**ARIMA** **: Non-seasonal Autoregressive Integrated Moving Averages**

* **ARIMA** stands for **Autoregressive Integrated Moving Average Model**. It belongs to a class of models that explains a given time series based on its own past values -i.e.- its own lags and the lagged forecast errors. The equation can be used to forecast future values. Any ‘non-seasonal’ time series that exhibits patterns and is not a random white noise can be modeled with ARIMA models.
* So, **ARIMA**, short for **AutoRegressive Integrated Moving Average**, is a forecasting algorithm based on the idea that the information in the past values of the time series can alone be used to predict the future values.
* **ARIMA Models** are specified by three order parameters: (p, d, q),

where,

* + p is the order of the AR term
  + q is the order of the MA term
  + d is the number of differencing required to make the time series stationary

**The meaning of p**

* p is the order of the **Auto Regressive (AR)** term. It refers to the number of lags of Y to be used as predictors.

**3.2 The meaning of d**

* The term **Auto Regressive**’ in ARIMA means it is a linear regression model that uses its own lags as predictors. Linear regression models, as we know, work best when the predictors are not correlated and are independent of each other. So we need to make the time series stationary.
* The most common approach to make the series stationary is to difference it. That is, subtract the previous value from the current value. Sometimes, depending on the complexity of the series, more than one differencing may be needed.
* The value of d, therefore, is the minimum number of differencing needed to make the series stationary. If the time series is already stationary, then d = 0.

**3.3 The meaning of q**

* **q** is the order of the **Moving Average (MA)** term. It refers to the number of lagged forecast errors that should go into the ARIMA Model.

Un modello ARIMA è un modello Arma di ordine che è un processo che diventa stazionario dopo essere stato differenziato d volte,.

Si parte quindi da una rappresentazione ARMA(,) e usando polinomi AR e MA nell’operatore B (sopra esposto) otteniamo un processo integrato che segue un’equazione del tipo:

Per utilizzare la meglio il modello ARIMA bisogna prima di tutto identificare l’ordine oltre a e . Per farlo si differenzia la serie temporale finchè la visualizzazione non diventa stazionaria e finchè ACF e PACF assumono la forma tipica di modelli ARMA, in altre parole quando diventano zero o tendono velocemente a zero. Si procede come in tutti gli altri casi all’estimazione e validazione del modello.

FONTI:

Fundamentals of time series analysis, for the working data scientist (DRAFT) (pag 57-60) Marco Fattore;

<https://www.kaggle.com/prashant111/arima-model-for-time-series-forecasting>

**TBATS MODEL: Trigonometric seasonality, Box-Cox transformation,**[**ARMA**](https://medium.com/analytics-vidhya/arima-fc1f962c22d4)**errors, Trend and Seasonal components.**

TBATS is a forecasting method to model time series data. The main aim of this is to forecast time series with complex seasonal patterns using exponential smoothing

TBATS will consider various alternatives and fit quite a few models. It will consider models:

* with Box-Cox transformation and without it Box-Cox transformation.
* with considering Trend and without Trend.
* with Trend Damping and without Trend Damping.
* with ARIMA(p,q) and without ARMA(p,q) process used to model residuals.
* non-seasonal model.
* various amounts of harmonics used to model seasonal effects

The final model will be chosen using [**Akaike information criterion**](https://en.wikipedia.org/wiki/Akaike_information_criterion)**(AIC)**.

In particular, auto [ARIMA](https://medium.com/analytics-vidhya/arima-fc1f962c22d4) is used to decide if residuals need modeling and what p and q values are suitable.

I modelli TBATS cambiano il modo di parametrizzare le componenti stagionali. In particolar modo viene richiamato un caso speciale del teorema di Fourier che afferma che qualsiasi funzione periodica con periodo m può essere scritta come una somma infinita di funzioni sinusoidali del tipo:

Dove per gli elementi armonici hanno periodi progressivamente più corti , partendo dal periodo .

Riscrivendo l’espressione in funzione di



ed introducendo la componente stagionale, la cui somma delle componenti di periodo per è idealmente uguale a zero. Possiamo ora riscrivere la componente stagionale come somma infinita di armoniche dove e il modello come:

Immagine che contiene testo, orologio

Descrizione generata automaticamente

Dove n è il numero selezionato di componenti di Fourier con:

Immagine che contiene testo

Descrizione generata automaticamente

Con come strumento per modellizzare l’evoluzione di .

Il modello TBATS si ottiene infine aggiungendo le componenti stagionali all’errore ARMA.

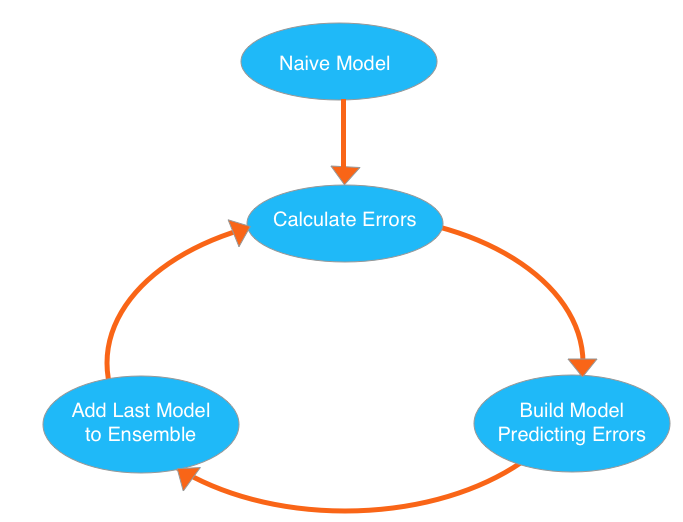
FONTI:

Fundamentals of time series analysis, for the working data scientist (DRAFT) (pag 81-83) Marco Fattore;

<https://medium.com/analytics-vidhya/time-series-forecasting-using-tbats-model-ce8c429442a9>

**XGBOOST**

XGBoost is an implementation of the **Gradient Boosted Decision Trees** algorithm .What is **Gradient Boosted Decision Trees**? We'll walk through a diagram.



We go through cycles that repeatedly builds new models and combines them into an **ensemble** model. We start the cycle by calculating the errors for each observation in the dataset. We then build a new model to predict those. We add predictions from this error-predicting model to the "ensemble of models."

To make a prediction, we add the predictions from all previous models. We can use these predictions to calculate new errors, build the next model, and add it to the ensemble.

There's one piece outside that cycle. We need some base prediction to start the cycle. In practice, the initial predictions can be pretty naive. Even if it's predictions are wildly inaccurate, subsequent additions to the ensemble will address those errors.

XGBoost has a few parameters that can dramatically affect your model's accuracy and training speed. The first parameters you should understand are:

### **n\_estimators and early\_stopping\_rounds**

**n\_estimators** specifies how many times to go through the modeling cycle described above.

In the [underfitting vs overfitting graph](http://i.imgur.com/2q85n9s.png), n\_estimators moves you further to the right. Too low a value causes underfitting, which is inaccurate predictions on both training data and new data. Too large a value causes overfitting, which is accurate predictions on training data, but inaccurate predictions on new data (which is what we care about).Typical values range from 100-1000, though this depends a lot on the **learning rate**.

The argument **early\_stopping\_rounds** offers a way to automatically find the ideal value. Early stopping causes the model to stop iterating when the validation score stops improving, even if we aren't at the hard stop for n\_estimators. It's smart to set a high value for **n\_estimators** and then use **early\_stopping\_rounds** to find the optimal time to stop iterating.

Since random chance sometimes causes a single round where validation scores don't improve, you need to specify a number for how many rounds of straight deterioration to allow before stopping. **early\_stopping\_rounds = 5** is a reasonable value.

### **learning\_rate**

Here's a subtle but important trick for better XGBoost models:

Instead of getting predictions by simply adding up the predictions from each component model, we will multiply the predictions from each model by a small number before adding them in. This means each tree we add to the ensemble helps us less. In practice, this reduces the model's propensity to overfit.

So, you can use a higher value of **n\_estimators** without overfitting. If you use early stopping, the appropriate number of trees will be set automatically.

In general, a small learning rate (and large number of estimators) will yield more accurate XGBoost models, though it will also take the model longer to train since it does more iterations through the cycle.

### **n\_jobs**

On larger datasets where runtime is a consideration, you can use parallelism to build your models faster. It's common to set the parameter **n\_jobs** equal to the number of cores on your machine. On smaller datasets, this won't help.

The resulting model won't be any better, so micro-optimizing for fitting time is typically nothing but a distraction. But, it's useful in large datasets where you would otherwise spend a long time waiting during the fit command.

XGBoost has a multitude of other parameters, but these will go a very long way in helping you fine-tune your XGBoost model for optimal performance.

FONTI:

<https://www.kaggle.com/dansbecker/xgboost>

**PROPHET**

Prophet is open source software released by Facebook’s Core Data Science team.

* So, [Prophet](https://facebook.github.io/prophet/) is the facebooks’ open source tool for making time series predictions.
* [Prophet](https://facebook.github.io/prophet/) decomposes time series data into trend, seasonality and holiday effect.
* **Trend** models non periodic changes in the time series data.
* **Seasonality** is caused due to the periodic changes like daily, weekly, or yearly seasonality.
* **Holiday effect** which occur on irregular schedules over a day or a period of days.
* **Error terms** is what is not explained by the model.

# **2. Advantages of Prophet**

[Prophet](https://facebook.github.io/prophet/) has several advantages associated with it.

* **1. Accurate and fast** - Prophet is accurate and fast. It is used in many applications across Facebook for producing reliable forecasts for planning and goal setting.
* **2. Fully automatic** - Prophet is fully automatic. We will get a reasonable forecast on messy data with no manual effort.
* **3. Tunable forecasts** - Prophet produces adjustable forecasts. It includes many possibilities for users to tweak and adjust forecasts. We can use human-interpretable parameters to improve the forecast by adding our domain knowledge.
* **4. Available in R or Python** - We can implement the Prophet procedure in R or Python.
* **5. Handles seasonal variations well** - Prophet accommodates seasonality with multiple periods.
* **6. Robust to outliers** - It is robust to outliers. It handles outliers by removing them.
* **7. Robust to missing data** - Prophet is resilient to missing data.

FONTI:

<https://www.kaggle.com/code/prashant111/tutorial-time-series-forecasting-with-prophet/notebook> ;

<https://www.kaggle.com/code/prashant111/tutorial-time-series-forecasting-with-prophet/notebook>