TensorFlow   API r1.4

tf.contrib.learn.DNNClassifier

Contents

Class **DNNClassifier**

Inherits From: **Estimator**

Defined in **tensorflow/contrib/learn/python/learn/estimators/dnn.py** .

See the guide: Learn (contrib) > Estimators

A classifier for TensorFlow DNN models.

Example:

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

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 = DNNClassifier(
    feature_columns=[sparse_feature_a_emb, sparse_feature_b_emb],
    hidden_units=[1024, 512, 256])

# Or estimator using the ProximalAdagradOptimizer optimizer with
# regularization.
estimator = DNNClassifier(
    feature_columns=[sparse_feature_a_emb, sparse_feature_b_emb],
    hidden_units=[1024, 512, 256],
    optimizer=tf.train.ProximalAdagradOptimizer(
      learning_rate=0.1,
      l1_regularization_strength=0.001
    ))

# Input builders
def input_fn_train: # returns x, y (where y represents label's class index).
  pass
estimator.fit(input_fn=input_fn_train)

def input_fn_eval: # returns x, y (where y represents label's class index).
  pass
estimator.evaluate(input_fn=input_fn_eval)

def input_fn_predict: # returns x, None
  pass
# predict_classes returns class indices.
estimator.predict_classes(input_fn=input_fn_predict)
```

If the user specifies `label_keys` in constructor, labels must be strings from the `label_keys` vocabulary. Example:

```
label_keys = ['label0', 'label1', 'label2']
estimator = DNNClassifier(
    feature_columns=[sparse_feature_a_emb, sparse_feature_b_emb],
    hidden_units=[1024, 512, 256],
    label_keys=label_keys)

def input_fn_train: # returns x, y (where y is one of label_keys).
  pass
estimator.fit(input_fn=input_fn_train)

def input_fn_eval: # returns x, y (where y is one of label_keys).
  pass
estimator.evaluate(input_fn=input_fn_eval)
def input_fn_predict: # returns x, None
# predict_classes returns one of label_keys.
estimator.predict_classes(input_fn=input_fn_predict)
```

Input of `fit` 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 `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__(
    hidden_units,
    feature_columns,
    model_dir=None,
    n_classes=2,
    weight_column_name=None,
    optimizer=None,
    activation_fn=tf.nn.relu,
    dropout=None,
    gradient_clip_norm=None,
    enable_centered_bias=False,
    config=None,
    feature_engineering_fn=None,
    embedding_lr_multipliers=None,
    input_layer_min_slice_size=None,
    label_keys=None
)
```

Initializes a DNNClassifier instance.

Args:

- `hidden_units` : List of hidden units per layer. All layers are fully connected. Ex. `[64, 32]` means first layer has 64 nodes and second one has 32.
- `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` .
- `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.
- `n_classes` : number of label classes. Default is binary classification. It must be greater than 1. Note: Class labels are integers representing the class index (i.e. values from 0 to n_classes-1). For arbitrary label values (e.g. string labels), convert to class indices first.
- `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 Adagrad optimizer.
- `activation_fn` : Activation function applied to each layer. If `None` , will use tf.nn.relu. Note that a string containing the unqualified name of the op may also be provided, e.g., "relu", "tanh", or "sigmoid".
- `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.

- `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.
- `label_keys` : Optional list of strings with size `[n_classes]` defining the label vocabulary. Only supported for `n_classes` > 2.

Returns:

A `DNNClassifier` estimator.

Raises:

- `ValueError` : If `n_classes` < 2.

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=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_classes, or set `outputs` argument.

By default, returns predicted classes. But this default will be dropped soon. Users should either pass `outputs` , or call `predict_classes` 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 classes.
- `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 classes with shape [batch_size] (or an iterable of predicted classes if as_iterable is True). Each predicted class is represented by its class index (i.e. integer from 0 to n_classes-1). If `outputs` is set, returns a dict of predictions.

predict_classes

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

Returns predicted classes 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 classes with shape [batch_size] (or an iterable of predicted classes if as_iterable is True). Each predicted class is represented by its class index (i.e. integer from 0 to n_classes-1).

predict_proba

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

Returns predicted probabilities 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 and y 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 probabilities with shape batch_size, n_classes .

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.

Stay Connected

Blog

GitHub

Twitter

Support

Issue Tracker

Release Notes

Stack Overflow

English

Terms | **Privacy**