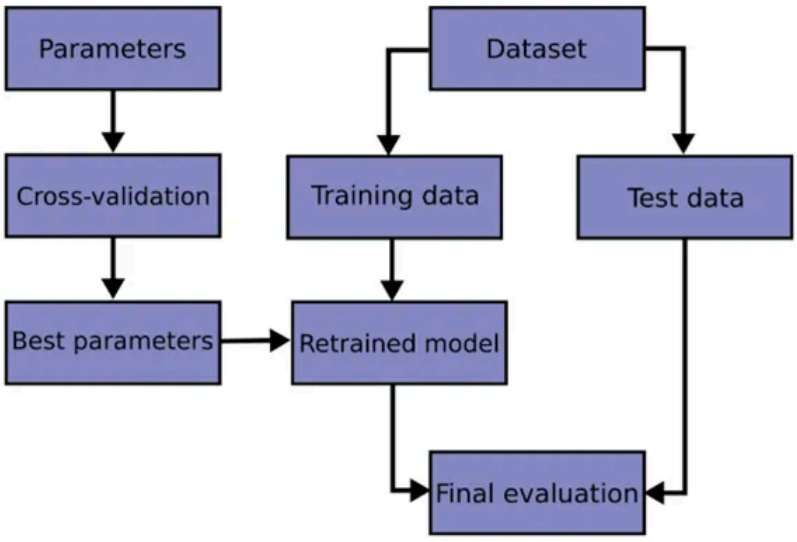
Cross Validation in Time Series

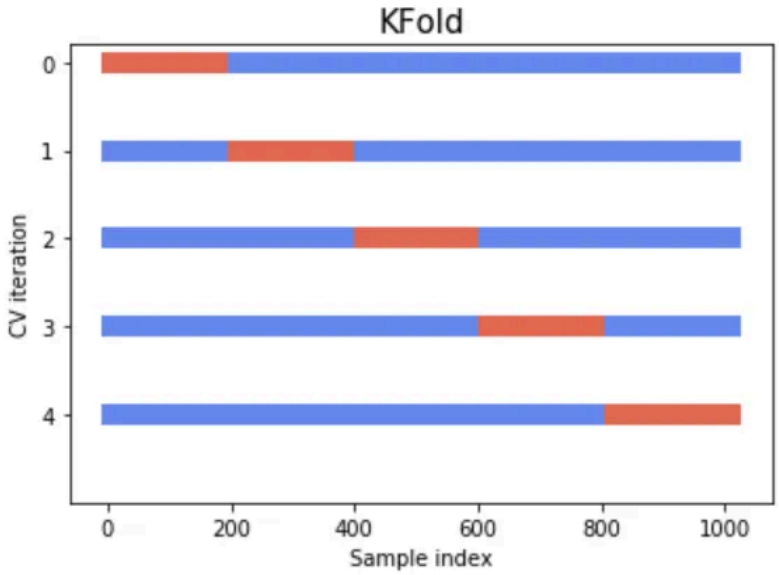
Soumya Shrivastava · Follow
9 min read · Jan 14, 2020

Cross Validation:

When you build your model, you need to evaluate its performance. Cross-validation is a statistical method that can help you with that. For example, in K-fold-Cross-Validation, you need to split your dataset into several folds, then you train your model on all folds except one and test model on remaining fold. You need to repeat this steps until you tested your model on each of the folds and your final metrics will be average of scores obtained in every fold. This allows you to prevent overfitting, and evaluate model performance in a more robust way than simple train-test.



This is the concept at the base of Cross Validation. The most accepted technique in the ML world consists in randomly picking samples out of the available data and split it in train and test set. Well to be completely precise the steps are generally the following:



1. Split randomly data in train and test set.
2. Focus on train set and split it again randomly in chunks (called folds).
3. Let's say you got 10 folds; train on 9 of them and test on the 10th.
4. Repeat step three 10 times to get 10 accuracy measures on 10 different and separate folds.
5. Compute the average of the 10 accuracies which is the final reliable number telling us how the model is performing.

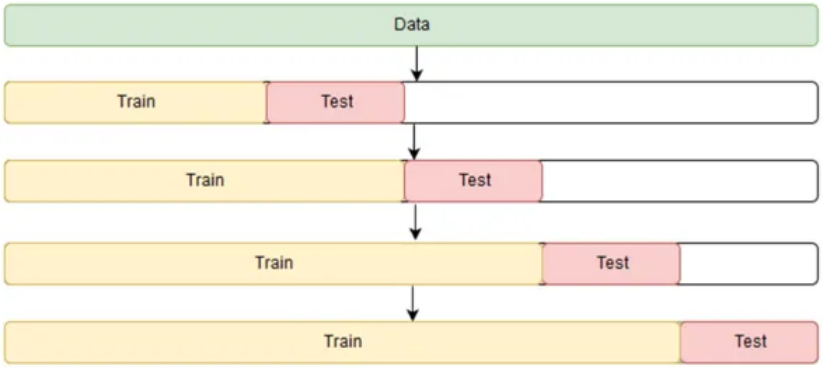
Why can't we use this process in Time Series:

In the case of time series, the cross-validation is not trivial. We cannot choose random samples and assign them to either the test set or the train set because it makes no sense to use the values from the future to forecast values in the past. In simple word we want to avoid future-looking when we train our model. There is a temporal dependency between observations, and we must preserve that relation during testing.

Cross Validation on Time Series:

The method that can be used for cross-validating the time-series model is cross-validation on a rolling basis. Start with a small subset of data for training purpose, forecast for the later data points and then checking the accuracy for the forecasted data points. The same forecasted data points are then included as part of the next training dataset and subsequent data points are forecasted.

To make things intuitive, here is an image for same:



The idea of cross-validation should be more straightforward to grasp when we look at an example. Imagine that we have only 5 observations in our cross-validation set and we want to perform 4-fold cross-validation.

Here is the dataset: [1, 2, 3, 4, 5]

What we need to do is to create 4 pairs of training/test sets that follow those two rules:

- every test set contains unique observations
- observations from the training set occur before their corresponding test set

There is only one way to generate such pairs from my dataset. As a result, I get 4 pairs of training/test sets:

- Training: [1] Test: [2]
- Training: [1, 2] Test: [3]
- Training: [1, 2, 3] Test: [4]
- Training: [1, 2, 3, 4] Test: [5]
- Compute the average of the accuracies of the 4 test fold.

Example:

```
import numpy as np
from sklearn.model_selection import TimeSeriesSplit
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([1, 2, 3, 4, 5, 6])
 tscv = TimeSeriesSplit()
print(tscv)
TimeSeriesSplit(max_train_size=None, n_splits=3)
for train_index, test_index in tscv.split(X):
print("TRAIN:", train_index, "TEST:", test_index) X_train, X_test =
```

```
X[train_index], X[test_index]
y_train, y_test = y[train_index], y[test_index]
```

```
TimeSeriesSplit(max_train_size=None, n_splits=3)
TRAIN: [0 1 2] TEST: [3]
TRAIN: [0 1 2 3] TEST: [4]
TRAIN: [0 1 2 3 4] TEST: [5]
```

Output

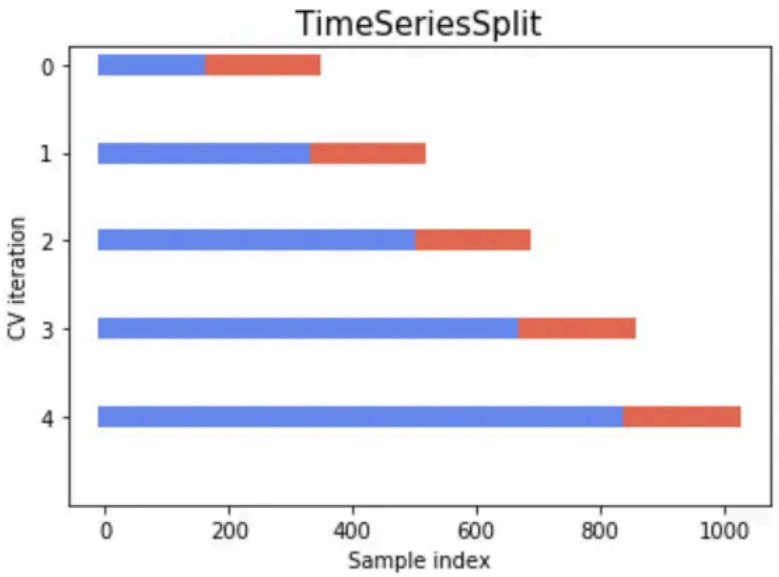
In this blog, we shall explore two more techniques for performing cross-validation; time series split cross-validation and blocked cross-validation, which is carefully adapted to solve issues encountered in time series forecasting.

- Time Series Split Cross-Validation
- Blocked Cross-Validation

Time Series Split Cross-Validation

Consider we have a data of student's performance for one year(January to December)

The best way to grasp the intuition behind blocked and time series splits is by visualizing them. The three split methods are depicted in the below diagram. The horizontal axis is the training set size while the vertical axis represents the cross-validation iterations. The folds used for training are depicted in blue and the folds used for validation are depicted in orange. You can intuitively interpret the horizontal axis as time progression line since we haven't shuffled the dataset and maintained the chronological order.



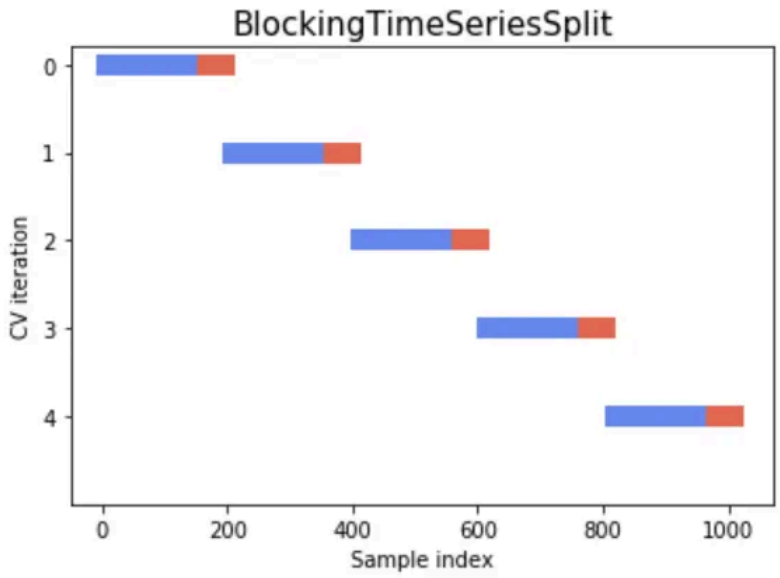
The idea for time series splits is to divide the training set into two folds at each iteration on condition that the validation set is always ahead of the training set. At the first iteration, one trains the candidate model on the student performance from January to March and validates on April's data, and for the next iteration, train on data from January to April, and validate on May's data, and so on to the end of the training set. This way dependence is respected.

Example:

After generating the training/test sets, we are going to fit an ARMA model and make a prediction. We will store the root mean squared error of the prediction in the "rmse" array. After the last test, we will calculate the average error.

```
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm

tscv = TimeSeriesSplit(n_splits = 4)
rmse = []

for train_index, test_index in tscv.split(cross_validation):
    cv_train, cv_test = cross_validation.iloc[train_index],
cross_validation.iloc[test_index]

    arma = sm.tsa.ARMA(cv_train, (2,2)).fit(disp=False)

    predictions = arma.predict(cv_test.index.values[0],
cv_test.index.values[-1])
    true_values = cv_test.values
    rmse.append(sqrt(mean_squared_error(true_values, predictions)))

print("RMSE: {}".format(np.mean(rmse)))
```

Blocked Cross-Validation



However, this may introduce leakage from future data to the model. The model will observe future patterns to forecast and try to memorize them. That's why blocked cross-validation was introduced. It works by adding margins at two positions. The first is between the training and validation folds in order to prevent the model from observing lag values which are used twice, once as a regressor and another as a response. The second is between the folds used at each iteration in order to prevent the model from memorizing patterns from an iteration to the next.

Nested CV Methods for a Time Series

We suggest two methods for nested CV with data from a single time series. We'll deal with the scenario where we have multiple days of data from a medical patient/participant:

1. Predict Second Half
2. Day Forward-Chaining

Predict Second Half

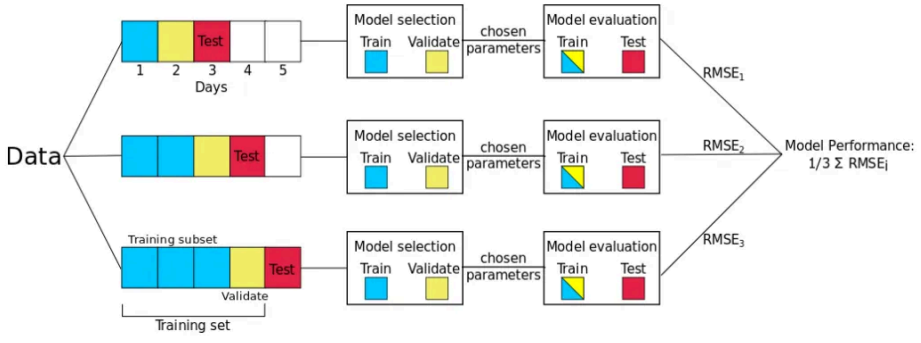
The first type, Predict Second Half, is the "base case" of nested CV with only 1 train/test split. The advantage to this is that this method is easy to implement; however, it still suffers from the limitation of an arbitrarily-chosen test set. The first half of data (split temporally) is assigned to the training set and the latter half becomes the test set. The validation set size can vary based on the given problem (e.g. 1 day of data in our example), but

it is important to ensure that the validation set is chronologically subsequent to the training subset.

Predict Second Half

Day Forward-Chaining

One shortcoming of the Predict Second Half nested cross-validation method is that the arbitrary choice of the hold-out test set can produce biased estimates of prediction error on an independent test set. In order to produce a better estimate of model prediction error, a common approach is to create many train/test splits and average the errors over all the splits. The technique we use, called Day Forward-Chaining is based on a method called forward-chaining and rolling-origin-recalibration evaluation. Using this method, we successively consider each day as the test set and assign all previous data into the training set. As an example, if our dataset has five days, then we would produce three different training and test splits, as shown in below Figure . Note that in this example we have three splits versus five because we need to ensure that there is at least one day of training and validation data available. This method produces many different train/test splits and the error on each split is averaged in order to compute a robust estimate of the model error.



Day Forward-Chaining

Nested Cross-Validation with Multiple Time Series

Now that we have two methods for splitting a single time series, we discuss how to handle a dataset with multiple different time series. Again, we use two types:

Regular

For “regular” nested cross-validation, the basic idea of how the train/validation/test splits are made is the same as before. The only change is that the splits now contain data from each participant in our dataset. For instance, if there are two participants, Participant A and B, the training set would contain the data from the first half of days from Participant A and the data from the first half of days from Participant B. Likewise, the testing set would contain the second half of days for each participant.

Population-Informed

For “population-informed nested cross-validation” we take advantage of the **independence** between different participants’ data. This allows us to break the strict temporal ordering, at least between individuals’ data (it is still necessary within an individual’s data). Because of this independence, we can slightly modify the regular nested cross-validation algorithm. Now the test and validation sets only contain data from one participant, say Participant A, and *all* data from all other participants in the dataset are allowed in the training set. See Figure 1 for a visual of how this works for population-informed Day Forward-Chaining nested cross-validation. The figure shows that Participant A’s 18*th* day is the testing set (colored red), the previous three days are the validation set (colored yellow), and the training set (colored green) contains all the previous data from Participant A, as well as *all* the data from the rest of the participants (*B*, *C*, *D*, and *E* in this example). The important thing to emphasize is that there is no data leakage from using “future” observations from the other participants precisely due to the independence of these participants’ time series.

Population-Informed Day Forward-Chaining, where, in addition to Day Forward-Chaining method (Figure 1)for Participant A, we also allow all other participants’ data to be in the training set (Figure 2). Note that the grey bars indicate when the participant was sleeping.

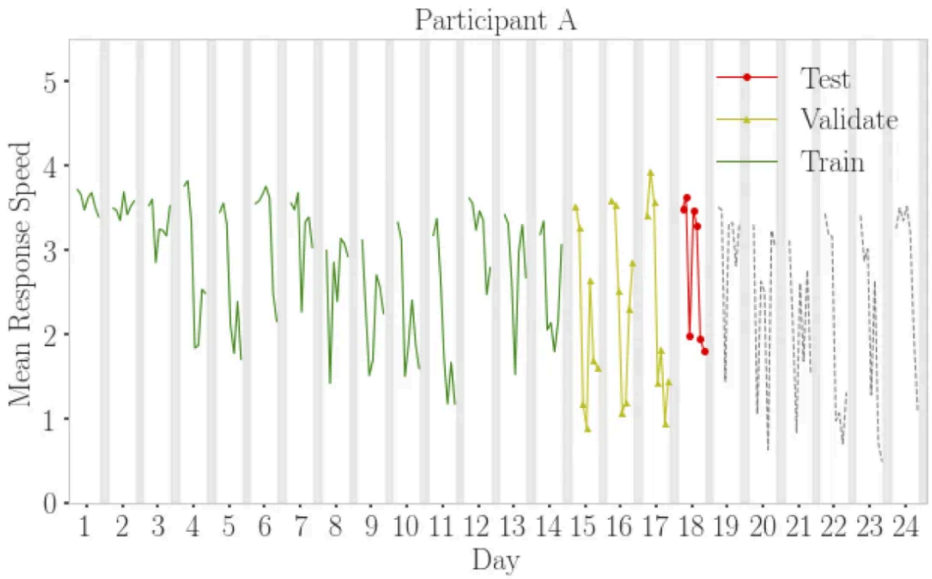


Figure 1

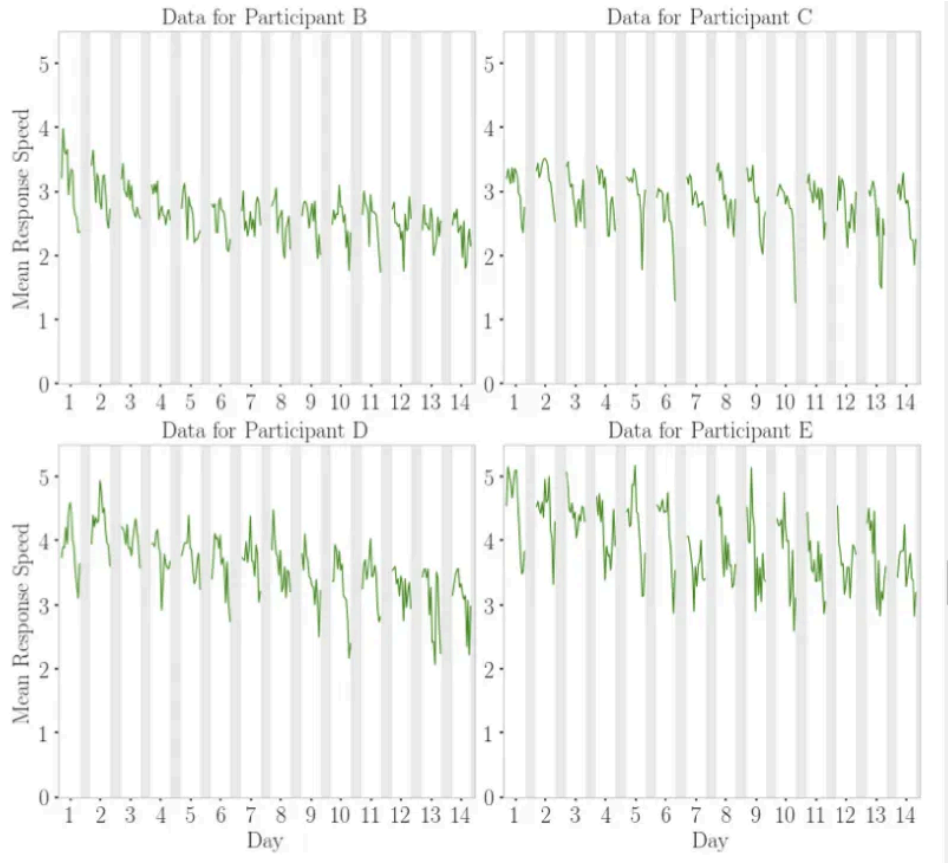


Figure 2

Other Important Nested Cross-Validation Considerations

Finally, we summarize pros and cons of the different nested cross-validation methods, particularly in regards to computation time and bias of independent test set error estimates. The number of splits assumes a dataset containing k participants and *d* days of data for each participant.

Method	Split	Pros	Cons
Time-series Split	k	• More splits • Can inspect how model fares on different days	• May create leakage from future data to the model
Blocked Cross Validation	k	• More splits • Solves data leakage issue	•Maybe very computationally expensive
Regular Predict Second Half	1	• Easy to implement • Produces a single model •Computationally inexpensive	• Arbitary test choice could produce biased estimate
Regular Day Forward Chaining	k	• More splits • Can inspect how model fares on different days	• Requires consistent number of days in data for each participant •Multiple models
Population Data Day Forward Chaining	k	• Most unbiased estimate of error versus other methods • Can inspect how model fares on different days	• Computationally expensive • Multiple models

Summary:

The objective of this article was to get the basic understanding on how to perform Cross Validation on time series data. We have seen different types of cross validation techniques like Nested Cross Validation, Time Series Split Cross Validation, Blocked Cross Validation. Also, we addressed how to handle multiple *independent* time series.

References:

URL: https://robjhyndman.com/hyndsight/tscv/

URL: https://otexts.com/fpp3/tscv.html

Machine Learning

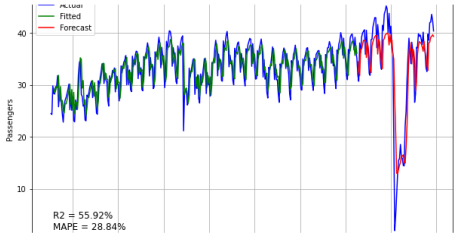
Written by Soumya Shrivastava

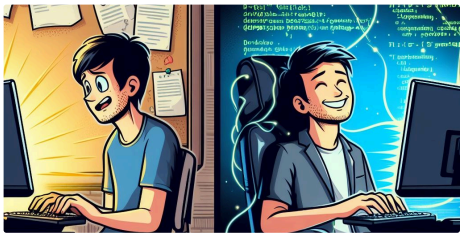
121 Followers

Data Science student at Praxis Business School

Follow

Recommended from Medium





Seyed Mousavi

How to fix a common mistake in LSTM time series forecasting

May 2 · 468 · 4

Abhay Parashar in The Pythoneers

17 Mindblowing Python Automation Scripts I Use Everyday

Scripts That Increased My Productivity and Performance

5d ago · 7.1K · 69

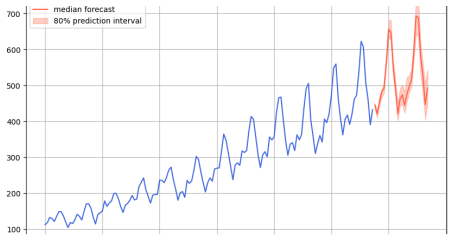
Lists

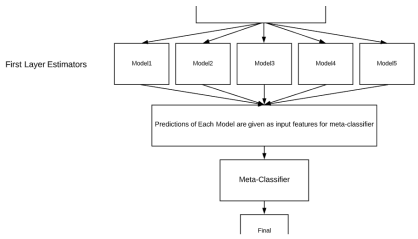
Predictive Modeling w/ Python
20 stories · 1476 saves

Practical Guides to Machine Learning
10 stories · 1802 saves

Natural Language Processing
1667 stories · 1242 saves

The New Chatbots: ChatGPT, Bard, and Beyond
12 stories · 450 saves





Manuele Caddeo

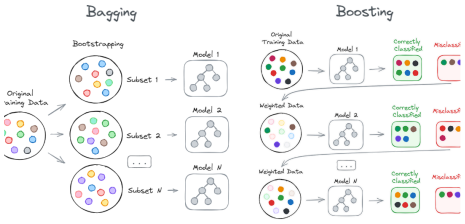
Time Series Forecasting with TimesFM

TimesFM is a pretrained model developed by Google Research for forecasting time series...

Kevin Akbari

Mastering Model Stacking: A Comprehensive Guide with Pytho...

In the realm of machine learning, ensemble methods play a pivotal role in enhancing...

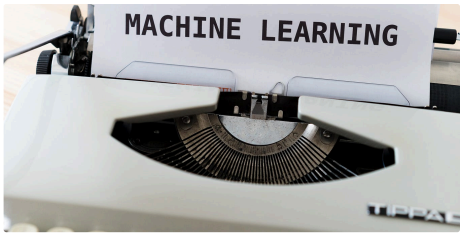


Thomas A Dorfer in Towards AI

Bagging vs. Boosting: The Power of Ensemble Methods in Machine...

How to maximize predictive performance by creating a strong learner from multiple weak...

Jun 16, 2023 562 3



Siddharth Ghosh

Hyper Parameter Tuning Techniques—Grid Search,...

As data scientists and machine learning engineers, we are captivated by high-...

Apr 13 2 1

See more recommendations