tf.contrib.learn.DNNLinearCombinedRegressor

Contents

Class **DNNLinearCombinedRegressor**

Inherits From: `Estimator`

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

See the guide: Learn (contrib) > Estimators

A regressor for TensorFlow Linear and DNN joined training models.

Note: New users must set `fix_global_step_increment_bug=True` when creating an estimator.

Example:

```
sparse_feature_a = sparse_column_with_hash_bucket(...)
sparse_feature_b = sparse_column_with_hash_bucket(...)

sparse_feature_a_x_sparse_feature_b = crossed_column(...)

sparse_feature_a_emb = embedding_column(sparse_id_column=sparse_feature_a,
                                        ...)
sparse_feature_b_emb = embedding_column(sparse_id_column=sparse_feature_b,
                                        ...)

estimator = DNNLinearCombinedRegressor(
    # common settings
    weight_column_name=weight_column_name,
    # wide settings
    linear_feature_columns=[sparse_feature_a_x_sparse_feature_b],
    linear_optimizer=tf.train.FtrlOptimizer(...),
    # deep settings
    dnn_feature_columns=[sparse_feature_a_emb, sparse_feature_b_emb],
    dnn_hidden_units=[1000, 500, 100],
    dnn_optimizer=tf.train.ProximalAdagradOptimizer(...))

# To apply L1 and L2 regularization, you can set optimizers as follows:
tf.train.ProximalAdagradOptimizer(
    learning_rate=0.1,
    l1_regularization_strength=0.001,
    l2_regularization_strength=0.001)
# It is same for FtrlOptimizer.

# Input builders
def input_fn_train: # returns x, y
  ...
def input_fn_eval: # returns x, y
  ...
def input_fn_predict: # returns x, None
  ...
estimator.train(input_fn_train)
estimator.evaluate(input_fn_eval)
estimator.predict(input_fn_predict)
```

Input of `fit`, `train`, and `evaluate` should have following features, otherwise there will be a `KeyError`: if `weight_column_name` is not `None`, a feature with `key=weight_column_name` whose value is a `Tensor`. for each `column` in `dnn_feature_columns` + `linear_feature_columns`: - if `column` is a `SparseColumn`, a feature with `key=column.name` whose `value` is a `SparseTensor`. - if `column` is a `WeightedSparseColumn`, two features: the first with `key` the id column name, the second with `key` the weight column name. Both features' `value` must be a `SparseTensor`. - if `column` is a `RealValuedColumn, a feature with` key=column.name `whose` value `is a` Tensor`.

Properties

`config`

`model_dir`

Methods

`__init__`

```
__init__(
    model_dir=None,
    weight_column_name=None,
    linear_feature_columns=None,
    linear_optimizer=None,
    _joint_linear_weights=False,
    dnn_feature_columns=None,
    dnn_optimizer=None,
    dnn_hidden_units=None,
    dnn_activation_fn=tf.nn.relu,
    dnn_dropout=None,
    gradient_clip_norm=None,
    enable_centered_bias=False,
    label_dimension=1,
    config=None,
    feature_engineering_fn=None,
    embedding_lr_multipliers=None,
    input_layer_min_slice_size=None,
    fix_global_step_increment_bug=False
)
```

Initializes a DNNLinearCombinedRegressor instance. (deprecated arguments)

SOME ARGUMENTS ARE DEPRECATED. They will be removed after 2017-04-15. Instructions for updating: Please set fix_global_step_increment_bug=True and update training steps in your pipeline. See pydoc for details.

★ **Note:** New users must set `fix_global_step_increment_bug=True` when creating an estimator.

Args:

- `model_dir`: Directory to save model parameters, graph and 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.
- `linear_feature_columns`: An iterable containing all the feature columns used by linear part of the model. All items in the set must be instances of classes derived from `FeatureColumn`.
- `linear_optimizer`: An instance of `tf.Optimizer` used to apply gradients to the linear part of the model. If `None`, will use a FTRL optimizer.
- `_joint_linear_weights`: If True a single (possibly partitioned) variable will be used to store the linear model weights. It's faster, but requires that all columns are sparse and have the 'sum' combiner.
- `dnn_feature_columns`: An iterable containing all the feature columns used by deep part of the model. All items in the set must be instances of classes derived from `FeatureColumn`.
- `dnn_optimizer`: An instance of `tf.Optimizer` used to apply gradients to the deep part of the model. If `None`, will use an Adagrad optimizer.
- `dnn_hidden_units`: List of hidden units per layer. All layers are fully connected.
- `dnn_activation_fn`: Activation function applied to each layer. If None, will use `tf.nn.relu`.
- `dnn_dropout`: When not None, the probability we will drop out a given coordinate.
- `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.
- `enable_centered_bias`: A bool. If True, estimator will learn a centered bias variable for each class. Rest of the model structure learns the residual after centered bias.
- `label_dimension`: Number of regression targets per example. This is the size of the last dimension of the labels and

logits `Tensor` objects (typically, these have shape `[batch_size, label_dimension]`).

- `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.
- `embedding_lr_multipliers`: Optional. A dictionary from `EmbeddingColumn` to a `float` multiplier. Multiplier will be used to multiply with learning rate for the embedding variables.
- `input_layer_min_slice_size`: Optional. The min slice size of input layer partitions. If not provided, will use the default of 64M.
- `fix_global_step_increment_bug`: If `False`, the estimator needs two fit steps to optimize both linear and dnn parts. If `True`, this bug is fixed. New users must set this to `True`, but it the default value is `False` for backwards compatibility.

Raises:

- `ValueError`: If both linear_feature_columns and dnn_features_columns are empty at the same time.

`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
)
```

See evaluable.Evaluable.

`export`

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

See BaseEstimator.export. (deprecated)

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

`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) (deprecated arguments)

SOME ARGUMENTS ARE DEPRECATED. They will be removed after 2016-09-15. Instructions for updating: The default behavior of predict() is changing. The default value for as_iterable will change to True, and then the flag will be removed altogether. The behavior of this flag is described below.

SOME ARGUMENTS ARE DEPRECATED. They will be removed after 2017-03-01. Instructions for updating: Please switch to predict_scores, or set `outputs` argument.

By default, returns predicted scores. But this default will be dropped soon. Users should either pass `outputs`, or call `predict_scores` method.

Args:

- `x`: features.
- `input_fn`: Input function. If set, x must be None.
- `batch_size`: Override default batch size.
- `outputs`: list of `str`, name of the output to predict. If `None`, returns scores.
- `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:

Numpy array of predicted scores (or an iterable of predicted scores if as_iterable is True). If `label_dimension == 1`, the shape of the output is `[batch_size]`, otherwise the shape is `[batch_size, label_dimension]`. If `outputs` is set, returns a dict of predictions.

predict_scores

```
predict_scores(
    x=None,
    input_fn=None,
    batch_size=None,
    as_iterable=True
)
```

Returns predicted scores for given features. (deprecated arguments)

SOME ARGUMENTS ARE DEPRECATED. They will be removed after 2016-09-15. Instructions for updating: The default behavior of predict() is changing. The default value for as_iterable will change to True, and then the flag will be removed altogether. The behavior of this flag is described below.

Args:

- `x`: features.
- `input_fn`: Input function. If set, x must be None.
- `batch_size`: Override default batch size.
- `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:

Numpy array of predicted scores (or an iterable of predicted scores if as_iterable is True). If `label_dimension == 1`, the shape of the output is `[batch_size]`, otherwise the shape is `[batch_size, label_dimension]`.

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.
