Member-only story

Why Backtesting Matters and How to Do It Right

How do we know if our forecasting model is accurate and reliable and evaluate its performance on unseen data? This is where backtesting comes in.

Davide Burba · Follow

Published in Towards Data Science · 7 min read · Jul 18

Image by Lorenzo Cafaro from Pixabay

What is Backtesting?

To evaluate the performance of a forecasting model, we use a procedure called backtesting (also known as **time-series cross-validation**). Backtesting is essentially a way of testing how a model would have performed if it had been used in the past.

How does it work?

To backtest a time-series forecasting model, we start by splitting the data into two parts: a training set and a validation set (sometimes also called a test set, but we'll clarify the difference in the next sections). The training set is used to train the model, while the test set is used to evaluate how well the model performs on unseen data. Once the model has been trained, you can then use it to make predictions on the test set. You can compare these predictions to the actual values to see how well the model performs.

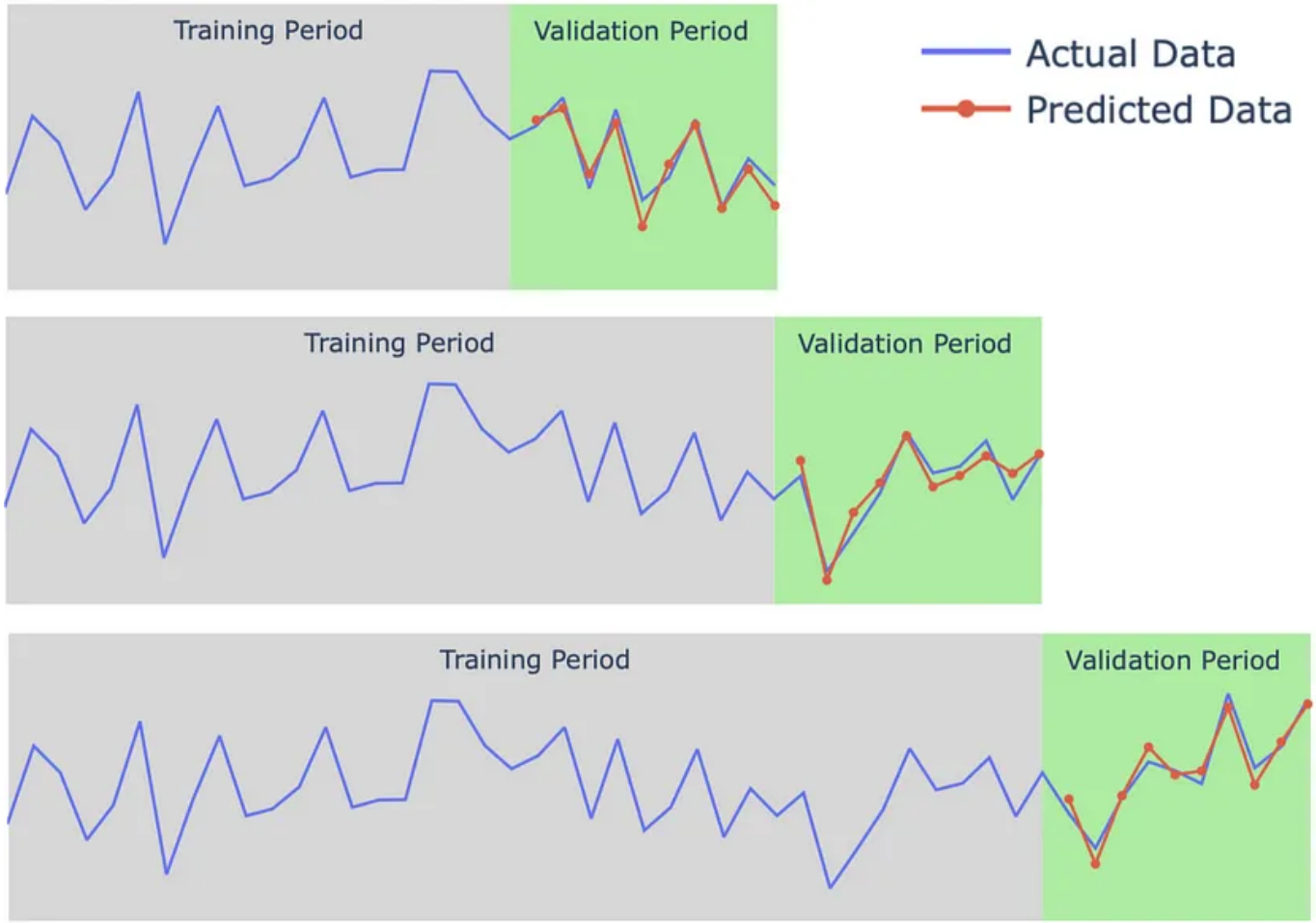
How do we measure the performance of a model?

There are several metrics that can be used to evaluate the performance of a time-series forecasting model, such as Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE). These metrics measure how close the predicted values are to the actual values.

The procedure is typically repeated multiple times, allowing us to:

- Have a good estimate of the model performance
- Visualize the evolution of the performance in time

Here below we show a graphical representation of the backtesting process, using 3 splits:



Backtesting with extending window. Image by author.

In the figure above we show only 3 non-overlapping validation periods. However, nothing prevents us to use more partially overlapping windows, potentially one per time-step.

Why is backtesting important?

If the model performs well on the test set, you can then have more confidence in its ability to make accurate predictions in the future. By backtesting a time-series forecasting model, you can gain insights into how well it performs on historical data and make more informed decisions about how to use it in the future.

How about using backtesting to improve our model?

Backtesting can also be used to improve our model, for example with hyper-parameters tuning or feature selection. In this case, since we usually end up by selecting the best performing combinations, our metric estimates are likely to be over-optimistic. If we want to have an unbiased performance estimate, we should perform two backtesting in two separate consecutive periods:

1. The first backtesting is used to tune hyper-parameters, select the best features, and in general make all the choices necessary to build our "final" model. The splits used for evaluating predictions in this period are known as **validation sets**.
2. The second backtesting is used to have a performance estimate of the final selected model. It should be performed on the period after the

one used for validation. The splits used for evaluating predictions in this period are known as **test sets**.

Backtesting variants

Extending or fixed training window?

In the previous figure, we can see that the size of the training window is extending. An alternative would be to use a fixed size time-window for the training set, as shown in the next figure.

Backtesting with fixed window. Image by author.

Using a fixed window has the drawback that the model uses less data to train, however it might be convenient if the data is very dynamic and therefore the most recent data is much more informative than older data. In general, the window size can be seen as an hyper-parameter that can be tweaked via backtesting.

Training cutoff vs feature cutoff

An important thing to keep in mind is that data used to train the model and data used to build the input features are two separate concepts.
In principle, we could have as many splits as the number of time-steps in the data. However, re-training the model for each time-step might be time-consuming and computationally intensive. An alternative that is often used in practice is to train a model less-frequently, let's say every N time steps, and do predictions with the most recent model at every time-step. The number N can also be tuned via backtesting.

Finally, note that in this article we are considering a single time-series for simplicity. However, the same concepts can be applied to models that make predictions for multiple time-series (if you are interested in this topic, you can check this article).

Example: Airline Passengers

Image by Lars Nissen from Pixabay

In this section we show a simple Python implementation of backtesting, using the Air Passengers dataset, which is available on Darts under the Apache 2.0 License.

Let's start by importing the necessary libraries.

```
import pandas as pd
import plotly.graph_objects as go
from lightgbm import LGBMRegressor
```

Data preparation

The dataset consists in monthly totals of international airline passengers from 1949 to 1960. Let's load and prepare the data.

```
# Load data.
data = pd.read_csv('https://raw.githubusercontent.com/unit8co/darts/master/datas
# Rename columns.
data = data.rename(columns = {"Month": "time", "#Passengers": "passengers"})
# Set time to datetime.
data.time = pd.to_datetime(data.time)
# Set time as index.
data = data.set_index("time")
```

Let's have a quick look at the data:

```
# Let's visualize the data.
def show_data(data,title=""):
```

```
    trace = [go.Scatter(x=data.index,y=data[c],name=c) for c in data.columns]
    go.Figure(trace,layout=dict(title=title)).show()


show_data(data,"Monthly totals of international airline passengers")
```

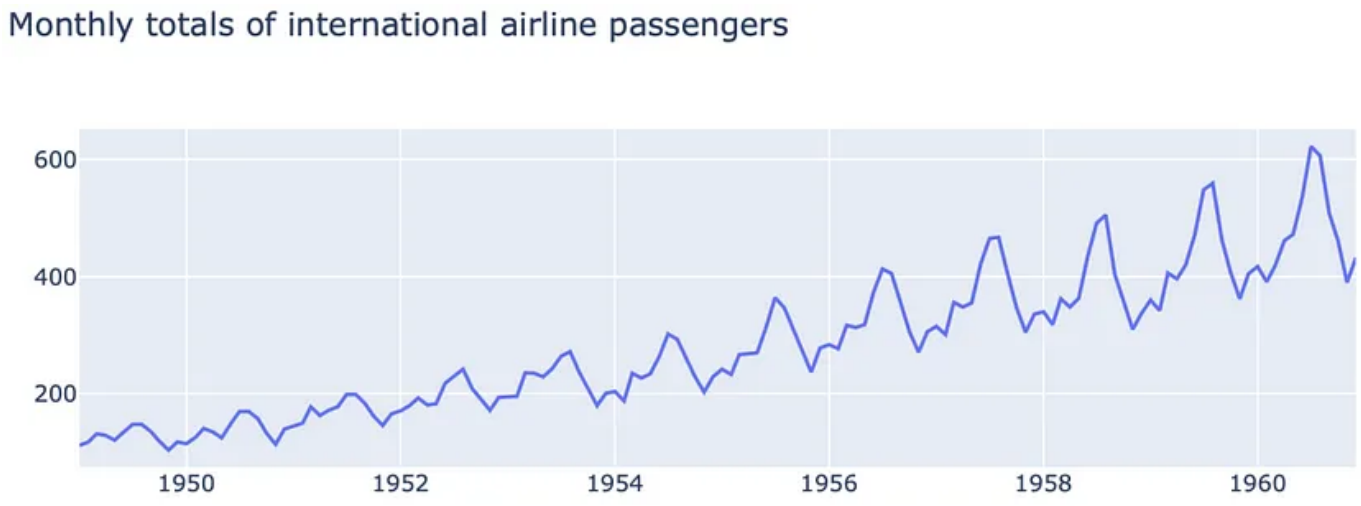


Image by author.

We can see that the data exhibits a strong increasing trend and yearly seasonality.

Data engineering

Let's predict the value of the next month based on:

- A few lagged values: the previous 3 months, plus 6/12/24 months ago
- The current month (as a categorical feature, to help with seasonality)

```
def build_target_features(data, lags, horizon=1):
    # Build lagged features.
    feat = pd.concat(
        [
            data[["passengers"]].shift(lag).rename(
                columns={"passengers": f"lag_{lag}"}
            )
            for lag in lags
        ],
        axis=1,
    )

    # Build month feature.
    feat["month"] = [f"M{m}" for m in data.index.month]
    feat["month"] = feat["month"].astype("category")
    # Build target at horizon.
    targ = data["passengers"].shift(-horizon).rename(f"horizon_{horizon}")
    # Drop missing values generated by lags/horizon.
    idx = ~(feat.isnull().any(axis=1) | targ.isnull())
    feat = feat.loc[idx]
    targ = targ.loc[idx]
    return targ, feat


# Build targets and features.
target, features = build_target_features(
    data,
```

```
    lags=[0,1,2,5,11,23],
    horizon=1,
)
```

Backtesting implementation

Let's implement a simple yet generic backtesting function. The function receives as input the model and the data, plus two parameters for the start training window and the frequency for re-training. We use an extending training window, and we make predictions at every time-step after the start window.

```
def run_backtest(
    model,
    target,
    features,
    start_window = 10,
    retrain_frequency = 6,
):
    """Simple backtesting implementation.

    Args:
        model: A model with fit and predict methods.
        targets: Series with the target in chronological order.
        features: Dataframe with the features in chronological order.
        start_window: The initial window to train a model.
        retrain_frequency: How often to retrain the model.

    Return:
```

```
        A dataframe with the validation target and prediction.
    """

    # Sanity check on shape
    assert features.shape[0] == target.shape[0]

    all_preds = []
    all_targets = []
    last_timestep = start_window

    while last_timestep < len(target):
        # Split train/valid
        targ_train = target.iloc[:last_timestep]
        feat_train = features.iloc[:last_timestep]
        targ_valid = target.iloc[last_timestep:last_timestep+1]
        feat_valid = features.iloc[last_timestep:last_timestep+1]

        # Train the model
        if last_timestep==start_window or last_timestep % retrain_frequency == 0
            model.fit(feat_train,targ_train)
        # Predict on valid set
        pred_valid = model.predict(feat_valid)
        # Save the output
        all_preds.append(pred_valid[0])
        all_targets.append(targ_valid)

        # Process next timestep
        last_timestep += 1

    # Format output
    output = pd.concat(all_targets).rename("target").to_frame()
    output["prediction"] = all_preds
    return output
```

The code above could be extended in several ways. For example you could allow to use a fixed training window, or make it more efficient by using vectorization and/or parallelization. However we suggest to start simple and add complexity only if necessary.

We can now apply the backtesting function to our data, and check the backtesting performances.

```
# Apply run_backtest to our data.
output_backtest = run_backtest(
    LGBMRegressor(min_child_samples=1, objective="mae"),
    target,
    features,
    start_window = 36,
    retrain_frequency = 6,
)
# First, let's visualize the data.
show_data(output_backtest,"Backtested predictions at horizon 1")
# Then, let's compute the MAE.
MAE = abs(output_backtest.prediction - output_backtest.target).mean()
print(f"MAE: {MAE:.1f}")
```

which gives a MAE of 25.6 and the figure below.

Image by author.

The full code used is this example is available here.

Conclusion

Backtesting is a crucial step in time-series forecasting that allows you to evaluate your model's accuracy and reliability on unseen data. By following some best practices and using appropriate methods for backtesting, you can improve your model's performance and avoid common pitfalls.

Enjoyed this article? Checkout my other ones and follow me for more! Click here to read unlimited articles and support me at no additional cost for you ❤️

Time Series Analysis | Forecasting | Machine Learning | Data Science | Backtesting

Written by Davide Burba

Follow

344 Followers · Writer for Towards Data Science

Developer based in Switzerland. I write about Software, Forecasting, Machine Learning.
linkedin.com/in/davide-burba

More from Davide Burba and Towards Data Science

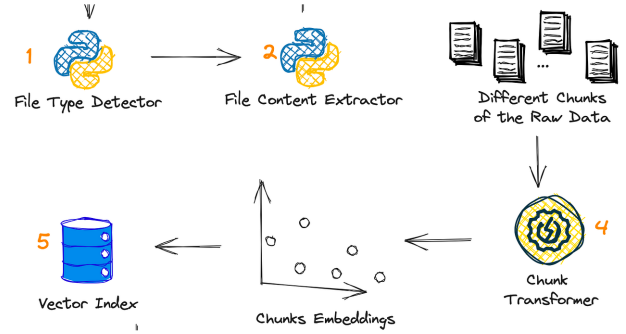
Davide Burba in Towards Data Science

Forecast Multiple Horizons: an Example with Weather Data

Predict precipitations in Switzerland by using the forecasting horizon as a feature.

✦ · 8 min read · Aug 6

137 1



Zoumana Keita in Towards Data Science

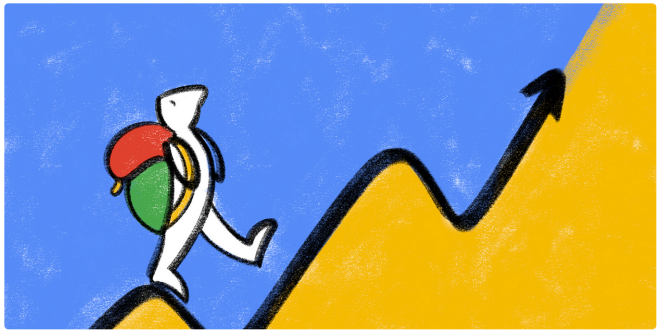
How to Chat With Any File from PDFs to Images Using Large...

Complete guide to building an AI assistant that can answer questions about any file

✦ · 9 min read · Aug 5

1.3K 12

Bex T. in Towards Data Science



Davide Burba in Towards Data Science

130 ML Tricks And Resources Curated Carefully From 3 Years...

Each one is worth your time

· 48 min read · Aug 1

3.4K 12

Forecasting API: an Example with Django and Google Trends

Build a web application to predict the evolution of Google Trends.

· 14 min read · Aug 1

94

See all from Davide Burba

See all from Towards Data Science

Recommended from Medium

Babak Abbaschian

Dynamic Pricing using Machine Learning

Have you ever searched for a plane ticket and checked the price to buy the ticket later, only...

16 min read · Jun 6

64

Daniel Frees in Towards Data Science

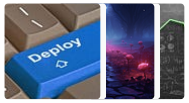
Scaling Agglomerative Clustering for Big Data

Learn how to use Reciprocal Agglomerative Clustering to power hierarchical clustering o...

8 min read · 6 days ago

294 2

Lists



Predictive Modeling w/ Python

20 stories · 343 saves

Practical Guides to Machine Learning

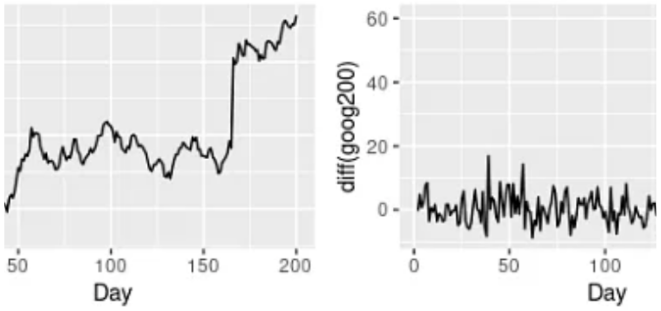
10 stories · 384 saves

Natural Language Processing

569 stories · 191 saves

New_Reading_List

174 stories · 93 saves



Krishnakanth Naik Jarapala

Crash Course in Time Series Analysis and Forecasting

by Krishnakanth Naik Jarapala, Venkata Bhargavi Sikhakolli

13 min read · Apr 9

55

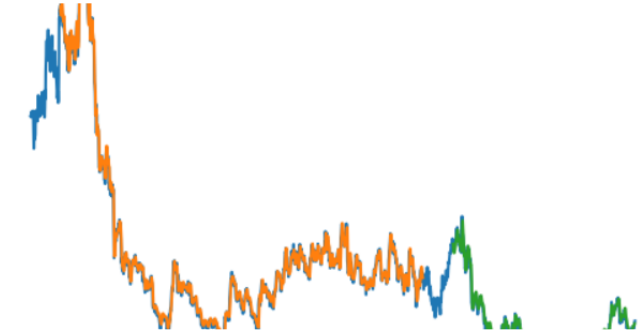
Mouna Labiadh in DataNess.AI

Understanding Temporal Fusion Transformer

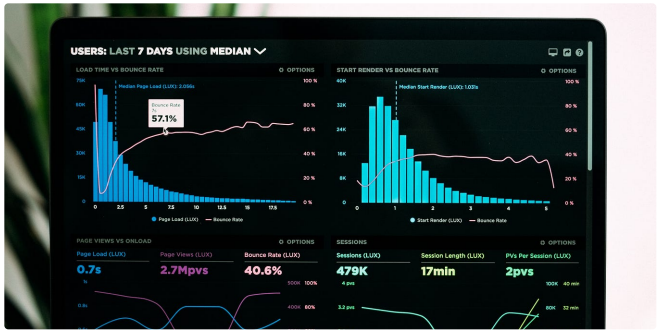
Breakdown of Google's Temporal Fusion Transformer (2021) for interpretable multi-...

6 min read · Apr 12

157 1



Prajjwal Chauhan



Profitsub

Stock Prediction and Forecasting Using LSTM(Long-Short-Term-...

In an ever-evolving world of finance, accurately predicting stock market...

6 min read · Jul 8

183 7

The Geometric Mean Combined Score (GMCS): A New Measure fo...

In the fast-paced world of finance, traders and quants are always on the lookout for the next...

4 min read · 6 days ago

2

See more recommendations