Basic Feature Engineering in Keras

Learning Objectives

1. Create an input pipeline using tf.data
2. Engineer features to create categorical, crossed, and numerical feature columns

Introduction

In this lab, we utilize feature engineering to improve the prediction of housing prices using a Keras Sequential Model.

Each learning objective will correspond to a **#TODO** in the student lab notebook -- try to complete that notebook first before reviewing this solution notebook.

Start by importing the necessary libraries for this lab.

In [1]:
```
# Run the chown command to change the ownership
!sudo chown -R jupyter:jupyter /home/jupyter/training-data-analyst
```

In [2]:
```
# Install Sklearn
# scikit-learn simple and efficient tools for predictive data analysis
# Built on NumPy, SciPy, and matplotlib
!python3 -m pip install --user sklearn
```

```
Collecting sklearn
  Downloading sklearn-0.0.tar.gz (1.1 kB)
Requirement already satisfied: scikit-learn in /opt/conda/lib/python3.7/site-packages (f
rom sklearn) (0.24.2)
Requirement already satisfied: numpy>=1.13.3 in /opt/conda/lib/python3.7/site-packages
(from scikit-learn->sklearn) (1.19.5)
Requirement already satisfied: scipy>=0.19.1 in /opt/conda/lib/python3.7/site-packages
(from scikit-learn->sklearn) (1.6.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python3.7/site-pac
kages (from scikit-learn->sklearn) (2.1.0)
Requirement already satisfied: joblib>=0.11 in /opt/conda/lib/python3.7/site-packages (f
rom scikit-learn->sklearn) (1.0.1)
Building wheels for collected packages: sklearn
  Building wheel for sklearn (setup.py) ... done
  Created wheel for sklearn: filename=sklearn-0.0-py2.py3-none-any.whl size=1316 sha256=
34e4de9b7508ad477e38cfa4368ad566310a0163724381d5f2b88953e446042a
  Stored in directory: /home/jupyter/.cache/pip/wheels/46/ef/c3/157e41f5ee1372d1be90b09f
74f82b10e391eaacca8f22d33e
Successfully built sklearn
Installing collected packages: sklearn
Successfully installed sklearn-0.0
```

In [3]:
```
# Ensure the right version of Tensorflow is installed.
!pip freeze | grep tensorflow==2.1 || pip install tensorflow==2.1
```

```
Collecting tensorflow==2.1
  Downloading tensorflow-2.1.0-cp37-cp37m-manylinux2010_x86_64.whl (421.8 MB)
     |▌                               | 421.8 MB 15 kB/s s eta 0:00:013
```

```
Collecting keras-applications>=1.0.8
  Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
     |████████████████████████████████| 50 kB 8.2 MB/s  eta 0:00:01
Collecting tensorboard<2.2.0,>=2.1.0
  Downloading tensorboard-2.1.1-py3-none-any.whl (3.8 MB)
     |████████████████████████████████| 3.8 MB 73.2 MB/s eta 0:00:01
Requirement already satisfied: keras-preprocessing>=1.1.0 in /opt/conda/lib/python3.7/si
te-packages (from tensorflow==2.1) (1.1.2)
Requirement already satisfied: termcolor>=1.1.0 in /opt/conda/lib/python3.7/site-package
s (from tensorflow==2.1) (1.1.0)
Requirement already satisfied: six>=1.12.0 in /opt/conda/lib/python3.7/site-packages (fr
om tensorflow==2.1) (1.16.0)
Requirement already satisfied: absl-py>=0.7.0 in /opt/conda/lib/python3.7/site-packages
(from tensorflow==2.1) (0.10.0)
Collecting astor>=0.6.0
  Downloading astor-0.8.1-py2.py3-none-any.whl (27 kB)
Requirement already satisfied: opt-einsum>=2.3.2 in /opt/conda/lib/python3.7/site-packag
es (from tensorflow==2.1) (3.3.0)
Collecting tensorflow-estimator<2.2.0,>=2.1.0rc0
  Downloading tensorflow_estimator-2.1.0-py2.py3-none-any.whl (448 kB)
     |████████████████████████████████| 448 kB 59.5 MB/s eta 0:00:01
Requirement already satisfied: wrapt>=1.11.1 in /opt/conda/lib/python3.7/site-packages
(from tensorflow==2.1) (1.12.1)
Requirement already satisfied: grpcio>=1.8.6 in /opt/conda/lib/python3.7/site-packages
(from tensorflow==2.1) (1.37.1)
Requirement already satisfied: wheel>=0.26 in /opt/conda/lib/python3.7/site-packages (fr
om tensorflow==2.1) (0.36.2)
Collecting scipy==1.4.1
  Downloading scipy-1.4.1-cp37-cp37m-manylinux1_x86_64.whl (26.1 MB)
     |████████████████████████████████| 26.1 MB 31.5 MB/s eta 0:00:01
Requirement already satisfied: numpy<2.0,>=1.16.0 in /opt/conda/lib/python3.7/site-packa
ges (from tensorflow==2.1) (1.19.5)
Collecting gast==0.2.2
  Downloading gast-0.2.2.tar.gz (10 kB)
Requirement already satisfied: google-pasta>=0.1.6 in /opt/conda/lib/python3.7/site-pack
ages (from tensorflow==2.1) (0.2.0)
Requirement already satisfied: protobuf>=3.8.0 in /opt/conda/lib/python3.7/site-packages
(from tensorflow==2.1) (3.16.0)
Requirement already satisfied: h5py in /opt/conda/lib/python3.7/site-packages (from kera
s-applications>=1.0.8->tensorflow==2.1) (2.10.0)
Requirement already satisfied: werkzeug>=0.11.15 in /opt/conda/lib/python3.7/site-packag
es (from tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (2.0.0)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /opt/conda/lib/python
3.7/site-packages (from tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (0.4.4)
Requirement already satisfied: requests<3,>=2.21.0 in /opt/conda/lib/python3.7/site-pack
ages (from tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (2.25.1)
Requirement already satisfied: setuptools>=41.0.0 in /opt/conda/lib/python3.7/site-packa
ges (from tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (49.6.0.post20210108)
Requirement already satisfied: markdown>=2.6.8 in /opt/conda/lib/python3.7/site-packages
(from tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (3.3.4)
Requirement already satisfied: google-auth<2,>=1.6.3 in /opt/conda/lib/python3.7/site-pa
ckages (from tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (1.30.0)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /opt/conda/lib/python3.7/site-p
ackages (from google-auth<2,>=1.6.3->tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (4.2.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /opt/conda/lib/python3.7/site-pa
ckages (from google-auth<2,>=1.6.3->tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (0.2.7)
Requirement already satisfied: rsa<5,>=3.1.4 in /opt/conda/lib/python3.7/site-packages
(from google-auth<2,>=1.6.3->tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (4.7.2)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /opt/conda/lib/python3.7/site
-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.2.0,>=2.1.0->tensorflow=
=2.1) (1.3.0)
Requirement already satisfied: importlib-metadata in /opt/conda/lib/python3.7/site-packa
ges (from markdown>=2.6.8->tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (4.0.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /opt/conda/lib/python3.7/site-pac
kages (from pyasn1-modules>=0.2.1->google-auth<2,>=1.6.3->tensorboard<2.2.0,>=2.1.0->ten
```

```
sorflow==2.1) (0.4.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/lib/python3.7/site-pa
ckages (from requests<3,>=2.21.0->tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (1.26.4)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/lib/python3.7/site-packages (f
rom requests<3,>=2.21.0->tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.7/site-packa
ges (from requests<3,>=2.21.0->tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (2020.12.5)
Requirement already satisfied: chardet<5,>=3.0.2 in /opt/conda/lib/python3.7/site-packag
es (from requests<3,>=2.21.0->tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (4.0.0)
Requirement already satisfied: oauthlib>=3.0.0 in /opt/conda/lib/python3.7/site-packages
(from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.2.0,>=2.
1.0->tensorflow==2.1) (3.0.1)
Requirement already satisfied: typing-extensions>=3.6.4 in /opt/conda/lib/python3.7/site
-packages (from importlib-metadata->markdown>=2.6.8->tensorboard<2.2.0,>=2.1.0->tensorfl
ow==2.1) (3.7.4.3)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-packages (from
importlib-metadata->markdown>=2.6.8->tensorboard<2.2.0,>=2.1.0->tensorflow==2.1) (3.4.1)
Building wheels for collected packages: gast
  Building wheel for gast (setup.py) ... done
  Created wheel for gast: filename=gast-0.2.2-py3-none-any.whl size=7538 sha256=800a98f7
16c83ea51233ceb23a0c166c33d7364172a07c4c33889303373f2108
  Stored in directory: /home/jupyter/.cache/pip/wheels/21/7f/02/420f32a803f7d0967b48dd82
3da3f558c5166991bfd204eef3
Successfully built gast
Installing collected packages: tensorflow-estimator, tensorboard, scipy, keras-applicati
ons, gast, astor, tensorflow
  Attempting uninstall: tensorflow-estimator
    Found existing installation: tensorflow-estimator 2.4.0
    Uninstalling tensorflow-estimator-2.4.0:
      Successfully uninstalled tensorflow-estimator-2.4.0
  Attempting uninstall: tensorboard
    Found existing installation: tensorboard 2.4.0
    Uninstalling tensorboard-2.4.0:
      Successfully uninstalled tensorboard-2.4.0
  Attempting uninstall: scipy
    Found existing installation: scipy 1.6.3
    Uninstalling scipy-1.6.3:
      Successfully uninstalled scipy-1.6.3
  Attempting uninstall: gast
    Found existing installation: gast 0.3.3
    Uninstalling gast-0.3.3:
      Successfully uninstalled gast-0.3.3
  Attempting uninstall: tensorflow
    Found existing installation: tensorflow 2.4.1
    Uninstalling tensorflow-2.4.1:
      Successfully uninstalled tensorflow-2.4.1
ERROR: pip's dependency resolver does not currently take into account all the packages t
hat are installed. This behaviour is the source of the following dependency conflicts.
tfx 0.28.0 requires attrs<21,>=19.3.0, but you have attrs 21.2.0 which is incompatible.
tfx 0.28.0 requires docker<5,>=4.1, but you have docker 5.0.0 which is incompatible.
tfx 0.28.0 requires google-api-python-client<2,>=1.7.8, but you have google-api-python-c
lient 2.3.0 which is incompatible.
tfx 0.28.0 requires kubernetes<12,>=10.0.1, but you have kubernetes 12.0.1 which is inco
mpatible.
tfx 0.28.0 requires pyarrow<3,>=1, but you have pyarrow 4.0.0 which is incompatible.
tfx 0.28.0 requires tensorflow!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.*,<3,>=1.15.2, but you have
 tensorflow 2.1.0 which is incompatible.
tfx-bsl 0.28.1 requires google-api-python-client<2,>=1.7.11, but you have google-api-pyt
hon-client 2.3.0 which is incompatible.
tfx-bsl 0.28.1 requires pyarrow<3,>=1, but you have pyarrow 4.0.0 which is incompatible.
tfx-bsl 0.28.1 requires tensorflow!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.*,<3,>=1.15.2, but you h
ave tensorflow 2.1.0 which is incompatible.
tensorflow-transform 0.28.0 requires pyarrow<3,>=1, but you have pyarrow 4.0.0 which is
 incompatible.
tensorflow-transform 0.28.0 requires tensorflow!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.*,<2.5,>=1.
```

```
15.2, but you have tensorflow 2.1.0 which is incompatible.
tensorflow-serving-api 2.4.0 requires tensorflow<3,>=2.4.0, but you have tensorflow 2.1.
0 which is incompatible.
tensorflow-probability 0.11.0 requires cloudpickle==1.3, but you have cloudpickle 1.6.0
 which is incompatible.
tensorflow-probability 0.11.0 requires gast>=0.3.2, but you have gast 0.2.2 which is inc
ompatible.
tensorflow-model-analysis 0.28.0 requires pyarrow<3,>=1, but you have pyarrow 4.0.0 whic
h is incompatible.
tensorflow-model-analysis 0.28.0 requires tensorflow!=2.0.*,!=2.1.*,!=2.2.*!=2.3.*,<3,>=
1.15.2, but you have tensorflow 2.1.0 which is incompatible.
tensorflow-io 0.17.0 requires tensorflow<2.5.0,>=2.4.0, but you have tensorflow 2.1.0 wh
ich is incompatible.
tensorflow-data-validation 0.28.0 requires joblib<0.15,>=0.12, but you have joblib 1.0.1
which is incompatible.
tensorflow-data-validation 0.28.0 requires pyarrow<3,>=1, but you have pyarrow 4.0.0 whi
ch is incompatible.
tensorflow-data-validation 0.28.0 requires tensorflow!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.*,<3,
>=1.15.2, but you have tensorflow 2.1.0 which is incompatible.
tensorflow-cloud 0.1.13 requires tensorboard>=2.3.0, but you have tensorboard 2.1.1 whic
h is incompatible.
phik 0.11.2 requires scipy>=1.5.2, but you have scipy 1.4.1 which is incompatible.
keras 2.4.0 requires tensorflow>=2.2.0, but you have tensorflow 2.1.0 which is incompati
ble.
fairness-indicators 0.28.0 requires tensorflow!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.*,<3,>=1.15.
2, but you have tensorflow 2.1.0 which is incompatible.
Successfully installed astor-0.8.1 gast-0.2.2 keras-applications-1.0.8 scipy-1.4.1 tenso
rboard-2.1.1 tensorflow-2.1.0 tensorflow-estimator-2.1.0
```

Note: Please ignore any incompatibility warnings and errors and re-run the cell to view the installed tensorflow version. `tensorflow==2.1.0` that is the installed version of tensorflow.

In [4]:

```
# You can use any Python source file as a module by executing an import statement in so
# The import statement combines two operations; it searches for the named module, then
# results of that search to a name in the local scope.
import os
import tensorflow.keras

# Use matplotlib for visualizing the model
import matplotlib.pyplot as plt
# Import Pandas data processing libraries
import pandas as pd
import tensorflow as tf

from tensorflow import feature_column as fc
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from keras.utils import plot_model

print("TensorFlow version: ",tf.version.VERSION)
```

```
TensorFlow version:  2.1.0
Using TensorFlow backend.
```

Many of the Google Machine Learning Courses Programming Exercises use the California Housing Dataset, which contains data drawn from the 1990 U.S. Census. Our lab dataset has been pre-processed so that there are no missing values.

First, let's download the raw .csv data by copying the data from a cloud storage bucket.

In [5]:

```
if not os.path.isdir("../data"):
    os.makedirs("../data")
```

In [6]:

```
# Download the raw .csv data by copying the data from a cloud storage bucket.
!gsutil cp gs://cloud-training-demos/feat_eng/housing/housing_pre-proc.csv ../data
```

```
Copying gs://cloud-training-demos/feat_eng/housing/housing_pre-proc.csv...
/ [1 files][  1.4 MiB/  1.4 MiB]
Operation completed over 1 objects/1.4 MiB.
```

In [8]:

```
# `ls` is a Linux shell command that lists directory contents
# `l` flag list all the files with permissions and details
!ls -l ../data/
```

```
total 1404
-rw-r--r-- 1 jupyter jupyter 1435069 May 22 16:52 housing_pre-proc.csv
```

Now, let's read in the dataset just copied from the cloud storage bucket and create a Pandas dataframe.

In [9]:

```
# `head()` function is used to get the first n rows of dataframe
housing_df = pd.read_csv('../data/housing_pre-proc.csv', error_bad_lines=False)
housing_df.head()
```

Out[9]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	med
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	

We can use .describe() to see some summary statistics for the numeric fields in our dataframe. Note, for example, the count row and corresponding columns. The count shows 20433.000000 for all feature columns. Thus, there are no missing values.

In [10]:

```
# `describe()` is use to get the statistical summary of the DataFrame
housing_df.describe()
```

Out[10]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	
count	20433.000000	20433.000000	20433.000000	20433.000000	20433.000000	20433.000000	2(
mean	-119.570689	35.633221	28.633094	2636.504233	537.870553	1424.946949	
std	2.003578	2.136348	12.591805	2185.269567	421.385070	1133.208490	
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	
25%	-121.800000	33.930000	18.000000	1450.000000	296.000000	787.000000	

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000
75%	-118.010000	37.720000	37.000000	3143.000000	647.000000	1722.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000

Split the dataset for ML

The dataset we loaded was a single CSV file. We will split this into train, validation, and test sets.

In [11]:
```
# Let's split the dataset into train, validation, and test sets
train, test = train_test_split(housing_df, test_size=0.2)
train, val = train_test_split(train, test_size=0.2)

print(len(train), 'train examples')
print(len(val), 'validation examples')
print(len(test), 'test examples')
```

```
13076 train examples
3270 validation examples
4087 test examples
```

Now, we need to output the split files. We will specifically need the test.csv later for testing. You should see the files appear in the home directory.

In [12]:
```
train.to_csv('../data/housing-train.csv', encoding='utf-8', index=False)
```

In [13]:
```
val.to_csv('../data/housing-val.csv', encoding='utf-8', index=False)
```

In [14]:
```
test.to_csv('../data/housing-test.csv', encoding='utf-8', index=False)
```

In [15]:
```
!head ../data/housing*.csv
```

```
==> ../data/housing-test.csv <==
longitude,latitude,housing_median_age,total_rooms,total_bedrooms,population,households,m
edian_income,median_house_value,ocean_proximity
-122.07,37.89,38.0,757.0,124.0,319.0,123.0,5.6558,263300.0,NEAR BAY
-117.93,33.67,27.0,3512.0,472.0,1391.0,481.0,8.1001,336500.0,<1H OCEAN
-119.34,36.22,38.0,2708.0,460.0,1260.0,455.0,3.0905,78200.0,INLAND
-119.67,36.8,9.0,3712.0,508.0,1632.0,474.0,6.011,163100.0,INLAND
-122.52,37.91,30.0,4174.0,739.0,1818.0,705.0,5.5951,402900.0,NEAR BAY
-118.2,33.82,43.0,1758.0,347.0,954.0,312.0,5.2606,198900.0,NEAR OCEAN
-117.16,34.06,33.0,2101.0,468.0,1997.0,412.0,2.8125,117200.0,INLAND
-118.01,33.95,36.0,1579.0,290.0,816.0,276.0,4.4318,181100.0,<1H OCEAN
-119.01,34.23,11.0,5785.0,1035.0,2760.0,985.0,4.6930000000000005,232200.0,<1H OCEAN

==> ../data/housing-train.csv <==
longitude,latitude,housing_median_age,total_rooms,total_bedrooms,population,households,m
edian_income,median_house_value,ocean_proximity
-122.26,37.8,20.0,2373.0,779.0,1659.0,676.0,1.6929,115000.0,NEAR BAY
-118.37,34.06,52.0,843.0,160.0,333.0,151.0,4.5192,446000.0,<1H OCEAN
```

```
-118.53,34.2,33.0,3270.0,818.0,2118.0,763.0,3.225,205300.0,<1H OCEAN
-117.22,34.44,5.0,4787.0,910.0,1944.0,806.0,2.6576,98500.0,INLAND
-122.91,39.05,27.0,789.0,208.0,295.0,108.0,3.7667,95000.0,INLAND
-118.25,33.98,40.0,1867.0,633.0,2223.0,609.0,1.7207,105100.0,<1H OCEAN
-119.77,36.84,15.0,1924.0,262.0,848.0,277.0,5.3886,125300.0,INLAND
-117.12,32.59,28.0,2793.0,706.0,1825.0,676.0,2.6724,144500.0,NEAR OCEAN
-121.76,36.92,36.0,2096.0,409.0,1454.0,394.0,3.2216,238300.0,<1H OCEAN

==> ../data/housing-val.csv <==
longitude,latitude,housing_median_age,total_rooms,total_bedrooms,population,households,m
edian_income,median_house_value,ocean_proximity
-117.8,33.64,8.0,4447.0,713.0,1680.0,705.0,8.8693,450400.0,<1H OCEAN
-119.79,34.4,20.0,3104.0,415.0,1061.0,380.0,9.6885,500001.0,NEAR OCEAN
-120.99,37.67,28.0,1768.0,423.0,1066.0,392.0,1.8315,90500.0,INLAND
-118.29,33.73,21.0,2492.0,711.0,1699.0,672.0,2.1382,242300.0,NEAR OCEAN
-122.26,37.81,34.0,5871.0,1914.0,2689.0,1789.0,2.8406,335700.0,NEAR BAY
-117.23,33.68,10.0,3659.0,650.0,1476.0,515.0,3.8869,125900.0,<1H OCEAN
-122.14,39.97,27.0,1079.0,222.0,625.0,197.0,3.1319,62700.0,INLAND
-119.58,36.11,21.0,2004.0,385.0,1397.0,398.0,2.2169,61600.0,INLAND
-116.97,32.76,33.0,3071.0,466.0,1348.0,513.0,6.1768,228900.0,<1H OCEAN

==> ../data/housing_pre-proc.csv <==
longitude,latitude,housing_median_age,total_rooms,total_bedrooms,population,households,m
edian_income,median_house_value,ocean_proximity
-122.23,37.88,41.0,880.0,129.0,322.0,126.0,8.3252,452600.0,NEAR BAY
-122.22,37.86,21.0,7099.0,1106.0,2401.0,1138.0,8.3014,358500.0,NEAR BAY
-122.24,37.85,52.0,1467.0,190.0,496.0,177.0,7.2574,352100.0,NEAR BAY
-122.25,37.85,52.0,1274.0,235.0,558.0,219.0,5.6431,341300.0,NEAR BAY
-122.25,37.85,52.0,1627.0,280.0,565.0,259.0,3.8462,342200.0,NEAR BAY
-122.25,37.85,52.0,919.0,213.0,413.0,193.0,4.0368,269700.0,NEAR BAY
-122.25,37.84,52.0,2535.0,489.0,1094.0,514.0,3.6591,299200.0,NEAR BAY
-122.25,37.84,52.0,3104.0,687.0,1157.0,647.0,3.12,241400.0,NEAR BAY
-122.26,37.84,42.0,2555.0,665.0,1206.0,595.0,2.0804,226700.0,NEAR BAY
```

Lab Task 1: Create an input pipeline using tf.data

Next, we will wrap the dataframes with tf.data. This will enable us to use feature columns as a bridge to map from the columns in the Pandas dataframe to features used to train the model.

Here, we create an input pipeline using tf.data. This function is missing two lines. Correct and run the cell.

In [16]:
```
# A utility method to create a tf.data dataset from a Pandas Dataframe
# TODO 1a
def df_to_dataset(dataframe, shuffle=True, batch_size=32):
    dataframe = dataframe.copy()
    labels = dataframe.pop('median_house_value')
    ds = tf.data.Dataset.from_tensor_slices((dict(dataframe), labels))
    if shuffle:
        ds = ds.shuffle(buffer_size=len(dataframe))
    ds = ds.batch(batch_size)
    return ds
```

In [ ]:
```

```

Next we initialize the training and validation datasets.

In [17]:
```
batch_size = 32
```

```
train_ds = df_to_dataset(train)
val_ds = df_to_dataset(val, shuffle=False, batch_size=batch_size)
```

In [18]:
```
train_ds
```

Out[18]:
```
<BatchDataset shapes: ({longitude: (None,), latitude: (None,), housing_median_age: (Non
e,), total_rooms: (None,), total_bedrooms: (None,), population: (None,), households: (No
ne,), median_income: (None,), ocean_proximity: (None,)}, (None,)), types: ({longitude: t
f.float32, latitude: tf.float32, housing_median_age: tf.float32, total_rooms: tf.float3
2, total_bedrooms: tf.float32, population: tf.float32, households: tf.float32, median_in
come: tf.float32, ocean_proximity: tf.string}, tf.float32)>
```

Now that we have created the input pipeline, let's call it to see the format of the data it returns. We have used a small batch size to keep the output readable.

In [ ]:
```
# TODO 1b
for feature_batch, label_batch in train_ds.take(1):
    print('Every feature:', list(feature_batch.keys()))
    print('A batch of households:', feature_batch['households'])
    print('A batch of ocean_proximity:', feature_batch['ocean_proximity'])
    print('A batch of targets:', label_batch)
```

We can see that the dataset returns a dictionary of column names (from the dataframe) that map to column values from rows in the dataframe.

Numeric columns

The output of a feature column becomes the input to the model. A numeric is the simplest type of column. It is used to represent real valued features. When using this column, your model will receive the column value from the dataframe unchanged.

In the California housing prices dataset, most columns from the dataframe are numeric. Let's create a variable called **numeric_cols** to hold only the numerical feature columns.

In [ ]:
```
# Let's create a variable called `numeric_cols` to hold only the numerical feature colu
# TODO 1c
numeric_cols = ['longitude', 'latitude', 'housing_median_age', 'total_rooms',
                'total_bedrooms', 'population', 'households', 'median_income']
```

Scaler function

It is very important for numerical variables to get scaled before they are "fed" into the neural network. Here we use min-max scaling. Here we are creating a function named 'get_scal' which takes a list of numerical features and returns a 'minmax' function, which will be used in tf.feature_column.numeric_column() as normalizer_fn in parameters. 'Minmax' function itself takes a 'numerical' number from a particular feature and return scaled value of that number.

Next, we scale the numerical feature columns that we assigned to the variable "numeric cols".

In [ ]:
```
# 'get_scal' function takes a list of numerical features and returns a 'minmax' functio
# 'Minmax' function itself takes a 'numerical' number from a particular feature and ret
# Scalar def get_scal(feature):
```

```
# TODO 1d
def get_scal(feature):
    def minmax(x):
        mini = train[feature].min()
        maxi = train[feature].max()
        return (x - mini)/(maxi-mini)
        return(minmax)
```

In [ ]:

```
# TODO 1e
feature_columns = []
for header in numeric_cols:
    scal_input_fn = get_scal(header)
    feature_columns.append(fc.numeric_column(header,
                                             normalizer_fn=scal_input_fn))
```

Next, we should validate the total number of feature columns. Compare this number to the number of numeric features you input earlier.

In [ ]:

```
print('Total number of feature coLumns: ', len(feature_columns))
```

Using the Keras Sequential Model

Next, we will run this cell to compile and fit the Keras Sequential model.

In [ ]:

```
# Model create
# `tf.keras.layers.DenseFeatures()` is a layer that produces a dense Tensor based on gi
feature_layer = tf.keras.layers.DenseFeatures(feature_columns, dtype='float64')

# `tf.keras.Sequential()` groups a linear stack of layers into a tf.keras.Model.
model = tf.keras.Sequential([
  feature_layer,
  layers.Dense(12, input_dim=8, activation='relu'),
  layers.Dense(8, activation='relu'),
  layers.Dense(1, activation='linear',  name='median_house_value')
])

# Model compile
model.compile(optimizer='adam',
              loss='mse',
              metrics=['mse'])

# Model Fit
history = model.fit(train_ds,
                    validation_data=val_ds,
                    epochs=32)
```

Next we show loss as Mean Square Error (MSE). Remember that MSE is the most commonly used regression loss function. MSE is the sum of squared distances between our target variable (e.g. housing median age) and predicted values.

In [ ]:

```
# Let's show loss as Mean Square Error (MSE)
loss, mse = model.evaluate(train_ds)
print("Mean Squared Error", mse)
```

Visualize the model loss curve

Next, we will use matplotlib to draw the model's loss curves for training and validation. A line plot is also created showing the mean squared error loss over the training epochs for both the train (blue) and test (orange) sets.

In [ ]:
```
# Use matplotlib to draw the model's loss curves for training and validation
def plot_curves(history, metrics):
    nrows = 1
    ncols = 2
    fig = plt.figure(figsize=(10, 5))

    for idx, key in enumerate(metrics):
        ax = fig.add_subplot(nrows, ncols, idx+1)
        plt.plot(history.history[key])
        plt.plot(history.history['val_{}'.format(key)])
        plt.title('model {}'.format(key))
        plt.ylabel(key)
        plt.xlabel('epoch')
        plt.legend(['train', 'validation'], loc='upper left');
```

In [ ]:
```
plot_curves(history, ['loss', 'mse'])
```

Load test data

Next, we read in the test.csv file and validate that there are no null values.

Again, we can use .describe() to see some summary statistics for the numeric fields in our dataframe. The count shows 4087.000000 for all feature columns. Thus, there are no missing values.

In [ ]:
```
test_data = pd.read_csv('../data/housing-test.csv')
test_data.describe()
```

Now that we have created an input pipeline using tf.data and compiled a Keras Sequential Model, we now create the input function for the test data and to initialize the test_predict variable.

In [ ]:
```
# TODO 1f
def test_input_fn(features, batch_size=256):
    """An input function for prediction."""
    # Convert the inputs to a Dataset without labels.
    return tf.data.Dataset.from_tensor_slices(dict(features)).batch(batch_size)
```

In [ ]:
```
test_predict = test_input_fn(dict(test_data))
```

Prediction: Linear Regression

Before we begin to feature engineer our feature columns, we should predict the median house value. By predicting the median house value now, we can then compare it with the median house value after feature engineering.

To predict with Keras, you simply call model.predict() and pass in the housing features you want to predict the median_house_value for. Note: We are predicting the model locally.

In [ ]:
```
# Use the model to do prediction with `model.predict()`
predicted_median_house_value = model.predict(test_predict)
```

Next, we run two predictions in separate cells - one where ocean_proximity=INLAND and one where ocean_proximity= NEAR OCEAN.

In [ ]:
```
# Ocean_proximity is INLAND
model.predict({
    'longitude': tf.convert_to_tensor([-121.86]),
    'latitude': tf.convert_to_tensor([39.78]),
    'housing_median_age': tf.convert_to_tensor([12.0]),
    'total_rooms': tf.convert_to_tensor([7653.0]),
    'total_bedrooms': tf.convert_to_tensor([1578.0]),
    'population': tf.convert_to_tensor([3628.0]),
    'households': tf.convert_to_tensor([1494.0]),
    'median_income': tf.convert_to_tensor([3.0905]),
    'ocean_proximity': tf.convert_to_tensor(['INLAND'])
}, steps=1)
```

In [ ]:
```
# Ocean_proximity is NEAR OCEAN
model.predict({
    'longitude': tf.convert_to_tensor([-122.43]),
    'latitude': tf.convert_to_tensor([37.63]),
    'housing_median_age': tf.convert_to_tensor([34.0]),
    'total_rooms': tf.convert_to_tensor([4135.0]),
    'total_bedrooms': tf.convert_to_tensor([687.0]),
    'population': tf.convert_to_tensor([2154.0]),
    'households': tf.convert_to_tensor([742.0]),
    'median_income': tf.convert_to_tensor([4.9732]),
    'ocean_proximity': tf.convert_to_tensor(['NEAR OCEAN'])
}, steps=1)
```

The arrays returns a predicted value. What do these numbers mean? Let's compare this value to the test set.

Go to the test.csv you read in a few cells up. Locate the first line and find the median_house_value - which should be 249,000 dollars near the ocean. What value did your model predicted for the median_house_value? Was it a solid model performance? Let's see if we can improve this a bit with feature engineering!

Lab Task 2: Engineer features to create categorical and numerical features

Now we create a cell that indicates which features will be used in the model.
Note: Be sure to bucketize 'housing_median_age' and ensure that 'ocean_proximity' is one-hot encoded. And, don't forget your numeric values!

In [ ]:
```
# TODO 2a
```

```
numeric_cols = ['longitude', 'latitude', 'housing_median_age', 'total_rooms',
                'total_bedrooms', 'population', 'households', 'median_income']

bucketized_cols = ['housing_median_age']

# indicator columns,Categorical features
categorical_cols = ['ocean_proximity']
```

Next, we scale the numerical, bucktized, and categorical feature columns that we assigned to the variables in the preceding cell.

In [ ]:
```
# Scalar def get_scal(feature):
def get_scal(feature):
    def minmax(x):
        mini = train[feature].min()
        maxi = train[feature].max()
        return (x - mini)/(maxi-mini)
        return(minmax)
```

In [ ]:
```
# All numerical features - scaling
feature_columns = []
for header in numeric_cols:
    scal_input_fn = get_scal(header)
    feature_columns.append(fc.numeric_column(header,
                                             normalizer_fn=scal_input_fn))
```

Categorical Feature

In this dataset, 'ocean_proximity' is represented as a string. We cannot feed strings directly to a model. Instead, we must first map them to numeric values. The categorical vocabulary columns provide a way to represent strings as a one-hot vector.

Next, we create a categorical feature using 'ocean_proximity'.

In [ ]:
```
# TODO 2b
for feature_name in categorical_cols:
    vocabulary = housing_df[feature_name].unique()
    categorical_c = fc.categorical_column_with_vocabulary_list(feature_name, vocabulary
    one_hot = fc.indicator_column(categorical_c)
    feature_columns.append(one_hot)
```

Bucketized Feature

Often, you don't want to feed a number directly into the model, but instead split its value into different categories based on numerical ranges. Consider our raw data that represents a homes' age. Instead of representing the house age as a numeric column, we could split the home age into several buckets using a bucketized column. Notice the one-hot values below describe which age range each row matches.

Next we create a bucketized column using 'housing_median_age'

In [ ]:
```
# TODO 2c
```

```
age = fc.numeric_column("housing_median_age")

# Bucketized cols
age_buckets = fc.bucketized_column(age, boundaries=[10, 20, 30, 40, 50, 60, 80, 100])
feature_columns.append(age_buckets)
```

Feature Cross

Combining features into a single feature, better known as feature crosses, enables a model to learn separate weights for each combination of features.

Next, we create a feature cross of 'housing_median_age' and 'ocean_proximity'.

In [ ]:
```
# TODO 2d
vocabulary = housing_df['ocean_proximity'].unique()
ocean_proximity = fc.categorical_column_with_vocabulary_list('ocean_proximity',
                                                             vocabulary)

crossed_feature = fc.crossed_column([age_buckets, ocean_proximity],
                                    hash_bucket_size=1000)
crossed_feature = fc.indicator_column(crossed_feature)
feature_columns.append(crossed_feature)
```

Next, we should validate the total number of feature columns. Compare this number to the number of numeric features you input earlier.

In [ ]:
```
print('Total number of feature columns: ', len(feature_columns))
```

Next, we will run this cell to compile and fit the Keras Sequential model. This is the same model we ran earlier.

In [ ]:
```
# Model create
# `tf.keras.layers.DenseFeatures()` is a layer that produces a dense Tensor based on gi
feature_layer = tf.keras.layers.DenseFeatures(feature_columns,
                                              dtype='float64')

# `tf.keras.Sequential()` groups a linear stack of layers into a tf.keras.Model.
model = tf.keras.Sequential([
  feature_layer,
  layers.Dense(12, input_dim=8, activation='relu'),
  layers.Dense(8, activation='relu'),
  layers.Dense(1, activation='linear',  name='median_house_value')
])

# Model compile
model.compile(optimizer='adam',
              loss='mse',
              metrics=['mse'])

# Model Fit
history = model.fit(train_ds,
                    validation_data=val_ds,
                    epochs=32)
```

Next, we show loss and mean squared error then plot the model.

In [ ]:
```
loss, mse = model.evaluate(train_ds)
print("Mean Squared Error", mse)
```

In [ ]:
```
plot_curves(history, ['loss', 'mse'])
```

Next we create a prediction model. Note: You may use the same values from the previous prediciton.

In [ ]:
```
# TODO 2e
# Median_house_value is $249,000, prediction is $234,000 NEAR OCEAN
model.predict({
    'longitude': tf.convert_to_tensor([-122.43]),
    'latitude': tf.convert_to_tensor([37.63]),
    'housing_median_age': tf.convert_to_tensor([34.0]),
    'total_rooms': tf.convert_to_tensor([4135.0]),
    'total_bedrooms': tf.convert_to_tensor([687.0]),
    'population': tf.convert_to_tensor([2154.0]),
    'households': tf.convert_to_tensor([742.0]),
    'median_income': tf.convert_to_tensor([4.9732]),
    'ocean_proximity': tf.convert_to_tensor(['NEAR OCEAN'])
}, steps=1)
```

Analysis

The array returns a predicted value. Compare this value to the test set you ran earlier. Your predicted value may be a bit better.

Now that you have your "feature engineering template" setup, you can experiment by creating additional features. For example, you can create derived features, such as households per population, and see how they impact the model. You can also experiment with replacing the features you used to create the feature cross.

Copyright 2020 Google Inc. 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 http://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.