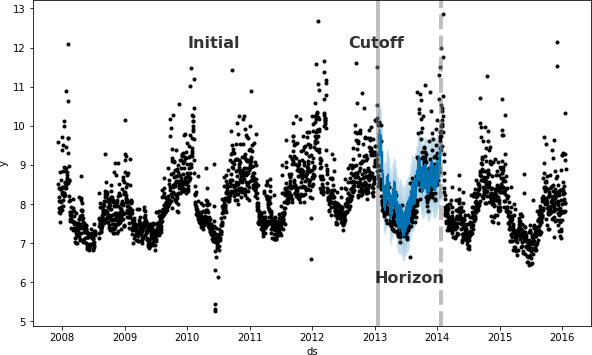
Diagnostics / 诊断验证方法

# Cross validation / 交叉验证

**Prophet includes functionality for time series cross validation to measure forecast error using historical data. Prophet包括时间序列交叉验证的功能，用历史数据衡量预测误差。**

This is done by selecting cutoff points in the history, and for each of them fitting the model using data only up to that cutoff point. We can then compare the forecasted values to the actual values. 这是通过选择历史上的分界点来实现的，对于每一个分界点，只用到该分界点的数据来拟合模型。然后我们可以将预测值与实际值进行比较。

This figure illustrates a simulated historical forecast on the Peyton Manning dataset, where the model was fit to an initial history of 5 years, and a forecast was made on a one year horizon. 该图显示了对佩顿-曼宁数据集的模拟历史预测，其中模型被拟合到5年的初始历史，并在一年的范围内进行预测。



[The Prophet paper](https://peerj.com/preprints/3190.pdf) gives further description of simulated historical forecasts.

This cross validation procedure can be done automatically for a range of historical cutoffs using the **cross\_validation** function. 这个交叉验证过程可以使用**cross\_validation函数**针对一系列历史截止日期自动完成。

We specify the forecast horizon (**horizon**), and then optionally the size of the initial training period (**initial**) and the spacing between cutoff dates (**period**). 我们指定**预测范围（horizon）**，然后可以选择**初始训练期（initial）的大小**和**截止日期的间隔（period）**。

**By default, the initial training period is set to three times the horizon, and cutoffs are made every half a horizon.在默认情况下，initial训练器被设置为horizon的三倍并且每半个horizon设置一个cutoff**

**对于cross\_validation()函数的解读**

**参数**

**Initial:表示开始时刻，即在此时刻前的数据作为初始的训练集**

**Period:表示设置cutoff的间隔，每隔多少时间设置一个cutoff**

**Horizon:每次从cutoff预测多少天**

**在设置以后，训练集会滚动增加，每次增加一个horizon**

The output of **cross\_validation** is a dataframe with the true values y and the out-of-sample forecast values **yhat**, at each simulated forecast date and for each cutoff date. In particular, a forecast is made for every observed point between **cutoff** and **cutoff + horizon**. This dataframe can then be used to compute error measures of **yhat** vs. **y**. **Cross\_validation**的输出是一个dataframe，其中包括每个模拟预测日期和每个截止日期的真实值**y**和样本外预测值**yhat**。特别是，对截止日期和截止日期+水平线之间的每个观察点进行预测。然后，这个dataframe可以用来计算**yhat(yhat\_upper和yhat\_lower)**与**y**的误差测量。

Here we do cross-validation to assess prediction performance on a horizon of 365 days, starting with 730 days of training data in the first cutoff and then making predictions every 180 days. On this 8 year time series, this corresponds to 11 total forecasts. 这里我们使用交叉验证来评估365天horizon范围内的预测性能，从第一个分界线的730天训练数据开始，然后每180天进行一次预测。在这个8年的时间序列中，这相当于11次总的预测。

1. # Python
2. from prophet.diagnostics import cross\_validation
3. df\_cv = cross\_validation(m, initial='730 days', period='180 days', horizon = '365 days')
4. # Python
5. df\_cv.head()



**0**2010-02-16 8.959678 8.470035 9.451618 8.242493 2010-02-15

**1**2010-02-17 8.726195 8.236734 9.219616 8.008033 2010-02-15

**2**2010-02-18 8.610011 8.104834 9.125484 8.045268 2010-02-15

**3**2010-02-19 8.532004 7.985031 9.041575 7.928766 2010-02-15

**4**2010-02-20 8.274090 7.779034 8.745627 7.745003 2010-02-15

**cutoff**

**y**

**yhat\_upper**

**yhat\_lower**

**yhat**

**ds**

Custom cutoffs can also be supplied as a list of dates to the cutoffs keyword in the cross\_validation function in Python and R. For example, three cutoffs six months apart, would need to be passed to the cutoffs argument in a date format like: 在Python和R的cross\_validation函数中，自定义截止日期也可以作为日期列表提供给cutoffs关键字。例如，三个截止日期相隔六个月，需要以日期格式传递给cutoffs参数：

1 # Python

2 cutoffs = pd.to\_datetime(['2013-02-15', '2013-08-15', '2014-02-15'])

3 df\_cv2 = cross\_validation(m, cutoffs=cutoffs, horizon='365 days')

The **performance\_metrics** utility can be used to compute some useful statistics of the prediction performance (**yhat, yhat\_lower**, and **yhat\_upper** compared to **y**), as a function of the distance from the cutoff (how far into the future the prediction was). The statistics computed are mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), mean absolute percent error (MAPE), median absolute percent error (MDAPE) and coverage of the **yhat\_lower** and **yhat\_upper** estimates. These are computed on a rolling window of the predictions in **df\_cv** after sorting by horizon (**ds** minus **cutoff**). By default 10% of the predictions will be included in each window, but this can be changed with the rolling\_window argument.

**performance\_metrics**工具可以用来计算预测性能的一些有用的统计数据（yhat、yhat\_lower和yhat\_upper与y相比），作为与截止点（预测在未来多远）的距离的函数。

计算的统计数据是平均平方误差（MSE）

平均平方根误差（RMSE）

平均绝对误差（MAE）

平均绝对百分比误差（MAPE）

中位绝对百分比误差（MDAPE）

以及yhat\_lower和yhat\_upper估计的覆盖率。

这些都是在按水平线（ds减去cutoff）排序后，对df\_cv中预测的滚动窗口进行计算。默认情况下，每个窗口将包括10%的预测值，但这可以通过rolling\_window参数改变。

1. # Python
2. from prophet.diagnostics import performance\_metrics 3 df\_p = performance\_metrics(df\_cv)

4 df\_p.head()



**0**37 days 0.493764 0.702683 0.504754 0.058485 0.049922 0.058774 0.674052

**1**38 days 0.499522 0.706769 0.509723 0.059060 0.049389 0.059409 0.672910

**2**39 days 0.521614 0.722229 0.515793 0.059657 0.049540 0.060131 0.670169

**3**40 days 0.528760 0.727159 0.518634 0.059961 0.049232 0.060504 0.671311

**4**41 days 0.536078 0.732174 0.519585 0.060036 0.049389 0.060641 0.678849

**coverage**

**smape**

**mdape**

**mape**

**mae**

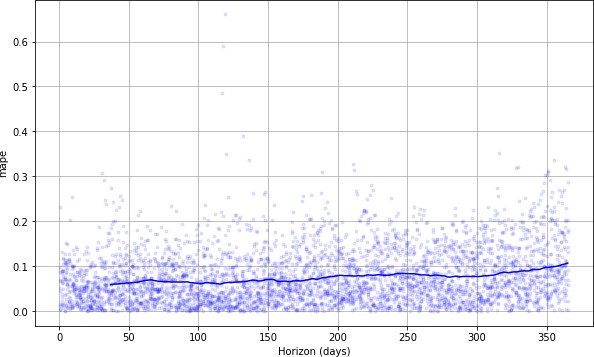
**rmse**

**mse**

**horizon**

Cross validation performance metrics can be visualized with **plot\_cross\_validation\_metric**, here shown for MAPE. 交叉验证指标可以使用 **plot\_cross\_validation\_metric** 进行可视化，下边的例子可视化该项目的MAPE。Dots show the absolute percent error for each prediction in **df\_cv**. The blue line shows the MAPE, where the mean is taken over a rolling window of the dots. **圆点表示df\_cv中每个预测的绝对误差百分比。蓝线显示的是MAPE，在点的滚动窗口上取的。**We see for this forecast that errors around 5% are typical for predictions one month into the future, and that errors increase up to around 11% for predictions that are a year out. 我们看到，在这个预测中，对未来一个月的预测，误差通常在5%左右，而对未来一年的预测，误差增加到11%左右。

1. # Python
2. from prophet.plot import plot\_cross\_validation\_metric
3. fig = plot\_cross\_validation\_metric(df\_cv, metric='mape')



The size of the rolling window in the figure can be changed with the optional argument rolling\_window, which specifies the proportion of forecasts to use in each rolling window. 图中滚动窗口的大小可以通过可选的参数**rolling\_window**来改变，它指定了每个滚动窗口中使用的预测比例。

The default is 0.1, corresponding to 10% of rows from **df\_cv** included in each window; increasing this will lead to a smoother average curve in the figure. 默认值是0.1，对应于每个窗口中包括10%的df\_cv行；增加这个值会使图中的平均曲线**更加平滑**。

The ***initial*** period should be long enough to capture all of the components of the model, in particular seasonalities and extra regressors: at least a year for yearly seasonality, at least a week for weekly seasonality, etc.*initial 期限*应该足够长，以捕获模型的所有组成部分，特别是季节性和额外的回归者：体现在序列中，对于年度季节性，此参数至少一年，对于周季节性，此参数至少一周，等等。

# Parallelizing cross validation

Cross-validation can also be run in parallel mode in Python, by setting specifying the parallel keyword. Four modes are supported

通过设置指定并行关键字，交叉验证也可以在Python中以并行模式运行。支持四种模式

* + parallel=None (Default, no parallelization)
  + parallel="processes"
  + parallel="threads"
  + parallel="dask"

For problems that aren’t too big, we recommend using **parallel="processes"**. It will achieve the highest performance when the parallel cross validation can be done on a single machine. For large problems, a [Dask](https://dask.org/) cluster can be used to do the cross validation on many machines. You will need to [install Dask](https://docs.dask.org/en/latest/install.html) separately, as it will not be installed with **prophet**.

1 from dask.distributed import Client 2

3

4

5 client = Client() # connect to the cluster 6

7 df\_cv = cross\_validation(m, initial='730 days', period='180 days', horizon='365 days', 8

9 parallel="dask")

10

11

# Hyperparameter tuning / 超参数的调整

Cross-validation can be used for tuning hyperparameters of the model, such as **changepoint\_prior\_scale** and **seasonality\_prior\_scale**. 交叉验证的目的就是调整模型的超参数，可以调整的参数有且不限于**changepoint\_prior\_scale**和**seasonality\_prior\_scal**e

A Python example is given below, with a 4x4 grid of those two parameters, with parallelization over cutoffs. Here parameters are evaluated on RMSE averaged over a 30-day horizon, but different performance metrics may be appropriate for different problems.

示例中网格搜索几个不同参数，在 cutoffs 上使用并行化

这里的参数是根据30天范围（horizon）内平均的 RMSE 来评估的，但是不同的性能评价指标指标可能适合不同的问题。

1. # Python
2. import itertools
3. import numpy as np
4. import pandas as pd 5

6 param\_grid = {

7 'changepoint\_prior\_scale': [0.001, 0.01, 0.1, 0.5],

8 'seasonality\_prior\_scale': [0.01, 0.1, 1.0, 10.0],

9 }

10

1. # Generate all combinations of parameters
2. all\_params = [dict(zip(param\_grid.keys(), v)) for v in itertools.product(\*param\_grid.values())] 13 rmses = [] # Store the RMSEs for each params here

14

15 # Use cross validation to evaluate all parameters 16 for params in all\_params:

1. m = Prophet(\*\*params).fit(df) # Fit model with given params
2. df\_cv = cross\_validation(m, cutoffs=cutoffs, horizon='30 days', parallel="processes")
3. df\_p = performance\_metrics(df\_cv, rolling\_window=1)
4. rmses.append(df\_p['rmse'].values[0]) 21
5. # Find the best parameters
6. tuning\_results = pd.DataFrame(all\_params) 24 tuning\_results['rmse'] = rmses

25 print(tuning\_results)

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | changepoint\_prior\_scale | seasonality\_prior\_scale | rmse |
| 2 0 | 0.001 | 0.01 | 0.757694 |
| 3 1 | 0.001 | 0.10 | 0.743399 |
| 4 2 | 0.001 | 1.00 | 0.753387 |
| 5 3 | 0.001 | 10.00 | 0.762890 |
| 6 4 | 0.010 | 0.01 | 0.542315 |
| 7 5 | 0.010 | 0.10 | 0.535546 |
| 8 6 | 0.010 | 1.00 | 0.527008 |
| 9 7 | 0.010 | 10.00 | 0.541544 |
| 10 8 | 0.100 | 0.01 | 0.524835 |
| 11 9 | 0.100 | 0.10 | 0.516061 |
| 12 10 | 0.100 | 1.00 | 0.521406 |
| 13 11 | 0.100 | 10.00 | 0.518580 |
| 14 12 | 0.500 | 0.01 | 0.532140 |
| 15 13 | 0.500 | 0.10 | 0.524668 |
| 16 14 | 0.500 | 1.00 | 0.521130 |
| 17 15 | 0.500 | 10.00 | 0.522980 |

1. # Python
2. best\_params = all\_params[np.argmin(rmses)] 3 print(best\_params)

1 {'changepoint\_prior\_scale': 0.1, 'seasonality\_prior\_scale': 0.1}

Alternatively, parallelization could be done across parameter combinations by parallelizing the loop above.

The Prophet model has a number of input parameters that one might consider tuning. Here are some general recommendations for hyperparameter tuning that may be a good starting place. Prophet模型有许多输入参数，我们可以考虑对其进行调整。这里有一些关于超参数调整的一般建议，可能是一个好的起点。

## Parameters that can be tuned / 可以被调整的超参数

* **changepoint\_prior\_scale**: This is probably the most impactful parameter. It determines the flexibility of the trend, and in particular how much the trend changes at the trend changepoints. As described in this documentation, if it is too small, the trend will be underfit and variance that should have been modeled with trend changes will instead end up being handled with the noise term. If it is too large, the trend will overfit and in the most extreme case you can end up with the trend capturing yearly seasonality. The default of 0.05 works for many time series, but this could be tuned; a range of [0.001, 0.5] would likely be about right. Parameters like this (regularization penalties; this is effectively a lasso penalty) are often tuned on a log scale.
* **seasonality\_prior\_scale**: This parameter controls the flexibility of the seasonality. Similarly, a large value allows the seasonality to fit large fluctuations, a small value shrinks the magnitude of the seasonality. The default is 10., which applies basically no regularization. That is because we very rarely see overfitting here (there’s inherent regularization with the fact that it is being modeled with a truncated Fourier series, so it’s essentially low-pass filtered). A reasonable range for tuning it would probably be [0.01, 10]; when set to 0.01 you should find that the magnitude of seasonality is forced to be very small. This likely also makes sense on a log scale, since it is effectively an L2 penalty like in ridge regression.
* **holidays\_prior\_scale**: This controls flexibility to fit holiday effects. Similar to seasonality\_prior\_scale, it defaults to 10.0 which applies basically no regularization, since we usually

have multiple observations of holidays and can do a good job of estimating their effects. This could also be tuned on a range of [0.01, 10] as with seasonality\_prior\_scale.

* **seasonality\_mode**: Options are ['additive', 'multiplicative']. Default is 'additive', but many business time series will have multiplicative seasonality. This is best identified just from looking at the time series and seeing if the magnitude of seasonal fluctuations grows with the magnitude of the time series (see the documentation here on multiplicative seasonality), but when that isn’t possible, it could be tuned.

## Maybe tune?

* **changepoint\_range**: This is the proportion of the history in which the trend is allowed to change. This defaults to 0.8, 80% of the history, meaning the model will not fit any trend changes in the last 20% of the time series. This is fairly conservative, to avoid overfitting to trend changes at the very end of the time series where there isn’t enough runway left to fit it well. With a human in the loop, this is something that can be identified pretty easily visually: one can pretty clearly see if the forecast is doing a bad job in the last 20%. In a fully-automated setting, it may be beneficial to be less conservative. It likely will not be possible to tune this parameter effectively with cross validation over cutoffs as described above. The ability of the model to generalize from a trend change in the last 10% of the time series will be hard to learn from looking at earlier cutoffs that may not have trend changes in the last 10%. So, this parameter is probably better not tuned, except perhaps over a large number of time series. In that setting, [0.8, 0.95] may be a reasonable range.

## Parameters that would likely not be tuned

* **growth**: Options are ‘linear’ and ‘logistic’. This likely will not be tuned; if there is a known saturating point and growth towards that point it will be included and the logistic trend will be used, otherwise it will be linear.
* **changepoints**: This is for manually specifying the locations of changepoints. None by default, which automatically places them.
* **n\_changepoints**: This is the number of automatically placed changepoints. The default of 25 should be plenty to capture the trend changes in a typical time series (at least the type that Prophet would work well on anyway). Rather than increasing or decreasing the number of changepoints, it will likely be more effective to focus on increasing or decreasing the flexibility at those trend changes, which is done with **changepoint\_prior\_scale**.
* **yearly\_seasonality**: By default (‘auto’) this will turn yearly seasonality on if there is a year of data, and off otherwise. Options are [‘auto’, True, False]. If there is more than a year of data, rather than trying to turn this off during HPO, it will likely be more effective to leave it on and turn down seasonal effects by tuning **seasonality\_prior\_scale**.
* **weekly\_seasonality**: Same as for **yearly\_seasonality**.
* **daily\_seasonality**: Same as for **yearly\_seasonality**.
* **holidays**: This is to pass in a dataframe of specified holidays. The holiday effects would be tuned with

**holidays\_prior\_scale**.

* **mcmc\_samples**: Whether or not MCMC is used will likely be determined by factors like the length of the time series and the importance of parameter uncertainty (these considerations are described in the documentation).
* **interval\_width**: Prophet **predict** returns uncertainty intervals for each component, like yhat\_lower and **yhat\_upper** for the forecast **yhat**. These are computed as quantiles of the posterior predictive distribution, and **interval\_width** specifies which quantiles to use. The default of 0.8 provides an 80% prediction interval. You could change that to 0.95 if you wanted a 95% interval. This will affect only the uncertainty interval, and will not change the forecast yhat at all and so does not need to be tuned.
* **uncertainty\_samples**: The uncertainty intervals are computed as quantiles from the posterior predictive interval, and the posterior predictive interval is estimated with Monte Carlo sampling. This parameter is the number of samples to use (defaults to 1000). The running time for predict will be linear in this number. Making it smaller will increase the variance (Monte Carlo error) of the uncertainty interval, and making it larger will reduce that variance. So, if the uncertainty estimates seem jagged this could be increased to further smooth them out, but it likely will not need to be changed. As with **interval\_width**, this parameter only affects the uncertainty intervals and changing it will not affect in any way the forecast yhat; it does not need to be tuned.
* **stan\_backend**: If both pystan and cmdstanpy backends set up, the backend can be specified. The predictions will be the same, this will not be tuned.

[Edit on GitHub](https://github.com/facebook/prophet/blob/main/docs/_docs/diagnostics.md)

**Facebook Open Source**

[Open Source Projects](https://code.facebook.com/projects/) [GitHub](https://github.com/facebook/) [Twitter](https://twitter.com/fbOpenSource) [Privacy](https://opensource.facebook.com/legal/privacy/) [Terms](https://opensource.facebook.com/legal/terms/) [Contribute to this project on GitHub](https://github.com/facebook/prophet)