tf.contrib.timeseries.StructuralEnsembleRegressor

Contents

Class `StructuralEnsembleRegressor`

Defined in `tensorflow/contrib/timeseries/python/timeseries/estimators.py`.

An Estimator for structural time series models.

"Structural" refers to the fact that this model explicitly accounts for structure in the data, such as periodicity and trends.

`StructuralEnsembleRegressor` is a state space model. It contains components for modeling level, local linear trends, periodicity, and mean-reverting transients via a moving average component. Multivariate series are fit with full covariance matrices for observation and latent state transition noise, each feature of the multivariate series having its own latent components.

Note that unlike `ARRegressor`, `StructuralEnsembleRegressor` is sequential, and so accepts variable window sizes with the same model.

For training, `RandomWindowInputFn` is recommended as an `input_fn`. Model state is managed through `ChainingStateManager`: since state space models are inherently sequential, we save state from previous iterations to get approximate/eventual consistency while achieving good performance through batched computation.

For evaluation, either pass a significant chunk of the series in a single window (e.g. set `window_size` to the whole series with `WholeDatasetInputFn`), or use enough random evaluation iterations to cover several passes through the whole dataset. Either method will ensure that stale saved state has been flushed.

Properties

`config`

`model_dir`

`model_fn`

Returns the model_fn which is bound to self.params.

Returns:

The model_fn with following signature: `def model_fn(features, labels, mode, config)`

params

Methods

__init__

```
__init__(
    periodicities,
    num_features,
    cycle_num_latent_values=11,
    moving_average_order=4,
    autoregressive_order=0,
    exogenous_feature_columns=None,
    exogenous_update_condition=None,
    dtype=tf.double,
    anomaly_prior_probability=None,
    optimizer=None,
    model_dir=None,
    config=None
)
```

Initialize the Estimator.

Args:

- `periodicities`: The expected periodicity of the data (for example 24 if feeding hourly data with a daily periodicity, or 60 * 24 if feeding minute-level data with daily periodicity). Either a scalar or a list. This parameter can be any real value, and does not control the size of the model. However, increasing this without increasing `num_values_per_cycle` will lead to smoother periodic behavior, as the same number of distinct values will be cycled through over a longer period of time.
- `num_features`: The dimensionality of the time series (one for univariate, more than one for multivariate).
- `cycle_num_latent_values`: Along with `moving_average_order` and `num_features`, controls the latent state size of the model. Square matrices of size `num_features * (moving_average_order + cycle_num_latent_values + 3)` are created and multiplied, so larger values may be slow. The trade-off is with resolution: cycling between a smaller number of latent values means that only smoother functions can be modeled.
- `moving_average_order`: Controls model size (along with `cycle_num_latent_values` and `autoregressive_order`) and the number of steps before transient deviations revert to the mean defined by the period and level/trend components.
- `autoregressive_order`: Each contribution from this component is a linear combination of this many previous contributions. Also helps to determine the model size. Learning autoregressive coefficients typically requires more steps and a smaller step size than other components.
- `exogenous_feature_columns`: A list of tf.contrib.layers.FeatureColumn objects (for example tf.contrib.layers.embedding_column) corresponding to exogenous features which provide extra information to the model but are not part of the series to be predicted. Passed to tf.contrib.layers.input_from_feature_columns.
- `exogenous_update_condition`: A function taking two Tensor arguments, `times` (shape [batch size]) and `features` (a dictionary mapping exogenous feature keys to Tensors with shapes [batch size, ...]), and returning a boolean Tensor with shape [batch size] indicating whether state should be updated using exogenous features for each part of the batch. Where it is False, no exogenous update is performed. If None (default), exogenous updates are always performed. Useful for avoiding "leaky" frequent exogenous updates when sparse updates are desired. Called only during graph construction. See the "known anomaly" example for example usage.
- `dtype`: The floating point data type to compute with. float32 may be faster, but can be problematic for larger models and longer time series.
- `anomaly_prior_probability`: If not None, the model attempts to automatically detect and ignore anomalies during

training. This parameter then controls the prior probability of an anomaly. Values closer to 0 mean that points will be discarded less frequently. The default value (None) means that anomalies are not discarded, which may be slightly faster.

- `optimizer`: The optimization algorithm to use when training, inheriting from tf.train.Optimizer. Defaults to Adam with step size 0.02.
- `model_dir`: See `Estimator`.
- `config`: See `Estimator`.

build_raw_serving_input_receiver_fn

```
build_raw_serving_input_receiver_fn(
    exogenous_features=None,
    default_batch_size=None,
    default_series_length=None
)
```

Build an input_receiver_fn for export_savedmodel which accepts arrays.

Args:

- `exogenous_features`: A dictionary mapping feature keys to exogenous features (either Numpy arrays or Tensors). Used to determine the shapes of placeholders for these features.
- `default_batch_size`: If specified, must be a scalar integer. Sets the batch size in the static shape information of all feature Tensors, which means only this batch size will be accepted by the exported model. If None (default), static shape information for batch sizes is omitted.
- `default_series_length`: If specified, must be a scalar integer. Sets the series length in the static shape information of all feature Tensors, which means only this series length will be accepted by the exported model. If None (default), static shape information for series length is omitted.

Returns:

An input_receiver_fn which may be passed to the Estimator's export_savedmodel.

evaluate

```
evaluate(
    input_fn,
    steps=None,
    hooks=None,
    checkpoint_path=None,
    name=None
)
```

Evaluates the model given evaluation data input_fn.

For each step, calls `input_fn`, which returns one batch of data. Evaluates until: - `steps` batches are processed, or - `input_fn` raises an end-of-input exception (`OutOfRangeError` or `StopIteration`).

Args:

- `input_fn`: Input function returning a tuple of: features - Dictionary of string feature name to `Tensor` or `SparseTensor`. labels - `Tensor` or dictionary of `Tensor` with labels.

- `steps`: Number of steps for which to evaluate model. If `None`, evaluates until `input_fn` raises an end-of-input exception.
- `hooks`: List of `SessionRunHook` subclass instances. Used for callbacks inside the evaluation call.
- `checkpoint_path`: Path of a specific checkpoint to evaluate. If `None`, the latest checkpoint in `model_dir` is used.
- `name`: Name of the evaluation if user needs to run multiple evaluations on different data sets, such as on training data vs test data. Metrics for different evaluations are saved in separate folders, and appear separately in tensorboard.

Returns:

A dict containing the evaluation metrics specified in `model_fn` keyed by name, as well as an entry `global_step` which contains the value of the global step for which this evaluation was performed.

Raises:

- `ValueError`: If `steps <= 0`.
- `ValueError`: If no model has been trained, namely `model_dir`, or the given `checkpoint_path` is empty.

`export_savedmodel`

```
export_savedmodel(
    export_dir_base,
    serving_input_receiver_fn,
    assets_extra=None,
    as_text=False,
    checkpoint_path=None
)
```

Exports inference graph as a SavedModel into given dir.

This method builds a new graph by first calling the serving_input_receiver_fn to obtain feature `Tensor`s, and then calling this `Estimator`'s model_fn to generate the model graph based on those features. It restores the given checkpoint (or, lacking that, the most recent checkpoint) into this graph in a fresh session. Finally it creates a timestamped export directory below the given export_dir_base, and writes a `SavedModel` into it containing a single `MetaGraphDef` saved from this session.

The exported `MetaGraphDef` will provide one `SignatureDef` for each element of the export_outputs dict returned from the model_fn, named using the same keys. One of these keys is always signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY, indicating which signature will be served when a serving request does not specify one. For each signature, the outputs are provided by the corresponding `ExportOutput`s, and the inputs are always the input receivers provided by the serving_input_receiver_fn.

Extra assets may be written into the SavedModel via the extra_assets argument. This should be a dict, where each key gives a destination path (including the filename) relative to the assets.extra directory. The corresponding value gives the full path of the source file to be copied. For example, the simple case of copying a single file without renaming it is specified as `{'my_asset_file.txt': '/path/to/my_asset_file.txt'}`.

Args:

- `export_dir_base`: A string containing a directory in which to create timestamped subdirectories containing exported SavedModels.
- `serving_input_receiver_fn`: A function that takes no argument and returns a `ServingInputReceiver`.
- `assets_extra`: A dict specifying how to populate the assets.extra directory within the exported SavedModel, or `None`

if no extra assets are needed.

- `as_text`: whether to write the SavedModel proto in text format.
- `checkpoint_path`: The checkpoint path to export. If `None` (the default), the most recent checkpoint found within the model directory is chosen.

Returns:

The string path to the exported directory.

Raises:

- `ValueError`: if no serving_input_receiver_fn is provided, no export_outputs are provided, or no checkpoint can be found.

`get_variable_names`

```
get_variable_names()
```

Returns list of all variable names in this model.

Returns:

List of names.

Raises:

- `ValueError`: If the Estimator has not produced a checkpoint yet.

`get_variable_value`

```
get_variable_value(name)
```

Returns value of the variable given by name.

Args:

- `name`: string or a list of string, name of the tensor.

Returns:

Numpy array - value of the tensor.

Raises:

- `ValueError`: If the Estimator has not produced a checkpoint yet.

`latest_checkpoint`

```
latest_checkpoint()
```

Finds the filename of latest saved checkpoint file in `model_dir`.

Returns:

The full path to the latest checkpoint or `None` if no checkpoint was found.

`predict`

```
predict(
    input_fn,
    predict_keys=None,
    hooks=None,
    checkpoint_path=None
)
```

Yields predictions for given features.

Args:

- `input_fn`: Input function returning features which is a dictionary of string feature name to `Tensor` or `SparseTensor`. If it returns a tuple, first item is extracted as features. Prediction continues until `input_fn` raises an end-of-input exception (`OutOfRangeError` or `StopIteration`).
- `predict_keys`: list of `str`, name of the keys to predict. It is used if the `EstimatorSpec.predictions` is a `dict`. If `predict_keys` is used then rest of the predictions will be filtered from the dictionary. If `None`, returns all.
- `hooks`: List of `SessionRunHook` subclass instances. Used for callbacks inside the prediction call.
- `checkpoint_path`: Path of a specific checkpoint to predict. If `None`, the latest checkpoint in `model_dir` is used.

Yields:

Evaluated values of `predictions` tensors.

Raises:

- `ValueError`: Could not find a trained model in model_dir.
- `ValueError`: if batch length of predictions are not same.
- `ValueError`: If there is a conflict between `predict_keys` and `predictions`. For example if `predict_keys` is not `None` but `EstimatorSpec.predictions` is not a `dict`.

`train`

```
train(
    input_fn,
    hooks=None,
    steps=None,
    max_steps=None,
    saving_listeners=None
)
```

Trains a model given training data input_fn.

Args:

- `input_fn`: Input function returning a tuple of: features - `Tensor` or dictionary of string feature name to `Tensor`. labels - `Tensor` or dictionary of `Tensor` with labels.
- `hooks`: List of `SessionRunHook` subclass instances. Used for callbacks inside the training loop.
- `steps`: Number of steps for which to train model. If `None`, train forever or train until input_fn generates the `OutOfRange` error or `StopIteration` exception. 'steps' works incrementally. If you call two times train(steps=10) then training occurs in total 20 steps. If `OutOfRange` or `StopIteration` occurs in the middle, training stops before 20 steps. If you don't want to have incremental behavior please set `max_steps` instead. If set, `max_steps` must be `None`.
- `max_steps`: Number of total steps for which to train model. If `None`, train forever or train until input_fn generates the `OutOfRange` error or `StopIteration` exception. If set, `steps` must be `None`. If `OutOfRange` or `StopIteration` occurs in the middle, training stops before `max_steps` steps. Two calls to `train(steps=100)` means 200 training iterations. On the other hand, two calls to `train(max_steps=100)` means that the second call will not do any iteration since first call did all 100 steps.
- `saving_listeners`: list of `CheckpointSaverListener` objects. Used for callbacks that run immediately before or after checkpoint savings.

Returns:

`self`, for chaining.

Raises:

- `ValueError`: If both `steps` and `max_steps` are not `None`.
- `ValueError`: If either `steps` or `max_steps` is <= 0.
