Additional Topics / 额外话题

# Saving models / 模型保存

It is possible to save fitted Prophet models so that they can be loaded and used later.

In Python, models should not be saved with pickle; the Stan backend attached to the model object will not pickle well, and will produce issues under certain versions of Python. Instead, you should use the built-in serialization functions to serialize the model to json: 在 Python 中，模型不应该用 pickle 保存；附在模型对象上的 Stan 后台不会很好地 pickle，并且在某些版本的 Python 中会产生问题。***相反，你应该使用内置的序列化函数来将模型序列化为json：***

1. # Python
2. from prophet.serialize import model\_to\_json, model\_from\_json 3
3. with open('serialized\_model.json', 'w') as fout:
4. fout.write(model\_to\_json(m)) # Save model 6
5. with open('serialized\_model.json', 'r') as fin:
6. m = model\_from\_json(fin.read()) # Load model

The json file will be portable across systems, and deserialization is backwards compatible with older versions of prophet.

# Flat trend and custom trends / 扁平化趋势和自定义趋势

For time series that exhibit strong seasonality patterns rather than trend changes, it may be useful to force the trend growth rate to be flat. This can be achieved simply by passing growth=flat when creating the model:对于有些序列，其季节性表现更加突出，突出到使得序列中的趋势性变得不那么明显，对于这种情况，在建立模型时将growth=flat实现趋势扁平化。这样的操作使得趋势变得扁平\平缓

1. # Python
2. m = Prophet(growth='flat')

Note that if this is used on a time series that doesn’t have a constant trend, any trend will be fit with the noise term and so there will be high predictive uncertainty in the forecast. 注意，扁平化趋势如果在一个没有恒定趋势的时间序列上使用，任何趋势都会与噪声项拟合，所以预测中会有很高的预测不确定性。

To use a trend besides these three built-in trend functions (piecewise linear, piecewise logistic growth, and flat), 为了使用这三个趋势项（growth=liner/logistic/flat）以外的参数，you can download the source code from github, modify the trend function as desired in a local branch, and then install that local version. 你可以从github下载源代码，在本地分支中根据需要修改趋势函数，然后安装该本地版本。[This PR](https://github.com/facebook/prophet/pull/1466/files) provides a good illustration of what must be done to implement a custom trend, as does [this one](https://github.com/facebook/prophet/pull/1794) that implements a step function trend. 这个PR很好地说明了实现自定义趋势必须做什么，这个实现阶梯函数趋势的PR也是如此。

# Updating fitted models / 更新训练过的模型

A common setting for forecasting is fitting models that need to be updated as additional data come in. 预测的一个常见设置是拟合模型，这些模型需要随着额外数据的出现而更新。 Prophet models can only be fit once, and a new model must be re-fit when new data become available. 预言家的模型只能拟合一次，当新的数据出现时必须重新拟合一个新的模型。In most settings, model fitting is fast enough that there isn’t any issue with re-fitting from scratch

. 在大多数情况下，模型拟合的速度足够快，以至于不存在从头再拟合的问题。However, it is possible to speed things up a little by warm-starting the fit from the model parameters of the earlier model. This code example shows how this can be done in Python: 然而，通过从先前的模型参数中热启动拟合，可以使事情加快一点。这个代码例子显示了如何在Python中完成这一工作：

1. # Python
2. def warm\_start\_params(m):

3 """

1. Retrieve parameters from a trained model in the format used to initialize a new Stan model.
2. 从一个训练有素的模型中检索参数，其格式用于初始化一个新的斯坦模型。
3. Note that the new Stan model must have these same settings:
4. 注意，新的斯坦模型必须有这些相同的设置：
5. n\_changepoints, seasonality features, mcmc sampling
6. for the retrieved parameters to be valid for the new model. 8 以使检索到的参数对新模型有效。

9 Parameters 参数

10 ----------

11 m: A trained model of the Prophet class. 12 一个经过训练的Prophet class 模型

13 Returns 14 返回值

15 A Dictionary containing retrieved parameters of m. 16 一个包含检索到的m的参数的字典

"""

1. res = {}
2. for pname in ['k', 'm', 'sigma\_obs']:
3. if m.mcmc\_samples == 0:
4. res[pname] = m.params[pname][0][0]
5. else:
6. res[pname] = np.mean(m.params[pname])
7. for pname in ['delta', 'beta']:
8. if m.mcmc\_samples == 0:
9. res[pname] = m.params[pname][0]
10. else:
11. res[pname] = np.mean(m.params[pname], axis=0)
12. return res 29

30 df = pd.read\_csv('https://raw.githubusercontent.com/facebook/prophet/main/examples/example\_wp\_log\_peyton\_manning.csv') 31 df1 = df.loc[df['ds'] < '2016-01-19', :] # All data except the last day

32 m1 = Prophet().fit(df1) # A model fit to all data except the last day 33

34

1. %timeit m2 = Prophet().fit(df) # Adding the last day, fitting from scratch
2. %timeit m2 = Prophet().fit(df, init=warm\_start\_params(m1)) # Adding the last day, warm-starting from m1
3. 1.33 s ± 55.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
4. 185 ms ± 4.46 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

As can be seen, the parameters from the previous model are passed in to the fitting for the next with the kwarg **init**. 可以看出，前一个模型的参数被传递到下一个模型的拟合中，并带有kwarg **init**。在In this case, model fitting was about 5x faster when using warm starting. The speedup will generally depend on how much the optimal model parameters have changed with the addition of the new data. 这种情况下，使用热启动时，模型拟合的速度大约是5倍。加速一般取决于新数据加入后，最佳模型参数的变化程度。

There are few caveats that should be kept in mind when considering warm-starting. 在考虑暖启动的时候，有一些注意事项应该牢记在心。First, warm-starting may work well for small updates to the data (like the addition of one day in the example above) but can be worse than fitting from scratch if there are large changes to the data (i.e., a lot of days have been added). 首先，暖启动对于数据的小规模更新（如上面例子中增加的一天）可能效果很好，但如果数据有大的变化（即增加了很多天），暖启动可能比从头开始拟合要差。This is because when a large amount of history is added, the location of the changepoints will be very different between the two models, and so the parameters from the previous model may actually produce a bad trend initialization. 这是因为当大量的历史被添加时，两个模型之间的变化点的位置会有很大的不同，所以之前模型的参数实际上可能会产生一个糟糕的趋势初始化。Second, as a detail, the number of changepoints need to be consistent from one model to the next or else an error will be raised because the changepoint prior parameter delta will be the wrong size. 其次，作为一个细节，变化点的数量需要从一个模型到下一个模型保持一致，否则将产生错误，因为变化点先验参数delta的大小是错误的。

# External references

These github repositories provide examples of building on top of Prophet in ways that may be of broad interest:

* + [forecastr](https://github.com/garethcull/forecastr): A web app that provides a UI for Prophet.
  + [NeuralProphet](https://github.com/ourownstory/neural_prophet): A Prophet-style model implemented in pytorch, to be more adaptable and extensible.

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

**Facebook Open Source**

[Open Source Project](https://code.facebook.com/projects/)s [GitHu](https://github.com/facebook/)b [Twitte](https://twitter.com/fbOpenSource)r [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)