Neural Forecast

neuralforecast - 🧠 NeuralForecast

NeuralForecast offers a large collection of neural forecasting models focused on their usability, and robustness. The models range from classic networks like MLP, RNNs to novel proven contributions li

nixtla.github.io

最简单的应用

1、演示数据：

	unique_id	ds	y
0	1.0	1949-01-31	112.0
1	1.0	1949-02-28	118.0
2	1.0	1949-03-31	132.0
3	1.0	1949-04-30	129.0
4	1.0	1949-05-31	121.0
...	...	...	...
139	1.0	1960-08-31	606.0
140	1.0	1960-09-30	508.0
141	1.0	1960-10-31	461.0
142	1.0	1960-11-30	390.0
143	1.0	1960-12-31	432.0

144 rows × 3 columns

unique_id 标识数据集中的单个时间序列，ds 是日期，y 是目标变量。

在本例中，数据集由一组单个序列组成，但您可以轻松地将模型拟合到长格式的大型数据集。

*数据集必须有 `['unique_id', 'ds', 'y']` 列. `y` 列不能有空值

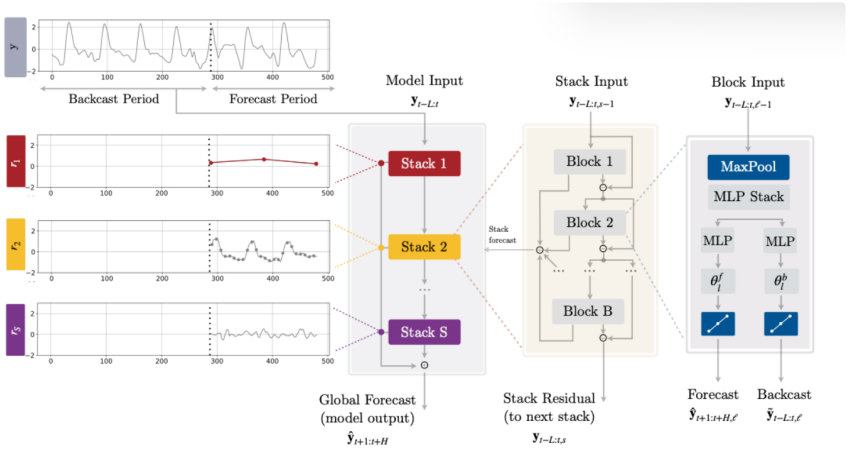
2、模型训练

使用 NeuralForecast.fit 可以训练一组模型到你的数据集。您可以定义预测范围【预测多少期】(在本例中为 12)，并修改模型的超参数。例如，对于 LSTM，我们更改了编码器和解码器的默认隐藏大小。

例子中用了 LSTM、NHITS 两个模型

NHITS 模型介绍

Neural Hierarchical Interpolation for Time Series Forecasting（神经分层插值）



Model 模块：有 S 个 Stack 模块组成，最后的输出通过累加得到最后的输出。和残差网络的构建是类似的，或者说和 boosting 的理念一致。

Stack 模块：是组成 Model 的子模块。可以看到从上到下，每个 block 的输出与原始输入做差值，再输入到下一个 block。也就是说下一个 block 学到的也是残差。最后输出到下一个 Stack 的也是整个 Stack 的残差。这两个模块结合在一起，可以统称为双残差 stack

block 模块：

下采样：下采样将时间序列采样为多种粒度的序列。采样用的池化层大小越大，则得到的序列更加低频/尺度较大；反之，则是更加高频/尺度较小。但是用了采样之后，得到的这些序列相较于原始序列，都变得更加低频/尺度较大了。利用不同 kernel_size 的池化层，就可以得到不同尺度的序列。下采样之后，序列长度变短了，所以复杂度变低了，效率变高了。

分层插值：与下采样正好对应，在预测结果上又做了个上采样。结合 N-HiTs 的模型架构图来理解，在第一个 stack，下采样的 kernel size 大，所以输入序列更短、尺度更大，预测出来的未来序列也更短，要想得到和期望 Horizon 一样的长度，就做一个上采样，也就是插值（比如线性插值，二次插值），需要插值很多个点。在最后一个 stack，下采样的 kernel size 小，所以序列更长、尺度更小，预测出来的未来序列也更长，就可以少插值一些。**所以每个 stack 实际上都是负责不同尺度的预测，最后把不同尺度的预测序列插值到相同粒度（也就是期望预测 Horizon 的粒度）然后相加即可。**可以结合模型图左侧来看，第一个 stack 因为序列短，尺度大，预测后插值结果就很平滑，更低频一些；而下面的 stack 则尺度越来越小，插值结果更高频一些。然后具体每个 stack 的 kernel size 怎么选取呢，一般用指数减小的方式即可

Python

```
horizon = 12

# Try different hyperparmeters to improve accuracy.
models = [LSTM(h=horizon,                   # 预测范围
               max_steps=500,               # 最大训练步数
               scaler_type='standard',      # 数据标准化方法
               encoder_hidden_size=64,      # LSTM隐藏层节点个数
               decoder_hidden_size=64,),    # MLP解码器的隐藏层大小
          NHITS(h=horizon,                  # 预测范围
                input_size=2 * horizon,     # 输入大小, y=[1,2,3,4] input_size=2 -> y_[t-2:t]=[1,2].
                max_steps=100,              # 最大训练步数
                n_freq_downsample=[2, 1, 1]) # 每个stack输出的下采样因子
          ]
nf = NeuralForecast(models=models, freq='M')
nf.fit(df=Y_df)
```

*这些模型的 Auto 版本 AutoLSTM 和 AutoNHITS 已经自动执行超参数选择。

*会对每个 unique_id、ds 和模型的预测

3、模型预测

Python

```
Y_hat_df = nf.predict()
Y_hat_df = Y_hat_df.reset_index()
Y_hat_df.head()
```

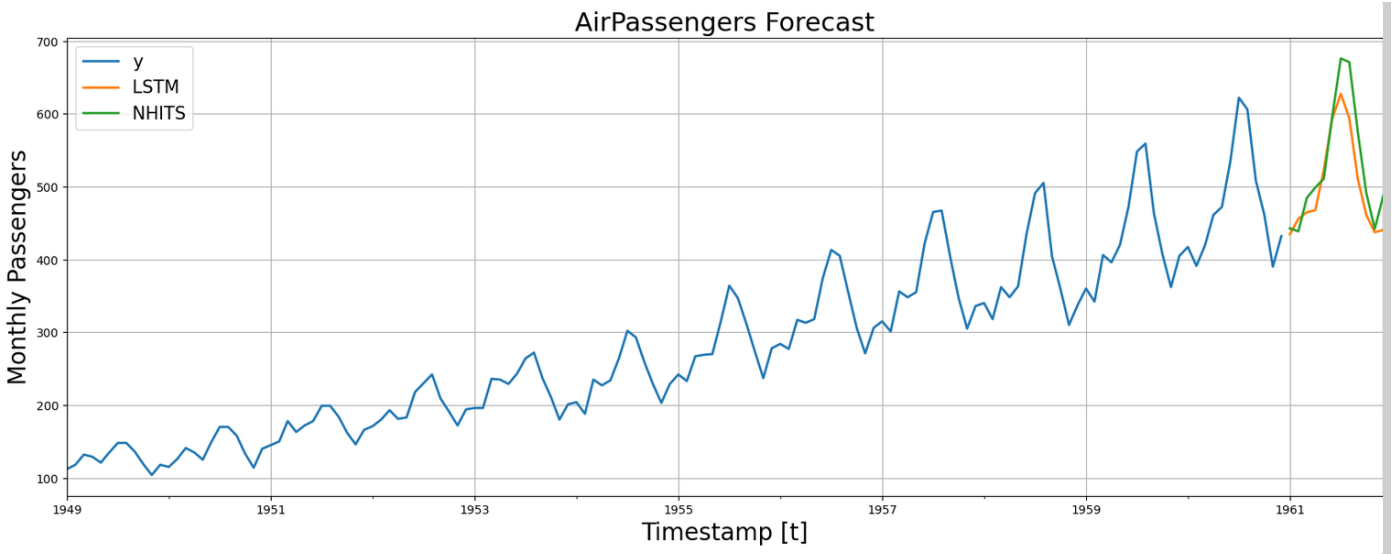
	unique_id	ds	LSTM	NHITS
0	1.0	1961-01-31	434.348785	442.606140
1	1.0	1961-02-28	455.762726	438.516235
2	1.0	1961-03-31	464.603516	484.361298
3	1.0	1961-04-30	467.556458	498.673004
4	1.0	1961-05-31	522.915771	510.665070

4、画图

Python

```
fig, ax = plt.subplots(1, 1, figsize = (20, 7))
plot_df = pd.concat([Y_df, Y_hat_df]).set_index('ds') # Concatenate the train and forecast dataframes
plot_df[['y', 'LSTM', 'NHITS']].plot(ax=ax, linewidth=2)

ax.set_title('AirPassengers Forecast', fontsize=22)
ax.set_ylabel('Monthly Passengers', fontsize=20)
ax.set_xlabel('Timestamp [t]', fontsize=20)
ax.legend(prop={'size': 15})
ax.grid()
```



完整流程

1、演示数据

Python

```
import pandas as pd
Y_df = pd.read_parquet('https://datasets-nixtla.s3.amazonaws.com/m4-hourly.parquet')
Y_df
```

	unique_id	ds	y
0	H1	1	605.0
1	H1	2	586.0
2	H1	3	586.0
3	H1	4	559.0
4	H1	5	511.0
...	...	...	...
373367	H99	744	24039.0
373368	H99	745	22946.0
373369	H99	746	22217.0
373370	H99	747	21416.0
373371	H99	748	19531.0

373372 rows × 3 columns

该数据集包含 414 个独特的序列，平均 900 个观测值。我们将只选择 10 个唯一 id，并只保留最近一周的 id。根据您的加工基础设施随意选择多或少的系列。

Python

```
#uids = Y_df['unique_id'].unique()[:10] # Select 10 ids to make the example faster

#Y_df = Y_df.query('unique_id in @uids')
max_len = 10 * 24
Y_df = Y_df.groupby('unique_id').tail(10 * 24) #选每组最后240个数据，Select last 10 days of data to make example faster
# 把每组的ds变成1-240
Y_df['ds'] = Y_df.groupby('unique_id')['ds'].transform(lambda x: range(len(x) - max_len + 1, max_len + 1))
Y_df
```

	unique_id	ds	y
508	H1	1	613.0
509	H1	2	550.0
510	H1	3	509.0
511	H1	4	481.0
512	H1	5	460.0
...	...	...	...
373367	H99	236	24039.0
373368	H99	237	22946.0
373369	H99	238	22217.0
373370	H99	239	21416.0
373371	H99	240	19531.0

99360 rows × 3 columns

```
Y_df[Y_df['unique_id']=='H1']
```

✓ 0.1s

	unique_id	ds	y
508	H1	1	613.0
509	H1	2	550.0
510	H1	3	509.0
511	H1	4	481.0
512	H1	5	460.0
...	...	...	...
743	H1	236	785.0
744	H1	237	756.0
745	H1	238	719.0
746	H1	239	703.0
747	H1	240	659.0

240 rows × 3 columns

2、绘制时间序列图

Python

```
from statsforecast import StatsForecast

StatsForecast.plot(Y_df, engine='matplotlib')
```

3、训练模型

每个 Auto 模型都包含一个默认的搜索空间，该搜索空间在多个大规模数据集上进行了广泛的测试。此外，用户可以为特定的数据集和任务定义特定的搜索空间。
首先，我们为 autohits 和 AutoTFT 模型创建一个自定义搜索空间。搜索空间是用字典指定的，对应于模型的超参数，其中的值是一个 Tune 函数，用于指定如何对超参数进行采样。例如，使用 randint 对整数进行统一采样，使用 choice 对列表中的值进行采样。

Python

```
config_nhits = {
    "input_size": tune.choice([48, 48*2, 48*3, 48*5]),              # Length of input window
    "n_blocks": 5*[1],                                              # Length of input window
    "mlp_units": 5 * [[64, 64]],                                   # Length of input window
    "n_pool_kernel_size": tune.choice([5*[1], 5*[2], 5*[4],
                                       [8, 4, 2, 1, 1]]),           # MaxPooling Kernel size
    "n_freq_downsample": tune.choice([[8, 4, 2, 1, 1],
                                      [1, 1, 1, 1, 1]]),            # Interpolation expressivity ratios
    "learning_rate": tune.loguniform(1e-4, 1e-2),                   # Initial Learning rate
    "scaler_type": tune.choice([None]),                             # Scaler type
    "max_steps": tune.choice([1000]),                               # Max number of training iterations
    "batch_size": tune.choice([32, 64, 128, 256]),                  # Number of series in batch
    "windows_batch_size": tune.choice([128, 256, 512, 1024]),       # Number of windows in batch
    "random_seed": tune.randint(1, 20),                             # Random seed
}
```

`Auto` 模型需要定义的参数

`h`：预测范围

`loss`：损失函数

`config`：超参数搜索范围. 如果是 `None`，`Auto` 会从预先定义的搜索范围搜索

`search_alg`：搜索算法，默认随机搜索。Refer to https://docs.ray.io/en/latest/tune/api_docs/suggestion.html for more information on the different search algorithm options.

`num_samples`：采样数量

*样本数量 num_samples 是一个关键参数!当我们在搜索空间中探索更多配置时，较大的值通常会产生更好的结果，但它会增加训练时间。更大的搜索空间通常需要更多的样本。作为一般规则，我们建议将 num_samples 设置为大于 20 的值。

Python

```
nf = NeuralForecast(
    models=[
        AutoNHITS(h=48, config=config_nhits, loss=MQLoss(), num_samples=5),
        AutoLSTM(h=48, loss=MQLoss(), num_samples=2),
    ],
```

```
    freq='H'
)
```

接下来，我们使用 Neuralforecast 类来训练 Auto 模型。在此步骤中，Auto 模型将自动执行超参数调优，训练具有不同超参数的多个模型，在验证集上产生预测，并对其进行评估。根据验证集上的 loss 选择最佳配置。在过程中，只存储和使用最佳模型。

4、画预测图

Python

```
nf.fit(df=Y_df)
fcst_df = nf.predict()
fcst_df.columns = fcst_df.columns.str.replace('-median', '')
fcst_df.head()
```

Python

```
#画图
StatsForecast.plot(Y_df, fcst_df, engine='matplotlib', max_insample_length=48 * 3, level=[80, 90])
#指定参数可以选择id和模型
StatsForecast.plot(Y_df, fcst_df, models=["AutoLSTM"], unique_ids=["H185", "H221"], level=[80, 90], engine='matplotlib')
```

5、模型评估

在前面的步骤中，我们用历史数据来预测未来。然而，为了评估其准确性，我们还想知道该模型在过去的表现。为了评估模型对数据的准确性和稳健性，请执行交叉验证。

对于时间序列数据，交叉验证是通过在历史数据上定义一个滑动窗口并预测其后的时间段来完成的。这种形式的交叉验证使我们能够在更大范围的时间实例中更好地估计模型的预测能力，同时保持训练集中的数据连续，这是我们模型所要求的。

下图描述了这样一个交叉验证策略:

设置 n_windows=1 反映了传统的训练-测试分割，我们的历史数据作为训练集，过去 48 小时作为测试集。

交叉验证函数 `cross_validation` 参数:

- `df`: 训练数据
- `step_size` (int): 每个窗口之间的步长。换句话说:希望多久运行一次预测过程。
- `n_windows` (int): 用于交叉验证的窗口数。换句话说:你想评估过去有多少个预测过程。

建立模型:

Python

```
from neuralforecast.auto import AutoNHITS, AutoLSTM
config_nhits = {
    "input_size": tune.choice([48, 48*2, 48*3, 48*5]),              # Length of input window
    "n_blocks": 5*[1],                                              # Length of input window
    "mlp_units": 5 * [[64, 64]],                                   # Length of input window
    "n_pool_kernel_size": tune.choice([5*[1], 5*[2], 5*[4],
                                       [8, 4, 2, 1, 1]]),           # MaxPooling Kernel size
    "n_freq_downsample": tune.choice([[8, 4, 2, 1, 1],
                                      [1, 1, 1, 1, 1]]),            # Interpolation expressivity ratios
    "learning_rate": tune.loguniform(1e-4, 1e-2),                   # Initial Learning rate
    "scaler_type": tune.choice([None]),                             # Scaler type
    "max_steps": tune.choice([1000]),                               # Max number of training iterations
    "batch_size": tune.choice([32, 64, 128, 256]),                  # Number of series in batch
    "windows_batch_size": tune.choice([128, 256, 512, 1024]),       # Number of windows in batch
    "random_seed": tune.randint(1, 20),                             # Random seed
}
nf = NeuralForecast(
    models=[
        AutoNHITS(h=48, config=config_nhits, loss=MQLoss(), num_samples=5),
        AutoLSTM(h=48, loss=MQLoss(), num_samples=2),
    ],
    freq='H'
)
```

交叉验证结果:

Python

```
cv_df = nf.cross_validation(Y_df, n_windows=2)
```

`cv_df` 包含以下列：

`unique_id`：识别时间序列

`ds`：时间戳或者时间索引

`cutoff`：n_windows 的最后一个日期戳或时间索引。如果 n_windows=1，那么一个唯一的截止值，如果 n_windows=2，那么两个唯一的截止值。

`y`：true value

`"model"`：columns with the model’s name and fitted value.

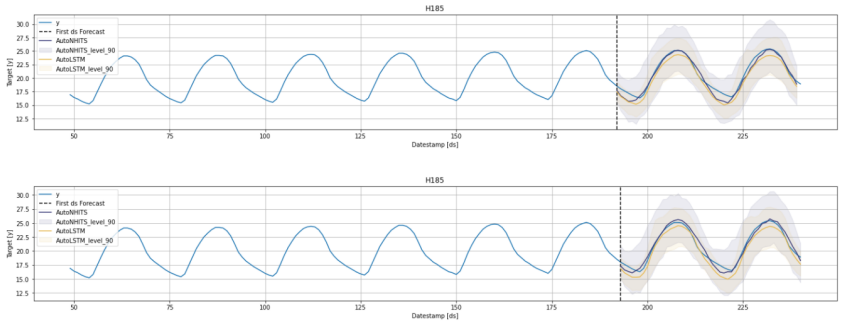
Python

```
cv_df.columns = cv_df.columns.str.replace('-median', '')
cv_df.head()
```

	unique_id	ds	cutoff	AutoNHITS	AutoNHITS-lo-90	AutoNHITS-lo-80	AutoNHITS-hi-80	AutoNHITS-hi-90	AutoLSTM	AutoL lo-90
0	H1	192	191	681.490662	646.704773	666.150208	713.381226	726.007996	662.800598	561.54
1	H1	193	191	625.682495	576.598633	596.370361	649.335388	690.485413	605.095581	509.80
2	H1	194	191	573.010193	537.415771	535.986572	598.875427	619.973267	557.995544	471.06
3	H1	195	191	521.889893	475.909180	492.678528	571.340271	577.343994	516.248901	420.72
4	H1	196	191	487.002380	427.999207	441.623596	521.709717	538.648560	492.773315	396.84

Python

```
for cutoff in cv_df['cutoff'].unique():
    StatsForecast.plot(
        Y_df,
        cv_df.query('cutoff == @cutoff').drop(columns=['y', 'cutoff']),
        max_insample_length=48 * 4,
        unique_ids=['H185'],
        engine='matplotlib'
    )
```



交叉验证评估结果：

Python

```
from datasetsforecast.losses import mse, mae, rmse
from datasetsforecast.evaluation import accuracy
evaluation_df = accuracy(cv_df, [mse, mae, rmse], agg_by=['unique_id'])
evaluation_df['best_model'] = evaluation_df.drop(columns=['metric', 'unique_id']).idxmin(axis=1)
#best_model列返回最小metric值的列名
evaluation_df.head()
```

*也可以使用平均百分比误差(MAPE)，但是对于精准预测，MAPE 值很难判断，对于评估预测质量没有用处。

	metric	unique_id	AutoNHITS	AutoLSTM	best_model
0	mae	H1	26.642945	25.342275	AutoLSTM
1	mae	H10	13.429816	19.879758	AutoNHITS
2	mae	H100	172.525824	171.068505	AutoLSTM
3	mae	H101	301.557668	104.159498	AutoLSTM
4	mae	H102	131.746874	505.692479	AutoNHITS

汇总表

Python

```
summary_df = evaluation_df.groupby(['metric', 'best_model']).size().sort_values().to_frame()

summary_df = summary_df.reset_index()
summary_df.columns = ['metric', 'model', 'nr. of unique_ids']
summary_df
```

	metric	model	nr. of unique_ids
0	mse	AutoNHITS	186
1	rmse	AutoNHITS	187
2	mae	AutoNHITS	188
3	mae	AutoLSTM	226
4	rmse	AutoLSTM	227
5	mse	AutoLSTM	228

Python

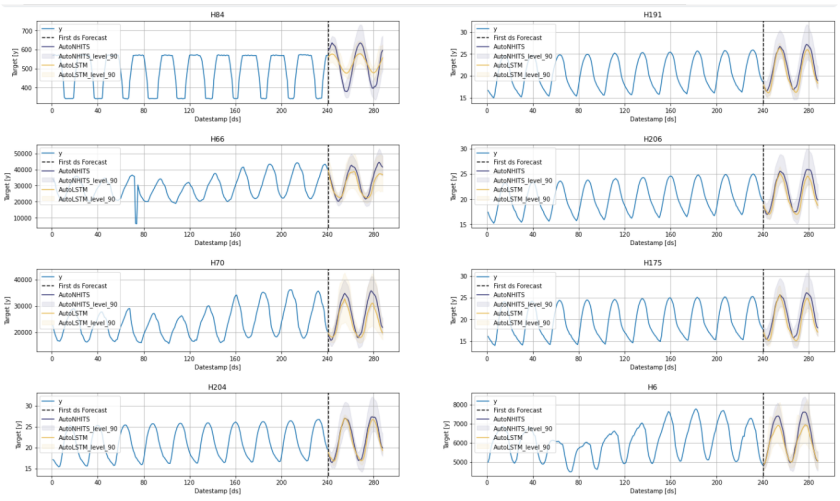
```
summary_df.query('metric == "mse"')
#筛选mse的模型
```

绘制特定 id 序列

Python

```
nhits_ids = evaluation_df.query('best_model == "AutoNHITS" and metric == "mse"')['unique_id'].unique()

StatsForecast.plot(Y_df, fcst_df, unique_ids=nhits_ids, engine='matplotlib')
```



6、给每个序列筛选最佳模型

定义一个实用函数，该函数使用预测值和测试出局获取预测数据，并为每个 unique_id 返回最佳预测

Python

```
def get_best_model_forecast(forecasts_df, evaluation_df, metric):
    df = forecasts_df.set_index('ds', append=True).stack().to_frame().reset_index(level=2) # Wide to long
    df.columns = ['model', 'best_model_forecast']
    df = df.join(evaluation_df.query('metric == @metric').set_index('unique_id')[['best_model']])
    df = df.query('model.str.replace("-lo-90|-hi-90", "", regex=True) == best_model').copy()
    df.loc[:, 'model'] = [model.replace(bm, 'best_model') for model, bm in zip(df['model'], df['best_model'])]
    df = df.drop(columns='best_model').set_index('model', append=True).unstack()
```

```
    df.columns = df.columns.droplevel()
    df = df.reset_index(level=1)
    return df
```

Python

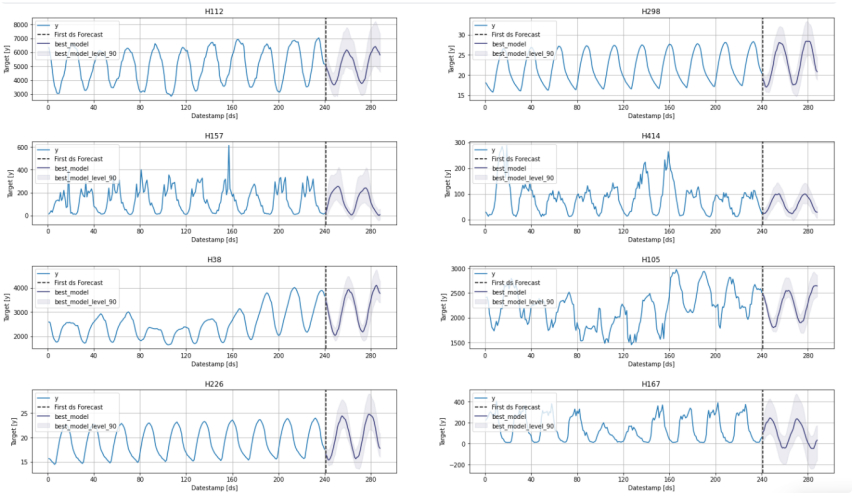
```
prod_forecasts_df = get_best_model_forecast(fcst_df, evaluation_df, metric='mse')

prod_forecasts_df.head()
```

model	ds	best_model	best_model-hi-90	best_model-lo-90
unique_id				
H1	241	612.727234	653.332764	575.606018
H1	242	561.754028	611.646851	510.381958
H1	243	534.809692	581.725769	486.517761
H1	244	509.679291	550.369507	466.653137
H1	245	485.630920	529.474731	446.374634

Python

```
StatsForecast.plot(Y_df, prod_forecasts_df, level=[90], engine='matplotlib')
```



数据格式及变量

数据格式：多变量数据

必须 concat 成一个序列，以 unique_id 区分

变量

df 数据框包含目标变量和外生变量过去的信息来训练模型。unique_id 列标识市场，ds 包含日期戳，y 包含电价。

	unique_id	ds	y	gen_forecast	system_load	week_day
0	FR	2015-01-01 00:00:00	53.48	76905.0	74812.0	3
1	FR	2015-01-01 01:00:00	51.93	75492.0	71469.0	3
2	FR	2015-01-01 02:00:00	48.76	74394.0	69642.0	3
3	FR	2015-01-01 03:00:00	42.27	72639.0	66704.0	3
4	FR	2015-01-01 04:00:00	38.41	69347.0	65051.0	3

包括历史和未来时间变量作为列。在本例中，我们将系统负载(system_load)添加为历史数据。对于未来的变量，我们包括对将产生多少电力的预测(gen_forecast)和星期几(week_day)。正如我们在 Olivares 等人(2022)中所证明的那样，电力系统需求和报价对价格的影响都很大，将这些变量纳入模型大大提高了性能。

历史变量和未来变量之间的区别将在后面作为模型的参数加以区分。

在单独的 static_df 数据框架中添加静态变量。在这个例子中，我们使用电力市场的单热编码。**对于 df 数据框的每个 unique_id, static_df 必须包含一个观测值(行)，不同的静态变量作为列。**

```
static_df = pd.read_csv('https://datasets-nixtla.s3.amazonaws.com/EPF_FR_BE_static.cs
static_df.head()
```

	unique_id	market_0	market_1
0	FR	1	0
1	BR	0	1

变量区分：

我们通过外生变量是否反映模型数据的静态方面或时间依赖方面来区分它们。
static exogenous variable 静态外生变量：静态外生变量携带每个时间序列的时不变信息。当使用全局参数构建模型以预测多个时间序列时，这些变量允许在具有相似静态变量水平的时间序列组内共享信息。静态变量的例子包括区域标识符、产品组标识符等。eg.预测产品销量时 ep_id

Historic exogenous variables 历史外生变量:这种时间相关的外生变量仅限于过去的观测值。它的预测能力取决于格兰杰因果关系，因为它的过去值可以提供关于目标变量未来值的重要信息。

> 格兰杰因果检验：已知上一时刻所有信息，这一时刻 X 的概率分布情况”和“已知上一时刻除 Y 以外的所有信息，这一时刻 X 的概率分布情况”，来判断 Y 对 X 是否存在因果关系；包含了变量 X、Y 的过去信息的条件下，对变量 Y 的预测效果要优于只单独由 Y 的过去信息对 Y 进行的预测效果，即**变量 X 有助于解释变量 Y 的将来变化**，则认为变量 X 是引致变量 Y 的格兰杰原因。再通俗一点，就是说，目标是预测 Y 的变化，加上 X 的预测结果要比只有 Y 的预测结果好

Future exogenous variables 未来外生变量:与历史外生变量相比，预测时的未来值是可用的。示例包括节假日、天气预报和可能导致大幅高峰和低谷的已知事件，例如计划的促销活动。

要向模型添加外生变量，首先在初始化期间将前面数据框中的每个变量的名称指定为相应的模型超参数:futr_exog_list、hist_exog_list 和 stat_exog_list。我们还将预测范围设置为 24 以生成第二天的小时预测，并设置 input_size 以使用最近 5 天的数据作为输入。

Python

```
horizon = 24 # day-ahead daily forecast
models = [NHITS(h = horizon,
                input_size = 5*horizon,
                futr_exog_list = ['gen_forecast', 'week_day'], # <- Future exogenous variables
                hist_exog_list = ['system_load'], # <- Historical exogenous variables
                stat_exog_list = ['market_0', 'market_1'], # <- Static exogenous variables
                scaler_type = 'robust')]#scaler_type对数据标准化，对所有变量都进行标准化
```

总之，要将外生变量添加到模型中，请确保遵循以下步骤:

1、将时间外生变量作为列添加到主数据框架(df)。
2、使用 static_df 数据框架添加静态外生变量。
3、在相应的模型超参数中指定每个变量的名称。
4、如果模型使用未来的外生变量，将未来的数据框(futr_df)传递给 predict 方法。

自动调参

深度学习模型是最先进的时间序列预测。它们的表现优于统计和基于树的方法，然而，它们的性能受到超参数选择的极大影响。选择最优配置(称为超参数调优的过程)对于实现最佳性能至关重要。

超参数调优的主要步骤是:

定义训练集和验证集。

定义搜索空间。

使用搜索算法采样配置，训练模型，并在验证集上评估它们。

选择并存储最佳模型。

使用 Neuralforecast，我们自动化和简化了使用 Auto 模型的超参数调整过程。库中的每个模型都有一个 Auto 版本(例如，autohits, AutoTFT)，它可以在默认或用户定义的搜索空间上执行自动超参数选择。Auto 模型将 Ray 的 Tune 库包装为一个用户友好的简化 API，并具有其大部分功能。

实例：

1、导入数据

```python
from neuralforecast.utils import AirPassengersDF

Y_df = AirPassengersDF
Y_df.head()
```

	unique_id	ds	y
0	1.0	1949-01-31	112.0
1	1.0	1949-02-28	118.0
2	1.0	1949-03-31	132.0
3	1.0	1949-04-30	129.0
4	1.0	1949-05-31	121.0

2、定义搜索空间

每个 Auto 模型都包含一个默认的搜索空间，该搜索空间在多个大规模数据集上进行了广泛的测试。此外，用户可以为特定的数据集和任务定义特定的搜索空间。

首先，我们为 autohits 模型创建一个自定义搜索空间。搜索空间是用字典指定的，其中键对应于模型的超参数，值是一个 Tune 函数，用于指定如何对超参数进行采样。例如，使用 randint 对整数进行统一采样，使用 choice 对列表中的值进行采样。loguniform 区间

在下面的例子中，我们正在优化 learning_rate 和两个 NHITS 特定的超参数:n_pool_kernel_size 和 n_freq_downsample。此外，我们使用搜索空间修改默认的超参数，如 max_steps 和 val_check_steps。

```python
from ray import tune
nhits_config = {
       "max_steps": 100,                                                     # Number of SGD steps
       "input_size": 24,                                                     # Size of input window
       "learning_rate": tune.loguniform(1e-5, 1e-1),                         # Initial Learning rate
       "n_pool_kernel_size": tune.choice([[2, 2, 2], [16, 8, 1]]),           # MaxPool's Kernelsize
       "n_freq_downsample": tune.choice([[168, 24, 1], [24, 12, 1], [1, 1, 1]]), # Interpolation expressivity ratios
       "val_check_steps": 50,                                                # Compute validation every 50 steps
       "random_seed": tune.randint(1, 10),                                   # Random seed
    }
```

*配置字典在模型之间是不可互换的，因为它们具有不同的超参数。https://nixtla.github.io/neuralforecast/models.html 获取每个模型超参数的完整列表。

3、建立自动模型

具体参数参考完整流程- 3、训练模型

接下来，我们使用 Neuralforecast 类来训练 Auto 模型。在此步骤中，Auto 模型将自动执行超参数调优，训练具有不同超参数的多个模型，在验证集上产生预测，并对其进行评估。根据验证集上的错误选择最佳配置。在推理过程中，只存储和使用最佳模型。

使用 fit 方法的 val_size 参数来控制验证集的长度。在这种情况下，我们将验证集设置为预测范围的两倍。

超参数调优的结果在 Auto 模型的 results 属性中可用。使用 get_dataframe 方法在 pandas 数据框中获取结果。

```python
from neuralforecast import NeuralForecast
nf = NeuralForecast(models=[model], freq='M')
nf.fit(df=Y_df, val_size=24)

results = model.results.get_dataframe()
results.head()
#输出参数调优结果
```

	loss	time_this_iter_s	done	timesteps_total	episodes_total	training_iteration	trial_id	expe
0	34791.984375	11.577301	False	NaN	NaN	2	12439254	80bb
1	42.189243	10.443844	False	NaN	NaN	2	125ce09c	80bb
2	19.739182	10.449568	False	NaN	NaN	2	22d60660	80bb
3	15.072639	9.953516	False	NaN	NaN	2	2fd95ef2	80bb
4	58.782948	11.148285	False	NaN	NaN	2	3c93c36c	80bb

我们使用预测方法使用最优超参数对未来 12 个月进行预测。

Python

```
Y_hat_df = nf.predict()
Y_hat_df = Y_hat_df.reset_index()
Y_hat_df.head()
```

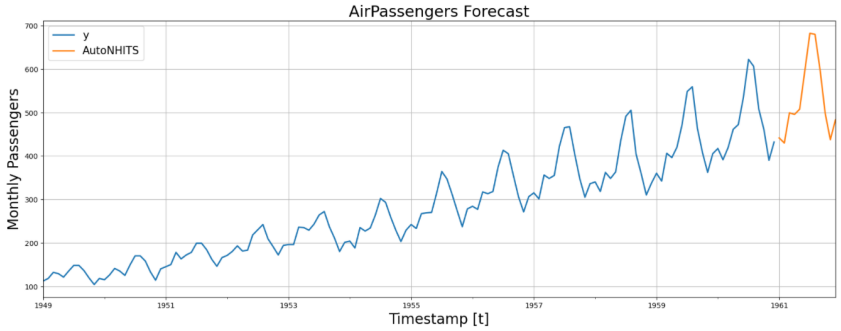
	unique_id	ds	AutoNHITS
0	1.0	1961-01-31	441.228943
1	1.0	1961-02-28	429.527832
2	1.0	1961-03-31	498.914246
3	1.0	1961-04-30	495.418640
4	1.0	1961-05-31	507.392731

绘制原始时间序列和预测结果。

Python

```
import pandas as pd
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1, figsize = (20, 7))
plot_df = pd.concat([Y_df, Y_hat_df]).set_index('ds') # Concatenate the train and forecast dataframes
plot_df[['y', 'AutoNHITS']].plot(ax=ax, linewidth=2)

ax.set_title('AirPassengers Forecast', fontsize=22)
ax.set_ylabel('Monthly Passengers', fontsize=20)
ax.set_xlabel('Timestamp [t]', fontsize=20)
ax.legend(prop={'size': 15})
ax.grid()
```



保存模型

保存

The two methods to consider are:

1. `NeuralForecast.save`: 将模型保存到磁盘，允许保存数据集和配置。
2. `NeuralForecast.load`: 从给定路径加载模型

`save` 函数参数:

- `path`: 保存模型的目录。
- `model_index`: 可选列表，用于指定要保存的模型。例如，要只保存 `NHITS` 就设置为 `model_index=[2]`.
- `overwrite`: 用于覆盖 `path` 中的现有文件。当为 True 时，该方法将只覆盖具有冲突名称的模型。
- `save_dataset`: 保存 `Dataset` 对象和数据集

```python
nf.save(path='./checkpoints/test_run/',
        model_index=None,
        overwrite=True,
        save_dataset=True)
```

对于每个模型，创建并存储两个文件:

- `[model_name]_[suffix].ckpt`: Pytorch Lightning 检查点文件与模型参数和超参数。
- `[model_name]_[suffix].pkl`: 具有配置属性的字典。

`model_name` 以小写形式对应于模型的名称(例如: `nhits`). 我们使用一个数字后缀来区分每个类别的多个模型。在本例中，名称将是 `automlp_0`, `nbeats_0`, and `nhits_0`.

Auto 模型将被存储为它们的基本模型。例如，上面训练的 AutoMLP 存储为 MLP 模型，并在调优期间找到最佳超参数。

*Auto 模型将被存储为它们的基本模型。例如，上面训练的 AutoMLP 存储为 MLP 模型，并在调优期间找到最佳超参数。

加载模型:

使用 Load 方法加载保存的模型，指定路径，并使用新的 nf2 对象生成预测。

```python
nf2 = NeuralForecast.load(path='./checkpoints/test_run/')
Y_hat_df = nf2.predict().reset_index()
Y_hat_df.head()
```

最后，绘制预测图以确认它们与原始预测相同。

```python
plot_df = pd.concat([Y_df, Y_hat_df]).set_index('ds') # Concatenate the train and forecast dataframes

plt.figure(figsize = (12, 3))
plot_df[['y', 'NBEATS', 'NHITS', 'MLP']].plot(linewidth=2)

plt.title('AirPassengers Forecast', fontsize=10)
plt.ylabel('Monthly Passengers', fontsize=10)
plt.xlabel('Timestamp [t]', fontsize=10)
plt.axvline(x=plot_df.index[-horizon], color='k', linestyle='--', linewidth=2)
plt.legend(prop={'size': 10})
plt.show()
```

问题：

horizon 是训练模型时用的？在预测时就不再使用了

静态变量必须是与 unique id 对应的？

Tutorial

德克萨斯州电力市场负荷数据(ERCOT)上训练 TFT 模型

数据重命名

首先，阅读 2022 年 ERCOT 市场的历史性总需求。我们通过添加由于夏令时而丢失的小时、解析 date 到 datetime 格式以及过滤感兴趣的列来处理原始数据。

增加 num_samples 参数可以为所选模型探索更广泛的配置集。根据经验，选择大于 15 的值。
num_samples=3 这个示例应该在大约 20 分钟内运行。

我们所有的模型都可以用于点预测和概率预测。要生成概率输出，只需将损失修改为我们的 `distributionloss` 之一。

*TFT 是一个非常大的模型，可能需要大量的内存!如果你用完 GPU 内存，尝试配置你的配置搜索空间并减少 hidden_size,n_heads 和 windows_batch_size 参数。