- hyper-tune.sh: [script to run a training job with hyper-parameter tuning on Cloud MLE]
- single-instance-training.sh: [script to run a single instance training job on Cloud MLE]
- online-prediction.sh: [script to execute an online prediction job on Cloud MLE]
- create-prediction-service.sh: [script to create a prediction service on Cloud MLE]

- hptuning_config: [configuration file for hyper-parameter tuning on Cloud MLE]
- gpu_hptuning_config.yaml: [configuration file for hyper-parameter tuning with GPU training on Cloud MLE]

NOTE: FOLLOW THESE INSTRUCTIONS TO RUN THE EXAMPLES FOR TRAINING ON CLOUD MACHINE LEARNING ENGINE

1. Launch a Notebook Instance on GCP AI Platform.
2. Pull the code repository.
3. Navigate to the book folder. Run the scripts in the sub-folder `tensorflow'.
4. Should you choose to work with Google Colab, authenticate the user by running the code

```
from google.colab import auth
    auth.authenticate_user()
```

The TensorFlow Model

Now let's briefly examine the TF model code in the file '**model.py**'.

```
import six
```

```
import tensorflow as tf
from tensorflow.python.estimator.model_fn import ModeKeys as Modes

# Define the format of your input data including unused columns.
CSV_COLUMNS = [
    'sepal_length', 'sepal_width', 'petal_length',
    'petal_width', 'class'
]
CSV_COLUMN_DEFAULTS = [[0.0], [0.0], [0.0], [0.0], [“]]
LABEL_COLUMN = 'class'
LABELS = ['setosa', 'versicolor', 'virginica']

# Define the initial ingestion of each feature used by your model.
# Additionally, provide metadata about the feature.
INPUT_COLUMNS = [
    # Continuous base columns.
    tf.feature_column.numeric_column('sepal_length'),
    tf.feature_column.numeric_column('sepal_width'),
    tf.feature_column.numeric_column('petal_length'),
    tf.feature_column.numeric_column('petal_width')
]

UNUSED_COLUMNS = set(CSV_COLUMNS) - {col.name for col in INPUT_COLUMNS} - \
    {LABEL_COLUMN}

def build_estimator(config, hidden_units=None, learning_rate=None):
    """Deep NN Classification model for predicting flower class.
    Args:
        config: (tf.contrib.learn.RunConfig) defining the runtime
        environment for
          the estimator (including model_dir).
        hidden_units: [int], the layer sizes of the DNN (input layer first)
        learning_rate: (int), the learning rate for the optimizer.
    Returns:
        A DNNClassifier
    """
    (sepal_length, sepal_width, petal_length, petal_width) = INPUT_COLUMNS
```

```
    columns = [
        sepal_length,
        sepal_width,
        petal_length,
        petal_width,
    ]

    return tf.estimator.DNNClassifier(
      config=config,
      feature_columns=columns,
      hidden_units=hidden_units or [256, 128, 64],
      n_classes = 3,
      optimizer=tf.train.AdamOptimizer(learning_rate)
    )

def parse_label_column(label_string_tensor):
  """Parses a string tensor into the label tensor.
  Args:
    label_string_tensor: Tensor of dtype string. Result of parsing the CSV
      column specified by LABEL_COLUMN.
  Returns:
    A Tensor of the same shape as label_string_tensor, should return
    an int64 Tensor representing the label index for classification tasks,
    and a float32 Tensor representing the value for a regression task.
  """
  # Build a Hash Table inside the graph
  table = tf.contrib.lookup.index_table_from_tensor(tf.constant(LABELS))

  # Use the hash table to convert string labels to ints and one-hot encode
  return table.lookup(label_string_tensor)

# [START serving-function]

def csv_serving_input_fn():
    """Build the serving inputs."""
    csv_row = tf.placeholder(shape=[None], dtype=tf.string)
    features = _decode_csv(csv_row)
```

```
    # Ignore label column
    features.pop(LABEL_COLUMN)
    return tf.estimator.export.ServingInputReceiver(features,
                                                    {'csv_row': csv_row})

def json_serving_input_fn():
    """Build the serving inputs."""
    inputs = {}
    for feat in INPUT_COLUMNS:
        inputs[feat.name] = tf.placeholder(shape=[None], dtype=feat.dtype)

    return tf.estimator.export.ServingInputReceiver(inputs, inputs)

# [END serving-function]

SERVING_FUNCTIONS = {
  'JSON': json_serving_input_fn,
  'CSV': csv_serving_input_fn
}

def _decode_csv(line):
    """Takes the string input tensor and returns a dict of rank-2 tensors."""

    # Takes a rank-1 tensor and converts it into rank-2 tensor
    row_columns = tf.expand_dims(line, -1)
    columns = tf.decode_csv(row_columns, record_defaults=CSV_COLUMN_DEFAULTS)
    features = dict(zip(CSV_COLUMNS, columns))

    # Remove unused columns
    for col in UNUSED_COLUMNS:
      features.pop(col)
    return features

def input_fn(filenames,
         num_epochs=None,
         shuffle=True,
         skip_header_lines=1,
         batch_size=200):
```

```
    """Generates features and labels for training or evaluation.
    This uses the input pipeline based approach using file name queue
    to read data so that entire data is not loaded in memory.
    """
    dataset = tf.data.TextLineDataset(filenames).skip(skip_header_lines).map(
      _decode_csv)

    if shuffle:
        dataset = dataset.shuffle(buffer_size=batch_size * 10)
    iterator = dataset.repeat(num_epochs).batch(
        batch_size).make_one_shot_iterator()
    features = iterator.get_next()
    return features, parse_label_column(features.pop(LABEL_COLUMN))
```

The code for the most part is self-explanatory; however, the reader should take note of the following points:

- The function 'build_estimator' uses the canned Estimator API to train a 'DNNClassifier' model on Cloud MLE. The learning rate and hidden units of the model can be adjusted and tuned as a hyper-parameter during training.
- The methods 'csv_serving_input_fn' and 'json_serving_input_fn' define the serving inputs for CSV and JSON serving input formats.
- The method 'input_fn' uses the TensorFlow Dataset API to build the input pipelines for training and evaluation on Cloud MLE. This method calls the private method _decode_csv() to convert the CSV columns to Tensors.

The Application Logic

Let's see the application logic in the file '**task.py**'.

```
import argparse
import json
import os
```