tf.feature_column.categorical_column_with_vocabulary_file

```
categorical_column_with_vocabulary_file(
    key,
    vocabulary_file,
    vocabulary_size,
    num_oov_buckets=0,
    default_value=None,
    dtype=tf.string
)
```

Defined in `tensorflow/python/feature_column/feature_column.py`.

A `_CategoricalColumn` with a vocabulary file.

Use this when your inputs are in string or integer format, and you have a vocabulary file that maps each value to an integer ID. By default, out-of-vocabulary values are ignored. Use either (but not both) of `num_oov_buckets` and `default_value` to specify how to include out-of-vocabulary values.

For input dictionary `features`, `features[key]` is either `Tensor` or `SparseTensor`. If `Tensor`, missing values can be represented by `-1` for int and `''` for string. Note that these values are independent of the `default_value` argument.

Example with `num_oov_buckets`: File '/us/states.txt' contains 50 lines, each with a 2-character U.S. state abbreviation. All inputs with values in that file are assigned an ID 0-49, corresponding to its line number. All other values are hashed and assigned an ID 50-54.

```
states = categorical_column_with_vocabulary_file(
    key='states', vocabulary_file='/us/states.txt', vocabulary_size=50,
    num_oov_buckets=5)
columns = [states, ...]
features = tf.parse_example(..., features=make_parse_example_spec(columns))
linear_prediction = linear_model(features, columns)
```

Example with `default_value`: File '/us/states.txt' contains 51 lines - the first line is 'XX', and the other 50 each have a 2-character U.S. state abbreviation. Both a literal 'XX' in input, and other values missing from the file, will be assigned ID 0. All others are assigned the corresponding line number 1-50.

```
states = categorical_column_with_vocabulary_file(
    key='states', vocabulary_file='/us/states.txt', vocabulary_size=51,
    default_value=0)
columns = [states, ...]
features = tf.parse_example(..., features=make_parse_example_spec(columns))
linear_prediction, _, _ = linear_model(features, columns)
```

And to make an embedding with either:

```
columns = [embedding_column(states, 3),...]
features = tf.parse_example(..., features=make_parse_example_spec(columns))
dense_tensor = input_layer(features, columns)
```

Args:

- `key`: A unique string identifying the input feature. It is used as the column name and the dictionary key for feature

parsing configs, feature `Tensor` objects, and feature columns.

- `vocabulary_file`: The vocabulary file name.
- `vocabulary_size`: Number of the elements in the vocabulary. This must be no greater than length of `vocabulary_file`, if less than length, later values are ignored.
- `num_oov_buckets`: Non-negative integer, the number of out-of-vocabulary buckets. All out-of-vocabulary inputs will be assigned IDs in the range `[vocabulary_size, vocabulary_size+num_oov_buckets)` based on a hash of the input value. A positive `num_oov_buckets` can not be specified with `default_value`.
- `default_value`: The integer ID value to return for out-of-vocabulary feature values, defaults to `-1`. This can not be specified with a positive `num_oov_buckets`.
- `dtype`: The type of features. Only string and integer types are supported.

Returns:

A `_CategoricalColumn` with a vocabulary file.

Raises:

- `ValueError`: `vocabulary_file` is missing.
- `ValueError`: `vocabulary_size` is missing or < 1.
- `ValueError`: `num_oov_buckets` is a negative integer.
- `ValueError`: `num_oov_buckets` and `default_value` are both specified.
- `ValueError`: `dtype` is neither string nor integer.

Last updated November 2, 2017.