binary_classification

August 30, 2021

[ ]:
```
#@title Copyright 2020 Google LLC. Double-click here for license information.
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

1 Binary Classification

So far, you've only created regression models. That is, you created models that produced floating-point predictions, such as, "houses in this neighborhood costs N thousand dollars." In this Colab, you'll create and evaluate a binary classification model. That is, you'll create a model that answers a binary question. In this exercise, the binary question will be, "Are houses in this neighborhood above a certain price?"

1.1 Learning Objectives:

After doing this Colab, you'll know how to:

- Convert a regression question into a classification question.
- Modify the classification threshold and determine how that modification influences the model.
- Experiment with different classification metrics to determine your model's effectiveness.

1.2 The Dataset

Like several of the previous Colabs, this Colab uses the California Housing Dataset.

1.3 Call the import statements

The following code imports the necessary modules.

[1]:
```
#@title Load the imports
```

```
# from __future__ import absolute_import, division, print_function,␣
↪unicode_literals

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import feature_column
from tensorflow.keras import layers
from matplotlib import pyplot as plt

# The following lines adjust the granularity of reporting.
pd.options.display.max_rows = 10
pd.options.display.float_format = "{:.1f}".format
# tf.keras.backend.set_floatx('float32')

print("Ran the import statements.")
```

```
Ran the import statements.
```

[2]:
```
from jupyterthemes import jtplot
jtplot.style('onedork', figsize=(16,9))
```

1.4 Load the datasets from the internet

The following code cell loads the separate .csv files and creates the following two pandas DataFrames:

- `train_df`, which contains the training set
- `test_df`, which contains the test set

[3]:
```
train_df = pd.read_csv("https://download.mlcc.google.com/mledu-datasets/
↪california_housing_train.csv")
test_df = pd.read_csv("https://download.mlcc.google.com/mledu-datasets/
↪california_housing_test.csv")
train_df = train_df.reindex(np.random.permutation(train_df.index)) # shuffle␣
↪the training set
```

Unlike some of the previous Colabs, the preceding code cell did not scale the label (`median_house_value`). The following section ("Normalize values") provides an alternative approach.

1.5 Normalize values

When creating a model with multiple features, the values of each feature should cover roughly the same range. For example, if one feature's range spans 500 to 100,000 and another feature's range spans 2 to 12, then the model will be difficult or impossible to train. Therefore, you should normalize features in a multi-feature model.

The following code cell normalizes datasets by converting each raw value (including the label) to its Z-score. A **Z-score** is the number of standard deviations from the mean for a particular raw value. For example, consider a feature having the following characteristics:

- The mean is 60.
- The standard deviation is 10.

The raw value 75 would have a Z-score of +1.5:

```
Z-score = (75 - 60) / 10 = +1.5
```

The raw value 38 would have a Z-score of -2.2:

```
Z-score = (38 - 60) / 10 = -2.2
```

[4]:
```
# Calculate the Z-scores of each column in the training set and
# write those Z-scores into a new pandas DataFrame named train_df_norm.
train_df_mean = train_df.mean()
train_df_std = train_df.std()
train_df_norm = (train_df - train_df_mean)/train_df_std

# Examine some of the values of the normalized training set. Notice that most
# Z-scores fall between -2 and +2.
train_df_norm.head()
```

[4]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
560          1.3      -1.4                -0.5         -0.1            -0.0
5303         0.7      -0.8                 1.2          0.2             0.2
3886         0.8      -0.9                -0.2          0.7             0.8
16618       -1.6       1.5                 0.0          0.1             0.2
2371         1.0      -0.8                -1.2          5.0             3.6

       population  households  median_income  median_house_value
560           0.1         0.1           -0.5                -0.7
5303         -0.1         0.3            0.3                 0.4
3886          0.7         0.8            0.0                -0.1
16618        -0.5        -0.5           -1.0                -1.0
2371          5.4         4.1            0.7                -0.2
```

[5]:
```
# Calculate the Z-scores of each column in the test set and
# write those Z-scores into a new pandas DataFrame named test_df_norm.
test_df_mean = test_df.mean()
test_df_std  = test_df.std()
test_df_norm = (test_df - test_df_mean)/test_df_std
```

1.6 Task 1: Create a binary label

In classification problems, the label for every example must be either 0 or 1. Unfortunately, the natural label in the California Housing Dataset, `median_house_value`, contains floating-point values like 80,100 or 85,700 rather than 0s and 1s, while the normalized version of `median_house_values` contains floating-point values primarily between -3 and +3.

Your task is to create a new column named `median_house_value_is_high` in both the training set and the test set . If the `median_house_value` is higher than a certain arbitrary value (defined by `threshold`), then set `median_house_value_is_high` to 1. Otherwise, set `median_house_value_is_high` to 0.

Hint: The cells in the `median_house_value_is_high` column must each hold `1` and `0`, not `True` and `False`. To convert `True` and `False` to `1` and `0`, call the pandas DataFrame function `astype(float)`.

[6]:
```
threshold = 265000 # This is the 75th percentile for median house values.
train_df_norm["median_house_value_is_high"] = (train_df["median_house_value"] >␣
↪threshold).astype(float)
test_df_norm["median_house_value_is_high"] = (test_df["median_house_value"] >␣
↪threshold).astype(float)

# Print out a few example cells from the beginning and
# middle of the training set, just to make sure that
# your code created only 0s and 1s in the newly created
# median_house_value_is_high column
train_df_norm["median_house_value_is_high"].head(8000)
```

[6]:
```
560      0.0
5303     0.0
3886     0.0
16618    0.0
2371     0.0
        ...
10184    1.0
6775     1.0
11850    0.0
3473     0.0
13772    1.0
Name: median_house_value_is_high, Length: 8000, dtype: float64
```

[ ]:
```
#@title Double-click for possible solutions.

# We arbitrarily set the threshold to 265,000, which is
# the 75th percentile for median house values.  Every neighborhood
# with a median house price above 265,000 will be labeled 1,
# and all other neighborhoods will be labeled 0.
threshold = 265000
train_df_norm["median_house_value_is_high"] = (train_df["median_house_value"] >␣
↪threshold).astype(float)
test_df_norm["median_house_value_is_high"] = (test_df["median_house_value"] >␣
↪threshold).astype(float)
train_df_norm["median_house_value_is_high"].head(8000)


# Alternatively, instead of picking the threshold
# based on raw house values, you can work with Z-scores.
# For example, the following possible solution uses a Z-score
# of +1.0 as the threshold, meaning that no more
# than 16% of the values in median_house_value_is_high
# will be labeled 1.

# threshold_in_Z = 1.0
# train_df_norm["median_house_value_is_high"] =␣
↪(train_df_norm["median_house_value"] > threshold_in_Z).astype(float)
# test_df_norm["median_house_value_is_high"] =␣
↪(test_df_norm["median_house_value"] > threshold_in_Z).astype(float)
```

1.7 Represent features in feature columns

This code cell specifies the features that you'll ultimately train the model on and how each of those features will be represented. The transformations (collected in `feature_layer`) don't actually get applied until you pass a DataFrame to it, which will happen when we train the model.

[7]:
```
# Create an empty list that will eventually hold all created feature columns.
feature_columns = []

# Create a numerical feature column to represent median_income.
median_income = tf.feature_column.numeric_column("median_income")
feature_columns.append(median_income)

# Create a numerical feature column to represent total_rooms.
tr = tf.feature_column.numeric_column("total_rooms")
feature_columns.append(tr)

# Convert the list of feature columns into a layer that will later be fed into
# the model.
feature_layer = layers.DenseFeatures(feature_columns)

# Print the first 3 and last 3 rows of the feature_layer's output when applied
# to train_df_norm:
feature_layer(dict(train_df_norm))
```

[7]:
```
<tf.Tensor: shape=(17000, 2), dtype=float32, numpy=
array([[-0.5340118 , -0.05397581],
       [ 0.28562745,  0.1831859 ],
       [ 0.0366961 ,  0.67494094],
       …,
       [ 0.17845596,  2.4809482 ],
       [ 0.42130816, -0.34297365],
       [ 0.48933193, -0.06819634]], dtype=float32)>
```

1.8 Define functions that build and train a model

The following code cell defines two functions:

- `create_model(my_learning_rate, feature_layer, my_metrics)`, which defines the model's topography.
- `train_model(model, dataset, epochs, label_name, batch_size, shuffle)`, uses input features and labels to train the model.

Prior exercises used ReLU as the activation function. By contrast, this exercise uses sigmoid as the activation function.

[8]:
```python
#@title Define the functions that create and train a model.
def create_model(my_learning_rate, feature_layer, my_metrics):
  """Create and compile a simple classification model."""
  # Most simple tf.keras models are sequential.
  model = tf.keras.models.Sequential()

  # Add the feature layer (the list of features and how they are represented)
  # to the model.
  model.add(feature_layer)

  # Funnel the regression value through a sigmoid function.
  model.add(tf.keras.layers.Dense(units=1, input_shape=(1,),
                                  activation=tf.sigmoid),)

  # Call the compile method to construct the layers into a model that
  # TensorFlow can execute.  Notice that we're using a different loss
  # function for classification than for regression.
  model.compile(optimizer=tf.keras.optimizers.
 ↪RMSprop(learning_rate=my_learning_rate),
                loss=tf.keras.losses.BinaryCrossentropy(),
                metrics=my_metrics)

  return model


def train_model(model, dataset, epochs, label_name,
                batch_size=None, shuffle=True):
  """Feed a dataset into the model in order to train it."""

  # The x parameter of tf.keras.Model.fit can be a list of arrays, where
  # each array contains the data for one feature.  Here, we're passing
  # every column in the dataset. Note that the feature_layer will filter
  # away most of those columns, leaving only the desired columns and their
  # representations as features.
  features = {name:np.array(value) for name, value in dataset.items()}
  label = np.array(features.pop(label_name))
  history = model.fit(x=features, y=label, batch_size=batch_size,
                              epochs=epochs, shuffle=shuffle)

  # The list of epochs is stored separately from the rest of history.
  epochs = history.epoch

  # Isolate the classification metric for each epoch.
  hist = pd.DataFrame(history.history)

  return epochs, hist

print("Defined the create_model and train_model functions.")
```

```
Defined the create_model and train_model functions.
```

1.9 Define a plotting function

The following matplotlib function plots one or more curves, showing how various classification metrics change with each epoch.

[9]:
```
#@title Define the plotting function.
def plot_curve(epochs, hist, list_of_metrics):
  """Plot a curve of one or more classification metrics vs. epoch."""
  # list_of_metrics should be one of the names shown in:
  # https://www.tensorflow.org/tutorials/structured_data/
 ↪imbalanced_data#define_the_model_and_metrics

  plt.figure()
  plt.xlabel("Epoch")
  plt.ylabel("Value")

  for m in list_of_metrics:
    x = hist[m]
    plt.plot(epochs[1:], x[1:], label=m)

  plt.legend()

print("Defined the plot_curve function.")
```

```
Defined the plot_curve function.
```

1.10 Invoke the creating, training, and plotting functions

The following code cell calls specify the hyperparameters, and then invokes the functions to create and train the model, and then to plot the results.

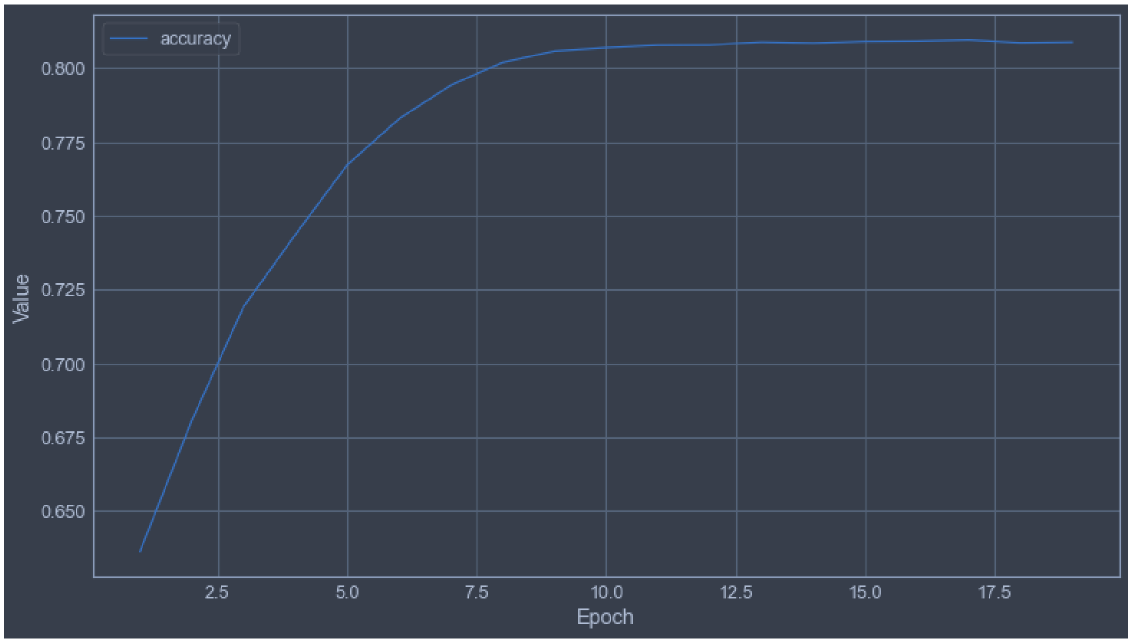
[10]:
```
# The following variables are the hyperparameters.
learning_rate = 0.001
epochs = 20
batch_size = 100
```

```python
label_name = "median_house_value_is_high"
classification_threshold = 0.35

# Establish the metrics the model will measure.
METRICS = [
           tf.keras.metrics.BinaryAccuracy(name='accuracy',
                                           threshold=classification_threshold),
          ]

# Establish the model's topography.
my_model = None
my_model = create_model(learning_rate, feature_layer, METRICS)

# Train the model on the training set.
epochs, hist = train_model(my_model, train_df_norm, epochs,
                           label_name, batch_size)

# Plot a graph of the metric(s) vs. epochs.
list_of_metrics_to_plot = ['accuracy']

plot_curve(epochs, hist, list_of_metrics_to_plot)
```

```
Epoch 1/20
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'longitude': <tf.Tensor
'ExpandDims_3:0' shape=(100, 1) dtype=float32>, 'latitude': <tf.Tensor
'ExpandDims_2:0' shape=(100, 1) dtype=float32>, 'housing_median_age': <tf.Tensor
'ExpandDims_1:0' shape=(100, 1) dtype=float32>, 'total_rooms': <tf.Tensor
'ExpandDims_8:0' shape=(100, 1) dtype=float32>, 'total_bedrooms': <tf.Tensor
'ExpandDims_7:0' shape=(100, 1) dtype=float32>, 'population': <tf.Tensor
'ExpandDims_6:0' shape=(100, 1) dtype=float32>, 'households': <tf.Tensor
'ExpandDims:0' shape=(100, 1) dtype=float32>, 'median_income': <tf.Tensor
'ExpandDims_5:0' shape=(100, 1) dtype=float32>, 'median_house_value': <tf.Tensor
'ExpandDims_4:0' shape=(100, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'longitude': <tf.Tensor
'ExpandDims_3:0' shape=(100, 1) dtype=float32>, 'latitude': <tf.Tensor
'ExpandDims_2:0' shape=(100, 1) dtype=float32>, 'housing_median_age': <tf.Tensor
'ExpandDims_1:0' shape=(100, 1) dtype=float32>, 'total_rooms': <tf.Tensor
'ExpandDims_8:0' shape=(100, 1) dtype=float32>, 'total_bedrooms': <tf.Tensor
'ExpandDims_7:0' shape=(100, 1) dtype=float32>, 'population': <tf.Tensor
'ExpandDims_6:0' shape=(100, 1) dtype=float32>, 'households': <tf.Tensor
'ExpandDims:0' shape=(100, 1) dtype=float32>, 'median_income': <tf.Tensor
'ExpandDims_5:0' shape=(100, 1) dtype=float32>, 'median_house_value': <tf.Tensor
'ExpandDims_4:0' shape=(100, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
170/170 [==============================] - 1s 3ms/step - loss: 0.5335 -
```

```
accuracy: 0.5669
Epoch 2/20
170/170 [==============================] - 0s 3ms/step - loss: 0.4964 -
accuracy: 0.6364
Epoch 3/20
170/170 [==============================] - 0s 3ms/step - loss: 0.4675 -
accuracy: 0.6807
Epoch 4/20
170/170 [==============================] - 0s 3ms/step - loss: 0.4446 -
accuracy: 0.7192
Epoch 5/20
170/170 [==============================] - 0s 3ms/step - loss: 0.4277 -
accuracy: 0.7436
Epoch 6/20
170/170 [==============================] - 0s 3ms/step - loss: 0.4151 -
accuracy: 0.7672
Epoch 7/20
170/170 [==============================] - 0s 3ms/step - loss: 0.4064 -
accuracy: 0.7828
Epoch 8/20
170/170 [==============================] - 0s 3ms/step - loss: 0.4013 -
accuracy: 0.7942
Epoch 9/20
170/170 [==============================] - 0s 3ms/step - loss: 0.3989 -
accuracy: 0.8019
Epoch 10/20
170/170 [==============================] - 0s 3ms/step - loss: 0.3976 -
accuracy: 0.8058
Epoch 11/20
170/170 [==============================] - 0s 3ms/step - loss: 0.3971 -
accuracy: 0.8069
Epoch 12/20
170/170 [==============================] - 0s 3ms/step - loss: 0.3969 -
accuracy: 0.8078
Epoch 13/20
170/170 [==============================] - 0s 3ms/step - loss: 0.3967 -
accuracy: 0.8079
Epoch 14/20
170/170 [==============================] - 0s 3ms/step - loss: 0.3967 -
accuracy: 0.8088
Epoch 15/20
170/170 [==============================] - 0s 3ms/step - loss: 0.3966 -
accuracy: 0.8085
Epoch 16/20
170/170 [==============================] - 0s 3ms/step - loss: 0.3966 -
accuracy: 0.8091
Epoch 17/20
170/170 [==============================] - 0s 3ms/step - loss: 0.3966 -
```

```
accuracy: 0.8092
Epoch 18/20
170/170 [==============================] - 0s 3ms/step - loss: 0.3966 -
accuracy: 0.8096
Epoch 19/20
170/170 [==============================] - 0s 3ms/step - loss: 0.3966 -
accuracy: 0.8086
Epoch 20/20
170/170 [==============================] - 0s 3ms/step - loss: 0.3966 -
accuracy: 0.8088
```



Accuracy should gradually improve during training (until it can improve no more).

1.11 Evaluate the model against the test set

At the end of model training, you ended up with a certain accuracy against the *training set*. Invoke the following code cell to determine your model's accuracy against the *test set*.

[11]:
```
features = {name:np.array(value) for name, value in test_df_norm.items()}
label = np.array(features.pop(label_name))

my_model.evaluate(x = features, y = label, batch_size=batch_size)
```

```
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'longitude': <tf.Tensor
'ExpandDims_3:0' shape=(100, 1) dtype=float32>, 'latitude': <tf.Tensor
'ExpandDims_2:0' shape=(100, 1) dtype=float32>, 'housing_median_age': <tf.Tensor
'ExpandDims_1:0' shape=(100, 1) dtype=float32>, 'total_rooms': <tf.Tensor
'ExpandDims_8:0' shape=(100, 1) dtype=float32>, 'total_bedrooms': <tf.Tensor
'ExpandDims_7:0' shape=(100, 1) dtype=float32>, 'population': <tf.Tensor
'ExpandDims_6:0' shape=(100, 1) dtype=float32>, 'households': <tf.Tensor
'ExpandDims:0' shape=(100, 1) dtype=float32>, 'median_income': <tf.Tensor
'ExpandDims_5:0' shape=(100, 1) dtype=float32>, 'median_house_value': <tf.Tensor
'ExpandDims_4:0' shape=(100, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
30/30 [==============================] - 0s 2ms/step - loss: 0.4069 - accuracy:
0.8013
```

[11]:
```
[0.4068581163883209, 0.8013333082199097]
```

1.12 Task 2: How accurate is your model really?

Is your model valuable?

[ ]:
```
#@title Double-click for a possible answer to Task 2.

# A perfect model would make 100% accurate predictions.
# Our model makes 80% accurate predictions. 80% sounds
# good, but note that a model that always guesses
# "median_house_value_is_high is False" would be 75%
# accurate.
```

1.13 Task 3: Add precision and recall as metrics

Relying solely on accuracy, particularly for a class-imbalanced data set (like ours), can be a poor way to judge a classification model. Modify the code in the following code cell to enable the model to measure not only accuracy but also precision and recall. We have added accuracy and precision; your task is to add recall. See the TensorFlow Reference for details.

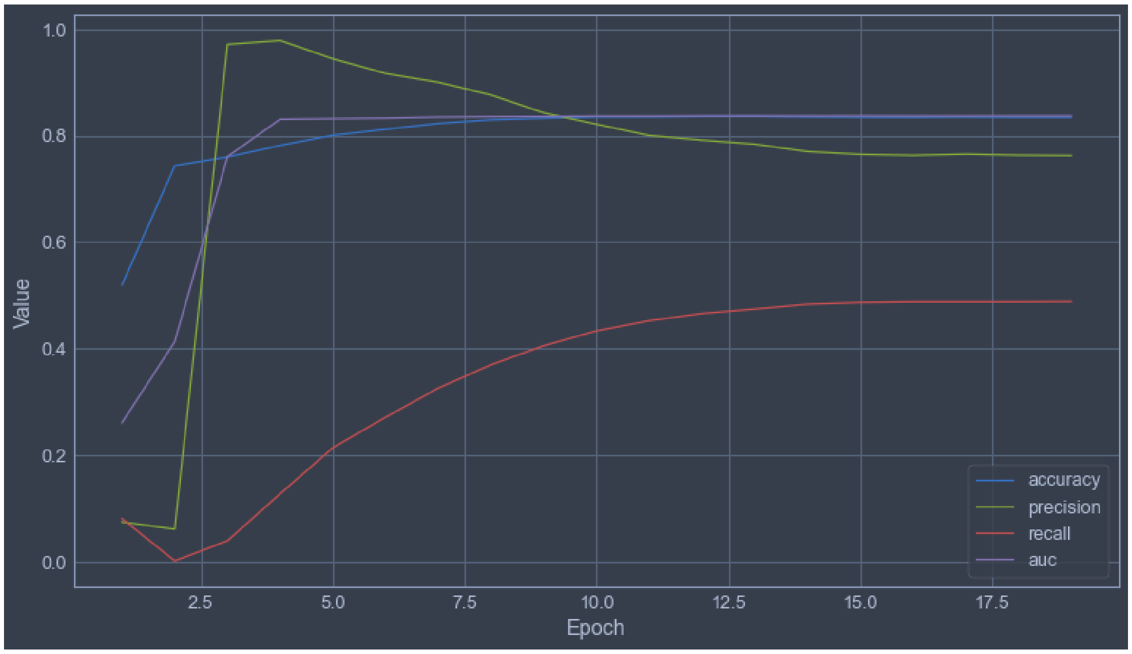
[12]:
```
# The following variables are the hyperparameters.
learning_rate = 0.001
epochs = 20
batch_size = 100
classification_threshold = 0.5
label_name = "median_house_value_is_high"

# Modify the following definition of METRICS to generate
# not only accuracy and precision, but also recall:
METRICS = [
      tf.keras.metrics.BinaryAccuracy(name='accuracy',
                                      threshold=classification_threshold),
      tf.keras.metrics.Precision(thresholds=classification_threshold,
                                 name='precision'
                                 ),
      tf.keras.metrics.Recall(thresholds=classification_threshold,
                              name='recall'
```

```
                            ),
      tf.keras.metrics.AUC(num_thresholds=100,
                           name='auc')
]

# Establish the model's topography.
my_model = None
my_model = create_model(learning_rate, feature_layer, METRICS)

# Train the model on the training set.
epochs, hist = train_model(my_model, train_df_norm, epochs,
                           label_name, batch_size)

# Plot metrics vs. epochs
list_of_metrics_to_plot = ['accuracy', 'precision', 'recall', 'auc']
plot_curve(epochs, hist, list_of_metrics_to_plot)
```

```
Epoch 1/20
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'longitude': <tf.Tensor
'ExpandDims_3:0' shape=(100, 1) dtype=float32>, 'latitude': <tf.Tensor
'ExpandDims_2:0' shape=(100, 1) dtype=float32>, 'housing_median_age': <tf.Tensor
'ExpandDims_1:0' shape=(100, 1) dtype=float32>, 'total_rooms': <tf.Tensor
'ExpandDims_8:0' shape=(100, 1) dtype=float32>, 'total_bedrooms': <tf.Tensor
'ExpandDims_7:0' shape=(100, 1) dtype=float32>, 'population': <tf.Tensor
'ExpandDims_6:0' shape=(100, 1) dtype=float32>, 'households': <tf.Tensor
'ExpandDims:0' shape=(100, 1) dtype=float32>, 'median_income': <tf.Tensor
'ExpandDims_5:0' shape=(100, 1) dtype=float32>, 'median_house_value': <tf.Tensor
'ExpandDims_4:0' shape=(100, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
WARNING:tensorflow:Layers in a Sequential model should only have a single input
tensor, but we receive a <class 'dict'> input: {'longitude': <tf.Tensor
'ExpandDims_3:0' shape=(100, 1) dtype=float32>, 'latitude': <tf.Tensor
'ExpandDims_2:0' shape=(100, 1) dtype=float32>, 'housing_median_age': <tf.Tensor
'ExpandDims_1:0' shape=(100, 1) dtype=float32>, 'total_rooms': <tf.Tensor
'ExpandDims_8:0' shape=(100, 1) dtype=float32>, 'total_bedrooms': <tf.Tensor
'ExpandDims_7:0' shape=(100, 1) dtype=float32>, 'population': <tf.Tensor
'ExpandDims_6:0' shape=(100, 1) dtype=float32>, 'households': <tf.Tensor
'ExpandDims:0' shape=(100, 1) dtype=float32>, 'median_income': <tf.Tensor
'ExpandDims_5:0' shape=(100, 1) dtype=float32>, 'median_house_value': <tf.Tensor
'ExpandDims_4:0' shape=(100, 1) dtype=float32>}
Consider rewriting this model with the Functional API.
170/170 [==============================] - 2s 6ms/step - loss: 0.8306 -
accuracy: 0.3299 - precision: 0.1042 - recall: 0.2213 - auc: 0.2341
Epoch 2/20
170/170 [==============================] - 1s 6ms/step - loss: 0.7207 -
accuracy: 0.5196 - precision: 0.0747 - recall: 0.0810 - auc: 0.2612
Epoch 3/20
```

```
170/170 [==============================] - 2s 10ms/step - loss: 0.6302 -
accuracy: 0.7435 - precision: 0.0615 - recall: 0.0019 - auc: 0.4133
Epoch 4/20
170/170 [==============================] - 2s 13ms/step - loss: 0.5604 -
accuracy: 0.7598 - precision: 0.9711 - recall: 0.0396 - auc: 0.7605
Epoch 5/20
170/170 [==============================] - 1s 6ms/step - loss: 0.5101 -
accuracy: 0.7814 - precision: 0.9784 - recall: 0.1279 - auc: 0.8306
Epoch 6/20
170/170 [==============================] - 1s 6ms/step - loss: 0.4754 -
accuracy: 0.8005 - precision: 0.9440 - recall: 0.2143 - auc: 0.8317
Epoch 7/20
170/170 [==============================] - 1s 6ms/step - loss: 0.4509 -
accuracy: 0.8118 - precision: 0.9165 - recall: 0.2715 - auc: 0.8326
Epoch 8/20
170/170 [==============================] - 2s 10ms/step - loss: 0.4333 -
accuracy: 0.8226 - precision: 0.8999 - recall: 0.3261 - auc: 0.8347
Epoch 9/20
170/170 [==============================] - 2s 12ms/step - loss: 0.4206 -
accuracy: 0.8295 - precision: 0.8762 - recall: 0.3699 - auc: 0.8354
Epoch 10/20
170/170 [==============================] - 1s 6ms/step - loss: 0.4115 -
accuracy: 0.8326 - precision: 0.8425 - recall: 0.4057 - auc: 0.8357
Epoch 11/20
170/170 [==============================] - 1s 6ms/step - loss: 0.4053 -
accuracy: 0.8349 - precision: 0.8211 - recall: 0.4335 - auc: 0.8364
Epoch 12/20
170/170 [==============================] - 2s 12ms/step - loss: 0.4015 -
accuracy: 0.8350 - precision: 0.8002 - recall: 0.4526 - auc: 0.8364
Epoch 13/20
170/170 [==============================] - 2s 13ms/step - loss: 0.3992 -
accuracy: 0.8359 - precision: 0.7913 - recall: 0.4660 - auc: 0.8367
Epoch 14/20
170/170 [==============================] - 2s 14ms/step - loss: 0.3979 -
accuracy: 0.8360 - precision: 0.7837 - recall: 0.4745 - auc: 0.8368
Epoch 15/20
170/170 [==============================] - 2s 13ms/step - loss: 0.3973 -
accuracy: 0.8349 - precision: 0.7701 - recall: 0.4834 - auc: 0.8368
Epoch 16/20
170/170 [==============================] - 2s 13ms/step - loss: 0.3969 -
accuracy: 0.8344 - precision: 0.7647 - recall: 0.4867 - auc: 0.8369
Epoch 17/20
170/170 [==============================] - 2s 13ms/step - loss: 0.3968 -
accuracy: 0.8342 - precision: 0.7629 - recall: 0.4879 - auc: 0.8367
Epoch 18/20
170/170 [==============================] - 2s 13ms/step - loss: 0.3967 -
accuracy: 0.8346 - precision: 0.7651 - recall: 0.4879 - auc: 0.8367
Epoch 19/20
```

```
170/170 [==============================] - 2s 14ms/step - loss: 0.3967 -
accuracy: 0.8342 - precision: 0.7632 - recall: 0.4879 - auc: 0.8368
Epoch 20/20
170/170 [==============================] - 2s 13ms/step - loss: 0.3966 -
accuracy: 0.8342 - precision: 0.7625 - recall: 0.4883 - auc: 0.8369
```



[ ]:

```
#@title Double-click to view the solution for Task 3.

# The following variables are the hyperparameters.
learning_rate = 0.001
epochs = 20
batch_size = 100
classification_threshold = 0.35
label_name = "median_house_value_is_high"

# Here is the updated definition of METRICS:
METRICS = [
      tf.keras.metrics.BinaryAccuracy(name='accuracy',
                                      threshold=classification_threshold),
      tf.keras.metrics.Precision(thresholds=classification_threshold,
                                 name='precision'
                                 ),
      tf.keras.metrics.Recall(thresholds=classification_threshold,
                              name="recall"),
]
```

```
# Establish the model's topography.
my_model = create_model(learning_rate, feature_layer, METRICS)

# Train the model on the training set.
epochs, hist = train_model(my_model, train_df_norm, epochs,
                           label_name, batch_size)

# Plot metrics vs. epochs
list_of_metrics_to_plot = ['accuracy', "precision", "recall"]
plot_curve(epochs, hist, list_of_metrics_to_plot)


# The new graphs suggest that precision and recall are
# somewhat in conflict. That is, improvements to one of
# those metrics may hurt the other metric.
```

1.14 Task 4: Experiment with the classification threshold (if time permits)

Experiment with different values for `classification_threshold` in the code cell within "Invoke the creating, training, and plotting functions." What value of `classification_threshold` produces the highest accuracy?

[ ]:
```
#@title Double-click to view the solution for Task 4.

# The following variables are the hyperparameters.
learning_rate = 0.001
epochs = 20
batch_size = 100
classification_threshold = 0.52
label_name = "median_house_value_is_high"

# Here is the updated definition of METRICS:
METRICS = [
      tf.keras.metrics.BinaryAccuracy(name='accuracy',
                                      threshold=classification_threshold),
      tf.keras.metrics.Precision(thresholds=classification_threshold,
                                 name='precision'
                                 ),
      tf.keras.metrics.Recall(thresholds=classification_threshold,
                              name="recall"),
]

# Establish the model's topography.
my_model = create_model(learning_rate, feature_layer, METRICS)

# Train the model on the training set.
epochs, hist = train_model(my_model, train_df_norm, epochs,
                              label_name, batch_size)

# Plot metrics vs. epochs
list_of_metrics_to_plot = ['accuracy', "precision", "recall"]
plot_curve(epochs, hist, list_of_metrics_to_plot)

# A `classification_threshold` of slightly over 0.5
# appears to produce the highest accuracy (about 83%).
# Raising the `classification_threshold` to 0.9 drops
# accuracy by about 5%.  Lowering the
# `classification_threshold` to 0.3 drops accuracy by
# about 3%.
```

1.15 Task 5: Summarize model performance (if time permits)

If time permits, add one more metric that attempts to summarize the model's overall performance.

[ ]:
```
#@title Double-click to view the solution for Task 5.

# The following variables are the hyperparameters.
learning_rate = 0.001
epochs = 20
batch_size = 100
label_name = "median_house_value_is_high"

# AUC is a reasonable "summary" metric for
# classification models.
# Here is the updated definition of METRICS to
# measure AUC:
METRICS = [
      tf.keras.metrics.AUC(num_thresholds=100, name='auc'),
]

# Establish the model's topography.
my_model = create_model(learning_rate, feature_layer, METRICS)

# Train the model on the training set.
epochs, hist = train_model(my_model, train_df_norm, epochs,
                              label_name, batch_size)

# Plot metrics vs. epochs
list_of_metrics_to_plot = ['auc']
plot_curve(epochs, hist, list_of_metrics_to_plot)
```