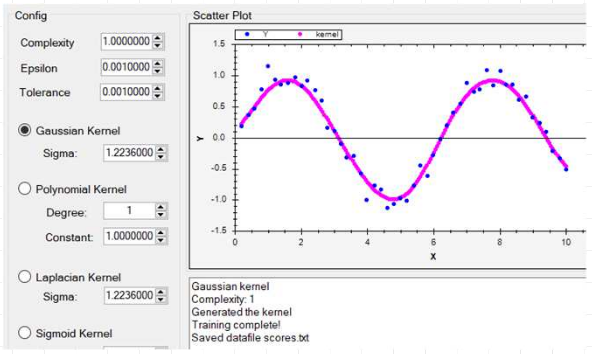
SVR and NN

Two main approaches to forecasts can be ascribed to the neural network literature:

- Support Vector Regression (SVR), based on support vector machines. (Are these neural models?)
- Multilayer or recurrent neural networks (MLP or RNN).

SVR, forecast

Regression models can be directly projected to future periods



Config

Complexity 1.0000000

Epsilon 0.0010000

Tolerance 0.0010000

(●) Gaussian Kernel

Sigma: 1.2236000

( ) Polynomial Kernel

Degree: 1

Constant: 1.0000000

( ) Laplacian Kernel

Sigma: 1.2236000

( ) Sigmoid Kernel

```
Gaussian kernel
Complexity: 1
Generated the kernel
Training complete!
Saved datafile scores.txt
```

NN for forecast

Neural networks (NN) are very effective in identifying non linear models.

Even with input data deriving from complex, nonlinear processes, NN are able to combine linear and nonlinear models to get whichever accuracy level is requested (in prediction):

- NN can learn patterns in linear time series
- NN can learn patterns in nonlinear time series
- NN can generalize linear and nonlinear patterns

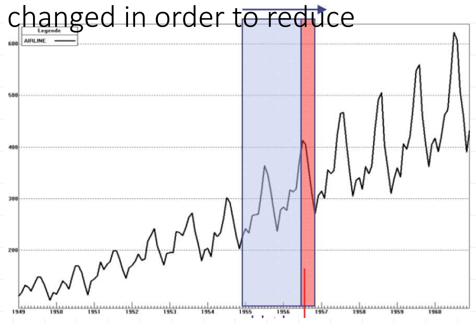
NN are nonparametric, they do not make hypotheses on noise distribution (gaussian or else).

NN identify a model from input data, essentially defining a model of the generating process.

Sliding window

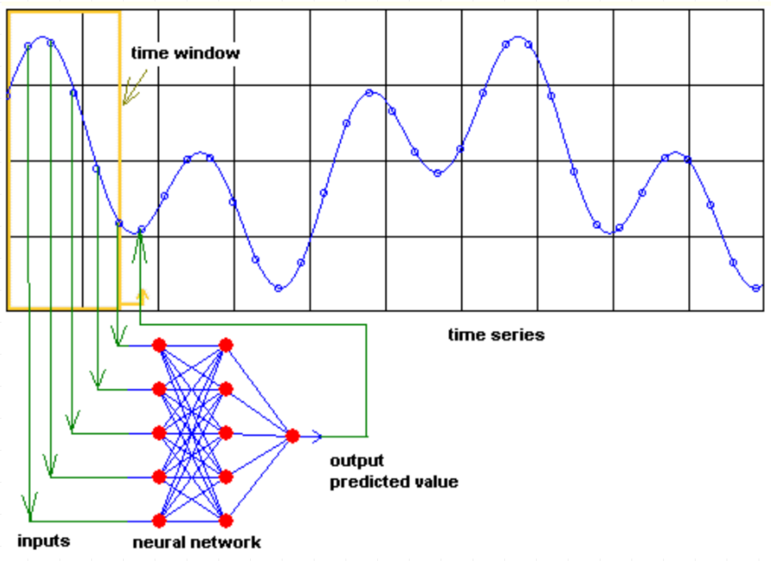
Learning:

1. A window on past data is shown in input.
2. The corresponding output is computed.
3. The output is compared with its corresponding actual value.
4. *Backpropagation*: weights are changed in order to reduce prediction error.
5. The window is shifted forward, presenting the following input.



61

Sliding window

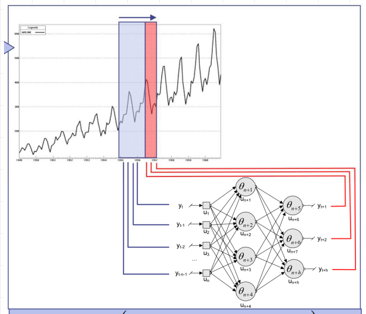


62

Sliding window

NN are able to approximate any function.

Forecasting can be thought of as the identification of a specific (nonlinear) function.



$$\hat{y}_{t+1} = f\left(y_t, y_{t-1}, y_{t-2}, \dots, y_{t-n-1}\right)$$

Non-linear autoregressive AR(p)-model

Only one neuron	$\rightarrow$ AR(p), nonlinear
Feedforward NN (MLP)	$\rightarrow$ combination of AR(p)
Recurrent NN (Elman, Jordan)	$\rightarrow$ ARMA(p,q) nonlinear

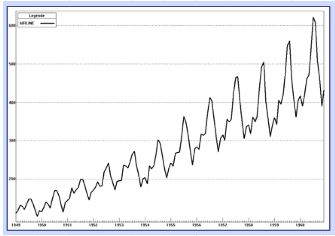
NN, example

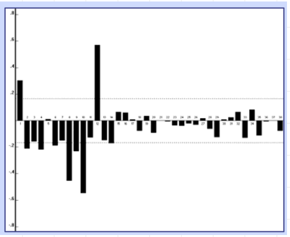
Box, Jenkins series, airline passengers

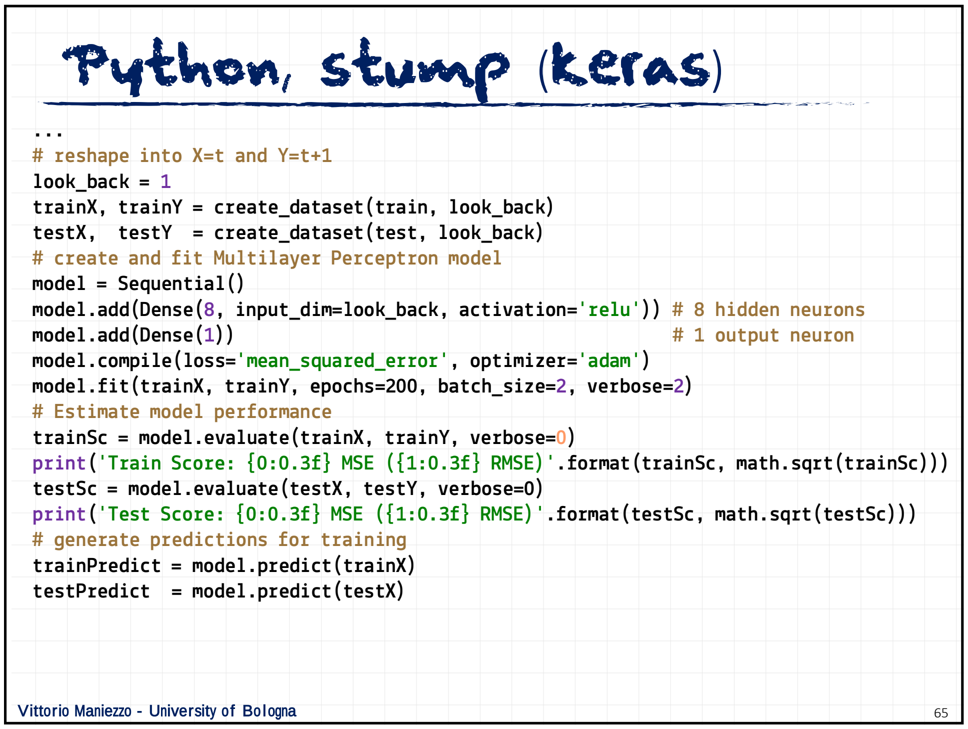
- NN: (12-8-1) 12 input nodes, 8 hidden units, 1 output node
- 12 input values: t, t-1, …, t-11 (last 12 observations)
- Prediction value t+1

Benchmark:

- 132 input values
- 13 time periods (year), monthly data





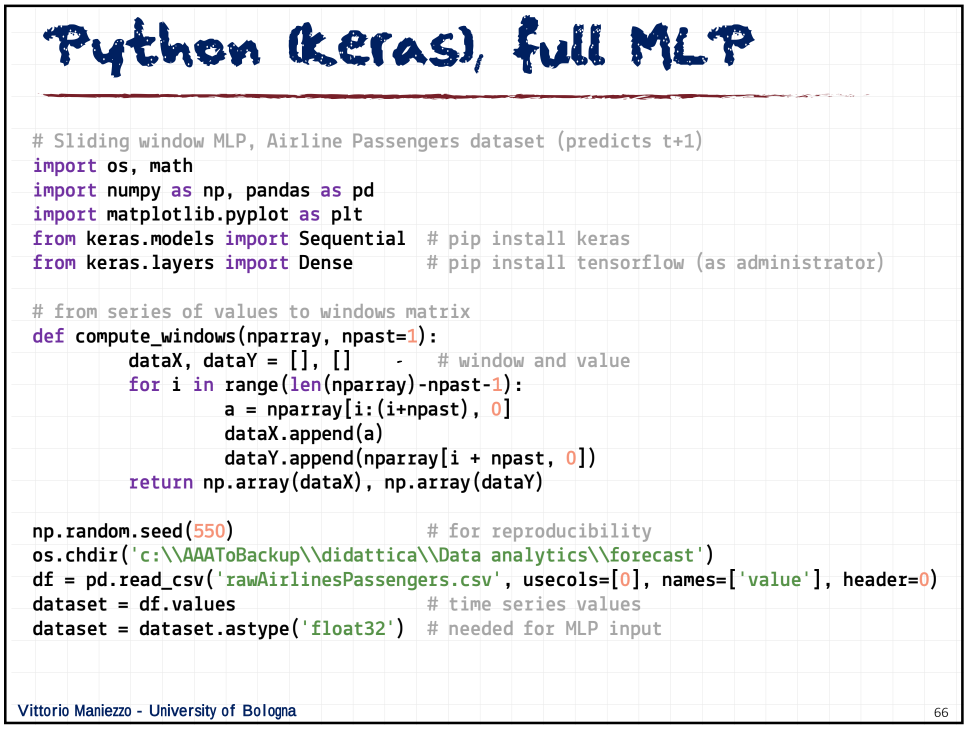


Python, stump (keras)

```
...
# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX,  testY  = create_dataset(test, look_back)
# create and fit Multilayer Perceptron model
model = Sequential()
model.add(Dense(8, input_dim=look_back, activation='relu')) # 8 hidden neurons
model.add(Dense(1))                                         # 1 output neuron
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=200, batch_size=2, verbose=2)
# Estimate model performance
trainSc = model.evaluate(trainX, trainY, verbose=0)
print('Train Score: {0:0.3f} MSE ({1:0.3f} RMSE)'.format(trainSc, math.sqrt(trainSc)))
testSc = model.evaluate(testX, testY, verbose=0)
print('Test Score: {0:0.3f} MSE ({1:0.3f} RMSE)'.format(testSc, math.sqrt(testSc)))
# generate predictions for training
trainPredict = model.predict(trainX)
testPredict  = model.predict(testX)
```

Vittorio Maniezzo - University of Bologna 65

65



Python (keras), full MLP

```
# Sliding window MLP, Airline Passengers dataset (predicts t+1)
import os, math
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential  # pip install keras
from keras.layers import Dense       # pip install tensorflow (as administrator)

# from series of values to windows matrix
def compute_windows(nparray, npast=1):
         dataX, dataY = [], []    -   # window and value
         for i in range(len(nparray)-npast-1):
                  a = nparray[i:(i+npast), 0]
                  dataX.append(a)
                  dataY.append(nparray[i + npast, 0])
         return np.array(dataX), np.array(dataY)

np.random.seed(550)                  # for reproducibility
os.chdir('c:\\AAAToBackup\\didattica\\Data analytics\\forecast')
df = pd.read_csv('rawAirlinesPassengers.csv', usecols=[0], names=['value'], header=0)
dataset = df.values                  # time series values
dataset = dataset.astype('float32')  # needed for MLP input
```

Vittorio Maniezzo - University of Bologna 66

66

Python (keras), full MLP

```
# train - test sets
cutpoint  = int(len(dataset) * 0.7)                           # 70% train, 30% test
train, test = dataset[:cutpoint], dataset[cutpoint:]
print("Len train={0}, len test={1}".format(len(train), len(test)))

# sliding window matrices (npast = window width); dim = n - npast - 1
npast = 12
trainX, trainY = compute_windows(train, npast)
testX, testY   = compute_windows(test, npast) # should get also the last npred of train

# Multilayer Perceptron model
model = Sequential()
n_hidden = 8
n_output = 1
model.add(Dense(n_hidden, input_dim=npast, activation='relu'))  # hidden neurons, 1 layer
model.add(Dense(n_output))                                        # output neurons
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=200, batch_size=10, verbose=2) # batch_size len(trainX)
```

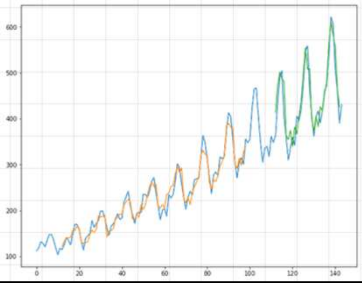
67

Python (keras), full MLP

```
# Model performance
trainScore = model.evaluate(trainX, trainY, verbose=0)
print('Score on train: MSE = {0:0.2f} '.format(trainScore))
testScore = model.evaluate(testX, testY, verbose=0)
print('Score on test:  MSE = {0:0.2f} '.format(testScore))

trainPredict = model.predict(trainX)    # predictions
testForecast = model.predict(testX)     # forecast

plt.rcParams["figure.figsize"] = (10,8) # redefines figure size
plt.plot(dataset)
plt.plot(np.concatenate((np.full(npast,np.nan),trainPredict[:,0])))
plt.plot(np.concatenate((np.full(len(train)+npast,np.nan), testForecast[:,0])))
plt.show()
```



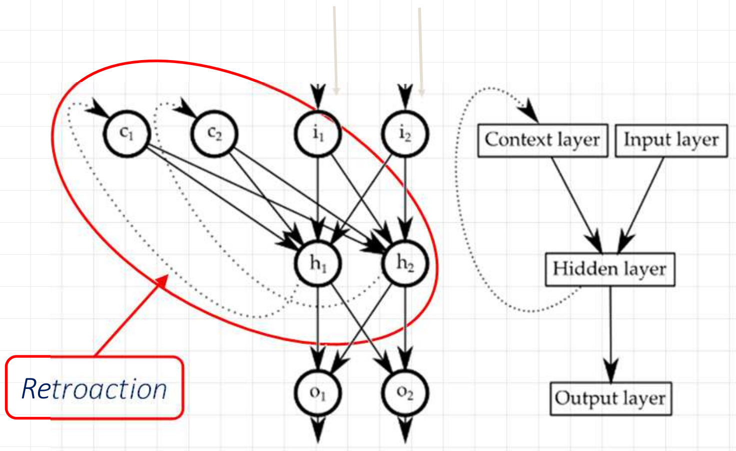
68

Modelling process

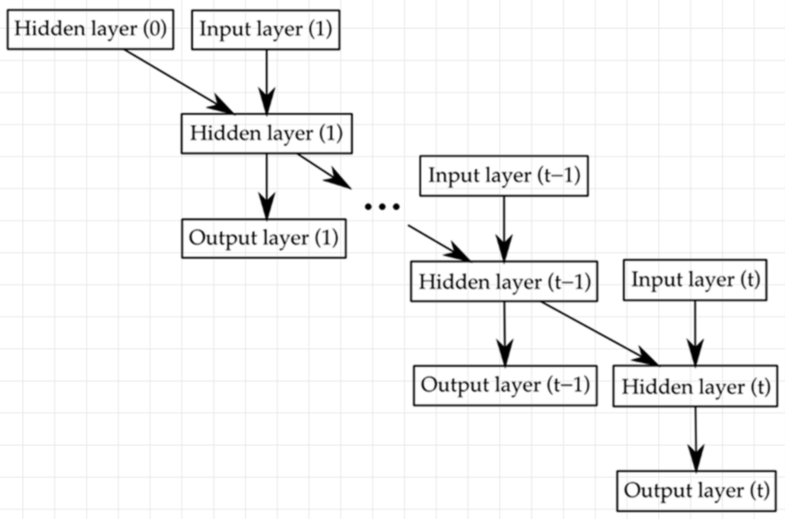
Phases for the definition of a neural forecast model :

1) Preprocessing (the same as statistical models)
   - Transforms (diff, log, ...)
   - Scaling, if different variables they must have comparable values
   - Normalization, to [0,1] or [-1,1]
2) Choice of the architecture of the NN
   - Number of neurons, input, hidden, output
   - Number of hidden layers (one: MLP, many: deep network)
   - Processing at the nodes (activation function: lin, logistic, ...)
   - Connections between layers
3) Training
   - Weights initialization (and reinitialization?)
   - Training algorithm (backpropagation, which one?, GLT ...)
   - Parameter setting
4) Use of the NN
5) Validation
   - Choice of the dataset
   - Validation criteria

RNN



RNN “unfolded”



RNN

RNN accumulate previous activities in the context layer, they do not need an input neuron for each input datum.

RNN do not approximate a function, but they model a process.

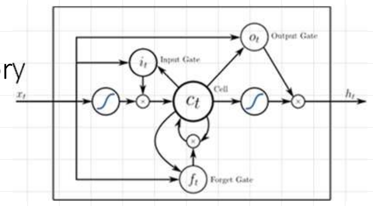
The forecast models can be represented as:

$$output_{t+1} \approx \mathrm{RNN}(NetworkState, input_t, output_t)$$

where *NetworkState* is the activation level of context layer nodes.

LSTM

Long Short-Term Memory (LSTM) neteorks are recurrent networks addtrained with Backprop. Through Time.



LSTM nets don't have neurons, but memory cells connected through the layers.

A cell is more complex than a normal neuron, it contains gates to manage internal state and output. Each gate works on the input sequence and decides whether to activate or not, thereby affecting state and output.

Three types of gate, each one with own weights to be learned:

- Forget Gate: decide which state information to drop.
- Input Gate: decide which input value affect the state.
- Output Gate: decide output value as a function of input and state