Table 30-1. (*continued*)

Function name	Description
Categorical column with identity – **tf.feature_column.categorical_column_with_identity()**	This function creates a one-hot encoded output of a categorical column containing identities, e.g, ['0', '1', '2', '3'].
Categorical column with vocabulary list – **tf.feature_column.categorical_ column_with_vocabulary_list()**	This function creates a one-hot encoded output of a categorical column with strings. It maps each string to an integer based on a vocabulary list. However, if the vocabulary list is long, it is best to create a file containing the vocabulary and use the function **tf.feature_column.categorical_ column_with_vocabulary_file().**
Categorical column with hash bucket – **tf.feature_column.categorical_ column_with_hash_buckets()**	This function specifies the number of categories by using the hash of the inputs. It is used when it is not possible to create a vocabulary for the number of categories due to memory considerations.
Crossed column – **tf.feature_columns.crossed_column()**	The function gives the ability to combine multiple input features into a single input feature.
Bucketized column – **tf.feature_column.bucketized_column()**	The function splits a column of numerical inputs into buckets to form new classes based on a specified set of numerical ranges.

The High-Level TensorFlow APIs

The high-level API provides simplified API calls that encapsulate lots of the details that are typically involved in creating a deep learning TensorFlow model. These high-level abstractions make it easier to develop powerful deep learning models quickly with fewer lines of code.

Estimator API

The Estimator API is a high-level TensorFlow functionality that is aimed at reducing the complexity involved in building machine learning models by exposing methods that abstract common models and processes. There are two ways of working with Estimators, and they include

- **Using the premade Estimators**: The premade Estimators are black box models made available by the TensorFlow team for building common machine learning/deep learning architectures such as linear regression/classification, Random forest regression/classification and deep neural networks for regression and classification. An illustration of the premade Estimators as subclasses of the Estimator class is shown in Figure 30-5.

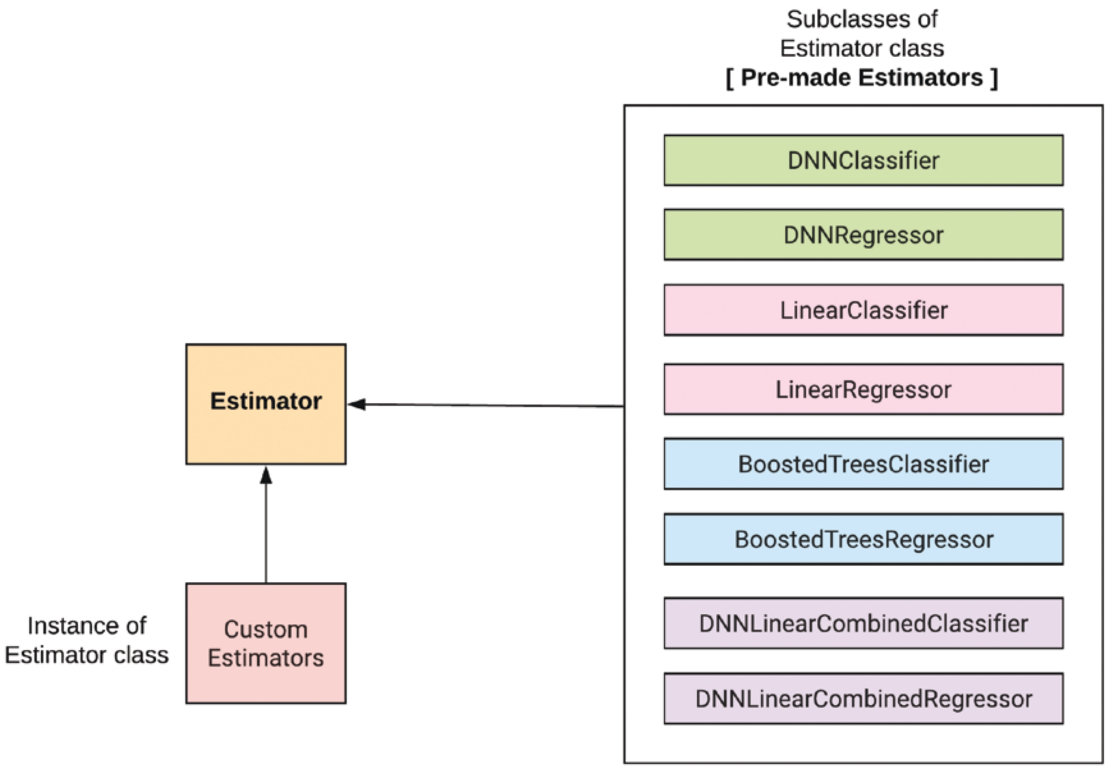

Figure 30-5. *Estimator class API hierarchy*

- **Creating a custom Estimator**: It is also possible to use the low-level TensorFlow methods to create a custom black box model for easy reusability. To do this, you must put your code in a method called the **model_fn**. The model function will include code that defines operations such as the labels or predictions, loss function, the training operations, and the operations for evaluation.

The Estimator class exposes four major methods, namely, the **fit()**, **evaluate()**, **predict()**, and **export_savedmodel()** methods. The **fit()** method is called to train the data by running a loop of training operations. The **evaluate()** method is called to evaluate the model performance by looping through a set of evaluation operations. The **predict()** method uses the trained model to make predictions, while the **export_savedmodel()** method is used for exporting the trained model to a specified directory. For both the premade and custom Estimators, we must write a method to build the data input pipeline into the model. This pipeline is built for both the training and evaluation data inputs. This is further illustrated in Figure 30-6.

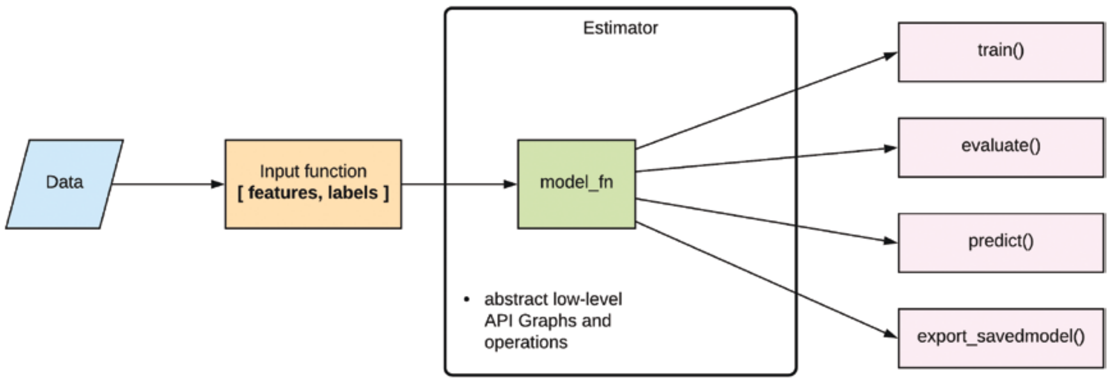

Figure 30-6. *Estimator data input pipeline*

Keras API

Keras provides a high-level specification for developing deep neural network models. The Keras API was initially separate from TensorFlow and only provided an interface for model building with TensorFlow as one of the frameworks running at the backend. However, in TensorFlow 2.0, Keras is an integral part of the TensorFlow codebase as preferred high-level API.

The Keras API version internal to TensorFlow is available from the ‘tf.keras’ package, whereas the broader Keras API blueprint that is not tied to a specific backend will remain available from the ‘keras’ package. In summary, when working with the ‘keras’ package, the backend can run with either TensorFlow, Microsoft CNTK, or Theano. On the other hand, working with ‘tf.keras’ provides a TensorFlow only version which is tightly integrated and compatible with all of the functionality of the core TensorFlow library.

In this book, we will focus on **‘tf.Keras’** as a high-level API of TensorFlow.

The Anatomy of a Keras Program

The Keras **‘Model’** forms the core of a Keras program. A ‘Model’ is first constructed, then it is compiled. Next, the compiled model is trained and evaluated using their respective training and evaluation datasets. Upon successful evaluation using the relevant metrics, the model is then used for making predictions on previously unseen data samples. Figure 30-7 shows the program flow for modeling with Keras.

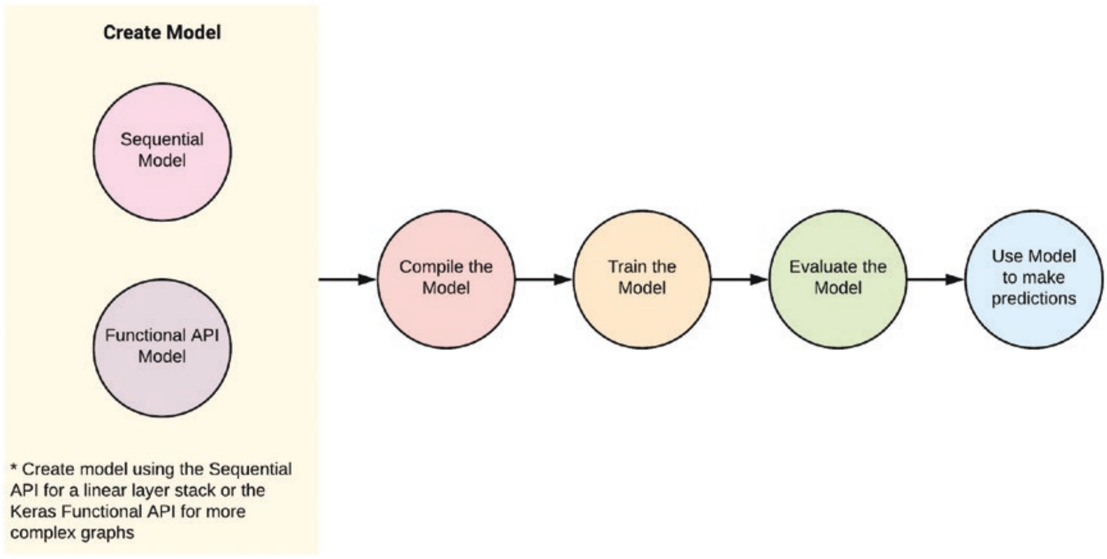
Create Model
Sequential Model
Functional API Model
* Create model using the Sequential API for a linear layer stack or the Keras Functional API for more complex graphs
Compile the Model
Train the Model
Evaluate the Model
Use Model to make predictions

Figure 30-7. *The anatomy of a Keras program*

As shown in Figure 30-7, the Keras ‘Model’ can be constructed using the Sequential API ‘tf.keras.Sequential’ or the Keras Functional API which defines a model instance ‘tf. keras.Model’. The Sequential model is the simplest method for creating a linear stack of