tf.estimator.classifier_parse_example_spec

```
classifier_parse_example_spec(
    feature_columns,
    label_key,
    label_dtype=tf.int64,
    label_default=None,
    weight_column=None
)
```

Defined in `tensorflow/python/estimator/canned/parsing_utils.py`.

Generates parsing spec for tf.parse_example to be used with classifiers.

If users keep data in tf.Example format, they need to call tf.parse_example with a proper feature spec. There are two main things that this utility helps:

- Users need to combine parsing spec of features with labels and weights (if any) since they are all parsed from same tf.Example instance. This utility combines these specs.
- It is difficult to map expected label by a classifier such as `DNNClassifier` to corresponding tf.parse_example spec. This utility encodes it by getting related information from users (key, dtype).

Example output of parsing spec:

```
# Define features and transformations
feature_b = tf.feature_column.numeric_column(...)
feature_c_bucketized = tf.feature_column.bucketized_column(
  tf.feature_column.numeric_column("feature_c"), ...)
feature_a_x_feature_c = tf.feature_column.crossed_column(
    columns=["feature_a", feature_c_bucketized], ...)

feature_columns = [feature_b, feature_c_bucketized, feature_a_x_feature_c]
parsing_spec = tf.estimator.classifier_parse_example_spec(
    feature_columns, label_key='my-label', label_dtype=tf.string)

# For the above example, classifier_parse_example_spec would return the dict:
assert parsing_spec == {
  "feature_a": parsing_ops.VarLenFeature(tf.string),
  "feature_b": parsing_ops.FixedLenFeature([1], dtype=tf.float32),
  "feature_c": parsing_ops.FixedLenFeature([1], dtype=tf.float32)
  "my-label" : parsing_ops.FixedLenFeature([1], dtype=tf.string)
}
```

Example usage with a classifier:

```
feature_columns = # define features via tf.feature_column
estimator = DNNClassifier(
    n_classes=1000,
    feature_columns=feature_columns,
    weight_column='example-weight',
    label_vocabulary=['photos', 'keep', ...],
    hidden_units=[256, 64, 16])
# This label configuration tells the classifier the following:
# * weights are retrieved with key 'example-weight'
# * label is string and can be one of the following ['photos', 'keep', ...]
# * integer id for label 'photos' is 0, 'keep' is 1, ...

# Input builders
def input_fn_train():  # Returns a tuple of features and labels.
  features = tf.contrib.learn.read_keyed_batch_features(
      file_pattern=train_files,
      batch_size=batch_size,
      # creates parsing configuration for tf.parse_example
      features=tf.estimator.classifier_parse_example_spec(
          feature_columns,
          label_key='my-label',
          label_dtype=tf.string,
          weight_column='example-weight'),
      reader=tf.RecordIOReader)
   labels = features.pop('my-label')
   return features, labels

estimator.train(input_fn=input_fn_train)
```

Args:

- `feature_columns` : An iterable containing all feature columns. All items should be instances of classes derived from `_FeatureColumn` .
- `label_key` : A string identifying the label. It means tf.Example stores labels with this key.
- `label_dtype` : A `tf.dtype` identifies the type of labels. By default it is `tf.int64` . If user defines a `label_vocabulary` , this should be set as `tf.string` . `tf.float32` labels are only supported for binary classification.
- `label_default` : used as label if label_key does not exist in given tf.Example. An example usage: let's say `label_key` is 'clicked' and tf.Example contains clicked data only for positive examples in following format `key:clicked, value:1` . This means that if there is no data with key 'clicked' it should count as negative example by setting `label_deafault=0` . Type of this value should be compatible with `label_dtype` .
- `weight_column` : A string or a `_NumericColumn` created by `tf.feature_column.numeric_column` defining feature column representing weights. It is used to down weight or boost examples during training. It will be multiplied by the loss of the example. If it is a string, it is used as a key to fetch weight tensor from the `features` . If it is a `_NumericColumn` , raw tensor is fetched by key `weight_column.key` , then weight_column.normalizer_fn is applied on it to get weight tensor.

Returns:

A dict mapping each feature key to a `FixedLenFeature` or `VarLenFeature` value.

Raises:

- `ValueError` : If label is used in `feature_columns` .
- `ValueError` : If weight_column is used in `feature_columns` .
- `ValueError` : If any of the given `feature_columns` is not a `_FeatureColumn` instance.

- `ValueError`: If `weight_column` is not a `_NumericColumn` instance.
- `ValueError`: if label_key is None.

Last updated November 2, 2017.

Stay Connected

Blog

GitHub

Twitter

Support

Issue Tracker

Release Notes

Stack Overflow

English

Terms | **Privacy**