

# Feature Columns

Learn about feature columns and how they're used to extract data features.

## Chapter Goals:

- Learn about feature columns and how they're used
- Implement a function that creates a list of feature columns

## A. Overview

Before we get into using a dataset of parsed protocol buffers, we need to first discuss *feature columns*. In TensorFlow, a feature column is how we specify what kind of data a feature contains. In this chapter, we'll focus on the two most common types of feature data: numeric and categorical data.

Feature columns are incredibly useful for converting raw data into an input layer for a machine learning model. Once we have a list of feature columns, we can use them to combine `tf.Tensor` and `tf.SparseTensor` feature data into a single input layer. We'll discuss more of this in the next chapter.

## B. Numeric features

For numeric features, we create a feature column using `tf.feature_column.numeric_column`. The function takes in the feature name as a required argument.

```
import tensorflow as tf

nc = tf.feature_column.numeric_column(
    'GPA', shape=5, dtype=tf.float32)

print(nc)
```



In the example above, `nc` represents a numeric feature column for the feature called `'GPA'`. We used the `shape` keyword argument to specify that the

feature must be 1-D and contain 5 elements. We also set the feature's datatype to `tf.float32`.

Other less commonly used keyword arguments for the function are `default_value` and `normalizer_fn`.

The `default_value` keyword argument sets the default value for the feature column if the feature is missing from the protocol buffer. If `None` (which is the default), it means that the specified feature must be present in all entries in order for us to use the numeric column without error.

The `normalizer_fn` keyword argument takes in a function that normalizes the feature data. This would be used in a similar way to how `map` is used for datasets.

### C. Categorical features

To create a feature column for categorical data, we need a *vocabulary* for the data. A vocabulary simply refers to the list of possible categories for the data. There are two functions we use to create categorical feature columns:

- `tf.feature_column.categorical_column_with_vocabulary_list`
- `tf.feature_column.categorical_column_with_vocabulary_file`

From this point on, we'll refer to these functions without their `tf.feature_column` prefix, for brevity.

```
import tensorflow as tf

cc1 = tf.feature_column.categorical_column_with_vocabulary_list(
    'name', ['a', 'b', 'c'])

cc2 = tf.feature_column.categorical_column_with_vocabulary_list(
    'name', [1, 2, 3])

cc3 = tf.feature_column.categorical_column_with_vocabulary_list(
    'name', ['a', 'b', 'NA'], default_value=2)

cc4 = tf.feature_column.categorical_column_with_vocabulary_list(
    'name', ['a', 'b', 'c'], num_oov_buckets=2)
```



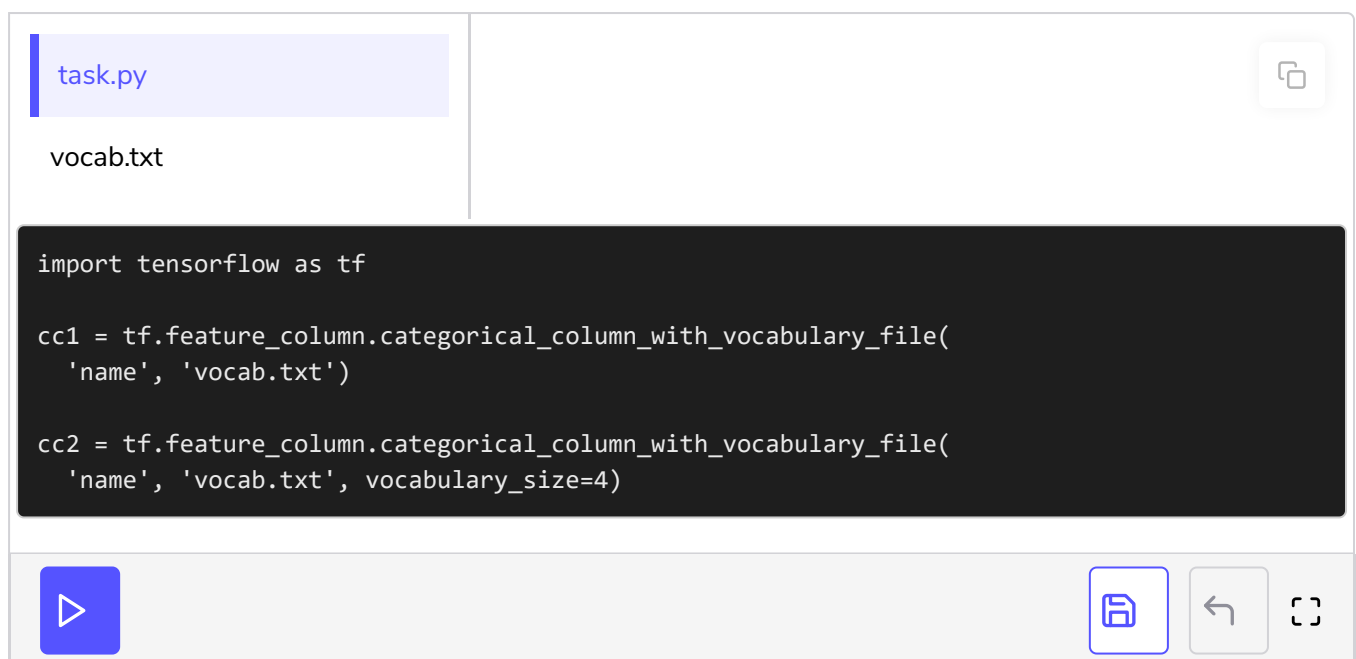
The `categorical_column_with_vocabulary_list` function takes in two required arguments: the feature name and the list of categories (vocabulary).

The `dtype` keyword argument specifies the datatype for the feature column. If `dtype` is not passed in, the datatype is inferred from the vocabulary list. In our example, `cc1` has type `tf.string` and `cc2` has type `tf.int32`.

The `default_value` argument specifies a non-negative default index for an OOV (out-of-vocabulary) category. In our example, each OOV value for `cc3` would be mapped to the `'NA'` category.

The `num_oov_buckets` argument specifies the number of OOV "dummy" categories to create. These "dummy" values will be filled when we actually run our TensorFlow computation graph. In our example, we create two "dummy" categories in `cc4` for placing OOV values into.

Note that `default_value` and `num_oov_buckets` keyword arguments cannot be used together. `default_value` assigns an already existing category to catch OOV values, while `num_oov_buckets` creates new categories to put OOV values in.



The screenshot shows a Jupyter Notebook interface. On the left, there is a file browser with two files: `task.py` and `vocab.txt`. The main area is a code cell with the following Python code:

```
import tensorflow as tf

cc1 = tf.feature_column.categorical_column_with_vocabulary_file(
    'name', 'vocab.txt')

cc2 = tf.feature_column.categorical_column_with_vocabulary_file(
    'name', 'vocab.txt', vocabulary_size=4)
```

At the bottom of the interface, there are several icons: a blue play button (run), a blue save icon, a grey undo icon, and a grey full-screen icon.

The `categorical_column_with_vocabulary_file` function takes in two required arguments: the feature name and the file containing the vocabulary. The vocabulary file must contain a single category per line.

The `vocabulary_size` keyword argument specifies how many of the

The `vocabulary_size` keyword argument specifies how many of the vocabulary categories to use. For `cc2`, we specified that the feature column only use the first 4 categories of the 11 categories listed in the file. When we don't set this argument, the function will infer the vocabulary size based on the number of lines in the file.

The remaining keyword arguments are identical to the ones in `categorical_column_with_vocabulary_list`.

## D. Indicator wrapping

Unlike numeric feature columns, categorical feature columns cannot be used directly to create the input layer for a machine learning model. In order to actually use a categorical feature column, we need to convert the categorical data into numeric values. One of the most common ways to do this is with the `tf.feature_column.indicator_column` function.



```
import tensorflow as tf

cc = tf.feature_column.categorical_column_with_vocabulary_file(
    'name', 'vocab.txt')

ic = tf.feature_column.indicator_column(cc)
print(ic)
```

The indicator column wrapper converts the categorical data into a count vector. The count vector gives a count of how many times each vocabulary category appears in the feature data. You'll see an explicit example of count vectors in the next chapter.

## Time to Code!

In this chapter you'll be completing the `create_feature_columns` function, which creates a list of feature columns for a variety of features. The feature configurations are given by the input dictionary, `config`.

Specifically, you'll be creating three helper functions: `create_list_column`, `create_file_column`, and `create_numeric_column`. These functions are used to create the necessary feature columns for `create_feature_columns`:

```
import tensorflow as tf

def create_feature_columns(config, example_spec, output_features=None):
    if output_features is None:
        output_features = config.keys()
    feature_columns = []
    for feature_name in output_features:
        dtype = example_spec[feature_name].dtype
        feature_config = config[feature_name]
        # HELPER FUNCTIONS USED
        if 'vocab_list' in feature_config:
            feature_col = create_list_column(feature_name, feature_config, dtype)
        elif 'vocab_file' in feature_config:
            feature_col = create_file_column(feature_name, feature_config, dtype)
        else:
            feature_col = create_numeric_column(feature_name, feature_config, dtype)
        feature_columns.append(feature_col)
    return feature_columns
```

The `create_list_column` function creates a categorical feature from a vocabulary list. To do this, we use the `tf.feature_column.categorical_column_with_vocabulary_list` function.

Set `vocab_feature_col` equal to the specified function applied with `feature_name` and `feature_config['vocab_list']` as the first two arguments, and `dtype=dtype` for the keyword argument.

Since `vocab_feature_col` is a categorical column, we need to wrap it with an indicator column.

Set `feature_col` equal to `tf.feature_column.indicator_column` applied with `vocab_feature_col` as the only argument. Then return `feature_col`.

```
import tensorflow as tf

def create_list_column(feature_name, feature_config, dtype):
    # CODE HERE
    pass
```



The `create_file_column` function creates a categorical feature from a vocabulary file. To do this, we use the `tf.feature_column.categorical_column_with_vocabulary_file` function.

Set `vocab_feature_col` equal to the specified function applied with `feature_name` and `feature_config['vocab_file']` as the first two arguments, and `dtype=dtype` for the keyword argument.

Since `vocab_feature_col` is a categorical column, we need to wrap it with an indicator column.






Set `feature_col` equal to `tf.feature_column.indicator_column` applied with `vocab_feature_col` as the only argument. Then return `feature_col`.

task.py

majors.txt

```
import tensorflow as tf

def create_file_column(feature_name, feature_config, dtype):
    # CODE HERE
    pass
```



The `create_numeric_column` function creates a numeric feature. To do this, we use the `tf.feature_column.numeric_column` function.

Set `feature_col` equal to the specified function applied with `feature_name` as the first argument, `feature_config['shape']` for the `shape` keyword argument, and `dtype` for the `dtype` keyword argument. Then return `feature_col`.

```
import tensorflow as tf

def create_numeric_column(feature_name, feature_config, dtype):
    # CODE HERE
    pass
```

