tf.contrib.learn.LinearEstimator

Contents

Class LinearEstimator
Properties
  config
  model_dir

Class **`LinearEstimator`**

Inherits From: `Estimator`

Defined in `tensorflow/contrib/learn/python/learn/estimators/linear.py`.

Linear model with user specified head.

Train a generalized linear model to predict label value given observation of feature values.

Example: To do poisson regression,

```
sparse_column_a = sparse_column_with_hash_bucket(...)
sparse_column_b = sparse_column_with_hash_bucket(...)

sparse_feature_a_x_sparse_feature_b = crossed_column(...)

estimator = LinearEstimator(
    feature_columns=[sparse_column_a, sparse_feature_a_x_sparse_feature_b],
    head=head_lib.poisson_regression_head())

# Input builders
def input_fn_train: # returns x, y
  ...
def input_fn_eval: # returns x, y
  ...
estimator.fit(input_fn=input_fn_train)
estimator.evaluate(input_fn=input_fn_eval)
estimator.predict(x=x)
```

Input of `fit` and `evaluate` should have following features, otherwise there will be a KeyError:

- if `weight_column_name` is not `None`: key=weight_column_name, value=a `Tensor`
- for column in `feature_columns`:
- if isinstance(column, `SparseColumn`): key=column.name, value=a `SparseTensor`
- if isinstance(column, `WeightedSparseColumn`): {key=id column name, value=a `SparseTensor`, key=weight column name, value=a `SparseTensor`}
- if isinstance(column, `RealValuedColumn`): key=column.name, value=a `Tensor`

Properties

`config`

`model_dir`

Methods

`__init__`

```
__init__(
    feature_columns,
    head,
    model_dir=None,
    weight_column_name=None,
    optimizer=None,
    gradient_clip_norm=None,
    _joint_weights=False,
    config=None,
    feature_engineering_fn=None
)
```

Construct a `LinearEstimator` object.

Args:

- `feature_columns`: An iterable containing all the feature columns used by the model. All items in the set should be instances of classes derived from `FeatureColumn`.
- `head`: An instance of _Head class.
- `model_dir`: Directory to save model parameters, graph, etc. This can also be used to load checkpoints from the directory into a estimator to continue training a previously saved model.
- `weight_column_name`: A string defining feature column name representing weights. It is used to down weight or boost examples during training. It will be multiplied by the loss of the example.
- `optimizer`: An instance of `tf.Optimizer` used to train the model. If `None`, will use an Ftrl optimizer.
- `gradient_clip_norm`: A `float` > 0. If provided, gradients are clipped to their global norm with this clipping ratio. See `tf.clip_by_global_norm` for more details.
- `_joint_weights`: If True use a single (possibly partitioned) variable to store the weights. It's faster, but requires all feature columns are sparse and have the 'sum' combiner. Incompatible with SDCAOptimizer.
- `config`: `RunConfig` object to configure the runtime settings.
- `feature_engineering_fn`: Feature engineering function. Takes features and labels which are the output of `input_fn` and returns features and labels which will be fed into the model.

Returns:

A `LinearEstimator` estimator.

Raises:

- `ValueError`: if optimizer is not supported, e.g., SDCAOptimizer

`evaluate`

```
evaluate(
    x=None,
    y=None,
    input_fn=None,
    feed_fn=None,
    batch_size=None,
    steps=None,
    metrics=None,
    name=None,
    checkpoint_path=None,
    hooks=None,
    log_progress=True
)
```

See `Evaluable`. (deprecated arguments)

SOME ARGUMENTS ARE DEPRECATED. They will be removed after 2016-12-01. Instructions for updating: Estimator is decoupled from Scikit Learn interface by moving into separate class SKCompat. Arguments x, y and batch_size are only available in the SKCompat class, Estimator will only accept input_fn. Example conversion: est = Estimator(...) -> est = SKCompat(Estimator(...))

Raises:

- `ValueError`: If at least one of `x` or `y` is provided, and at least one of `input_fn` or `feed_fn` is provided. Or if `metrics` is not `None` or `dict`.

`export`

```
export(
    export_dir,
    input_fn=export._default_input_fn,
    input_feature_key=None,
    use_deprecated_input_fn=True,
    signature_fn=None,
    prediction_key=None,
    default_batch_size=1,
    exports_to_keep=None,
    checkpoint_path=None
)
```

Exports inference graph into given dir. (deprecated)

THIS FUNCTION IS DEPRECATED. It will be removed after 2017-03-25. Instructions for updating: Please use Estimator.export_savedmodel() instead.

Args:

- `export_dir`: A string containing a directory to write the exported graph and checkpoints.
- `input_fn`: If `use_deprecated_input_fn` is true, then a function that given `Tensor` of `Example` strings, parses it into features that are then passed to the model. Otherwise, a function that takes no argument and returns a tuple of (features, labels), where features is a dict of string key to `Tensor` and labels is a `Tensor` that's currently not used (and so can be `None`).
- `input_feature_key`: Only used if `use_deprecated_input_fn` is false. String key into the features dict returned by `input_fn` that corresponds to a the raw `Example` strings `Tensor` that the exported model will take as input. Can only be `None` if you're using a custom `signature_fn` that does not use the first arg (examples).
- `use_deprecated_input_fn`: Determines the signature format of `input_fn`.

- `signature_fn`: Function that returns a default signature and a named signature map, given `Tensor` of `Example` strings, `dict` of `Tensor`s for features and `Tensor` or `dict` of `Tensor`s for predictions.
- `prediction_key`: The key for a tensor in the `predictions` dict (output from the `model_fn`) to use as the `predictions` input to the `signature_fn`. Optional. If `None`, predictions will pass to `signature_fn` without filtering.
- `default_batch_size`: Default batch size of the `Example` placeholder.
- `exports_to_keep`: Number of exports to keep.
- `checkpoint_path`: the checkpoint path of the model to be exported. If it is `None` (which is default), will use the latest checkpoint in export_dir.

Returns:

The string path to the exported directory. NB: this functionality was added ca. 2016/09/25; clients that depend on the return value may need to handle the case where this function returns None because subclasses are not returning a value.

`export_savedmodel`

```
export_savedmodel(
    export_dir_base,
    serving_input_fn,
    default_output_alternative_key=None,
    assets_extra=None,
    as_text=False,
    checkpoint_path=None,
    graph_rewrite_specs=(GraphRewriteSpec((tag_constants.SERVING,), ()),)
)
```

Exports inference graph as a SavedModel into given dir.

Args:

- `export_dir_base`: A string containing a directory to write the exported graph and checkpoints.
- `serving_input_fn`: A function that takes no argument and returns an `InputFnOps`.
- `default_output_alternative_key`: the name of the head to serve when none is specified. Not needed for single-headed models.
- `assets_extra`: A dict specifying how to populate the assets.extra directory within the exported SavedModel. Each key should give the 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'}`.
- `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.
- `graph_rewrite_specs`: an iterable of `GraphRewriteSpec`. Each element will produce a separate MetaGraphDef within the exported SavedModel, tagged and rewritten as specified. Defaults to a single entry using the default serving tag ("serve") and no rewriting.

Returns:

The string path to the exported directory.

Raises:

- `ValueError`: if an unrecognized export_type is requested.

`fit`

```
fit(
    x=None,
    y=None,
    input_fn=None,
    steps=None,
    batch_size=None,
    monitors=None,
    max_steps=None
)
```

See **`Trainable`**. (deprecated arguments)

SOME ARGUMENTS ARE DEPRECATED. They will be removed after 2016-12-01. Instructions for updating: Estimator is decoupled from Scikit Learn interface by moving into separate class SKCompat. Arguments x, y and batch_size are only available in the SKCompat class, Estimator will only accept input_fn. Example conversion: est = Estimator(...) -> est = SKCompat(Estimator(...))

Raises:

- `ValueError`: If **`x`** or **`y`** are not **`None`** while **`input_fn`** is not **`None`**.
- `ValueError`: If both **`steps`** and **`max_steps`** are not **`None`**.

`get_params`

```
get_params(deep=True)
```

Get parameters for this estimator.

Args:

- `deep`: boolean, optional

  If **`True`**, will return the parameters for this estimator and contained subobjects that are estimators.

Returns:

- `params`: mapping of string to any Parameter names mapped to their values.

`get_variable_names`

```
get_variable_names()
```

Returns list of all variable names in this model.

Returns:

List of names.

`get_variable_value`

```
get_variable_value(name)
```

Returns value of the variable given by name.

Args:

- `name`: string, name of the tensor.

Returns:

Numpy array - value of the tensor.

`partial_fit`

```
partial_fit(
    x=None,
    y=None,
    input_fn=None,
    steps=1,
    batch_size=None,
    monitors=None
)
```

Incremental fit on a batch of samples. (deprecated arguments)

SOME ARGUMENTS ARE DEPRECATED. They will be removed after 2016-12-01. Instructions for updating: Estimator is decoupled from Scikit Learn interface by moving into separate class SKCompat. Arguments x, y and batch_size are only available in the SKCompat class, Estimator will only accept input_fn. Example conversion: est = Estimator(...) -> est = SKCompat(Estimator(...))

This method is expected to be called several times consecutively on different or the same chunks of the dataset. This either can implement iterative training or out-of-core/online training.

This is especially useful when the whole dataset is too big to fit in memory at the same time. Or when model is taking long time to converge, and you want to split up training into subparts.

Args:

- `x`: Matrix of shape [n_samples, n_features...]. Can be iterator that returns arrays of features. The training input samples for fitting the model. If set, **`input_fn`** must be **`None`**.
- `y`: Vector or matrix [n_samples] or [n_samples, n_outputs]. Can be iterator that returns array of labels. The training label values (class labels in classification, real numbers in regression). If set, **`input_fn`** must be **`None`**.
- `input_fn`: Input function. If set, **`x`**, **`y`**, and **`batch_size`** must be **`None`**.
- `steps`: Number of steps for which to train model. If **`None`**, train forever.
- `batch_size`: minibatch size to use on the input, defaults to first dimension of **`x`**. Must be **`None`** if **`input_fn`** is provided.
- `monitors`: List of **`BaseMonitor`** subclass instances. Used for callbacks inside the training loop.

Returns:

`self`, for chaining.

Raises:

- `ValueError`: If at least one of `x` and `y` is provided, and `input_fn` is provided.

`predict`

```
predict(
    x=None,
    input_fn=None,
    batch_size=None,
    outputs=None,
    as_iterable=True
)
```

Returns predictions for given features. (deprecated arguments)

SOME ARGUMENTS ARE DEPRECATED. They will be removed after 2016-12-01. Instructions for updating: Estimator is decoupled from Scikit Learn interface by moving into separate class SKCompat. Arguments x, y and batch_size are only available in the SKCompat class, Estimator will only accept input_fn. Example conversion: est = Estimator(...) -> est = SKCompat(Estimator(...))

Args:

- `x`: Matrix of shape [n_samples, n_features...]. Can be iterator that returns arrays of features. The training input samples for fitting the model. If set, `input_fn` must be `None`.
- `input_fn`: Input function. If set, `x` and 'batch_size' must be `None`.
- `batch_size`: Override default batch size. If set, 'input_fn' must be 'None'.
- `outputs`: list of `str`, name of the output to predict. If `None`, returns all.
- `as_iterable`: If True, return an iterable which keeps yielding predictions for each example until inputs are exhausted. Note: The inputs must terminate if you want the iterable to terminate (e.g. be sure to pass num_epochs=1 if you are using something like read_batch_features).

Returns:

A numpy array of predicted classes or regression values if the constructor's `model_fn` returns a `Tensor` for `predictions` or a `dict` of numpy arrays if `model_fn` returns a `dict`. Returns an iterable of predictions if as_iterable is True.

Raises:

- `ValueError`: If x and input_fn are both provided or both `None`.

`set_params`

```
set_params(**params)
```

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The former have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Args:

- `**params`: Parameters.

Returns:

self

Raises:

- `ValueError`: If params contain invalid names.