AI学习笔记--Tensorflow--TF Basic regression

在 *回归 (regression)* 问题中，我们的目的是预测出如价格或概率这样连续值的输出。相对于*分类 (classification)* 问题，*分类(classification)* 的目的是从一系列的分类出选择出一个分类 （如，给出一张包含苹果或橘子的图片，识别出图片中是哪种水果）。

本 notebook 使用经典的 Auto MPG 数据集，构建了一个用来预测70年代末到80年代初汽车燃油效率的模型。为了做到这一点，我们将为该模型提供许多那个时期的汽车描述。这个描述包含：气缸数，排量，马力以及重量。

在运行本例之前，需要先安装绘制矩阵型图的模块：

```
pip install -q seaborn
```

并且导入所需要的模块：

```
from __future__ import absolute_import, division, print_function, unicode_literals

import pathlib

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
```

同时编辑辅助性函数，例如：打印训练进度的方法，数据标准化方法，绘制结果的方法：

```
# 通过为每个完成的时期打印一个点来显示训练进度
class PrintDot(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    if epoch % 100 == 0: print('')
    print('.', end='')

# 数据标准化
def norm(x):
  return (x - train_stats['mean']) / train_stats['std']

# 结果作图方法
def plot_history(history):
  hist = pd.DataFrame(history.history)
  hist['epoch'] = history.epoch

  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Abs Error [MPG]')
  plt.plot(hist['epoch'], hist['mean_absolute_error'],
           label='Train Error')
  plt.plot(hist['epoch'], hist['val_mean_absolute_error'],
           label = 'Val Error')
  plt.ylim([0,5])
  plt.legend()

  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Square Error [$MPG^2$]')
  plt.plot(hist['epoch'], hist['mean_squared_error'],
           label='Train Error')
  plt.plot(hist['epoch'], hist['val_mean_squared_error'],
           label = 'Val Error')
  plt.ylim([0,20])
  plt.legend()
  plt.show()
```

```
# 建立模型方法
def build_model():
  model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=[len(train_dataset.keys())]),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)
  ])

  optimizer = tf.keras.optimizers.RMSprop(0.001)

model.compile(loss='mse',
                optimizer=optimizer,
                metrics=['mae', 'mse'])
  return model
```

然后，开始下载对应的训练数据集合，并且导入：

```
# 导入工具
layers = keras.layers

# 导入数据
dataset_path = keras.utils.get_file("auto-mpg.data",
"http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data")
dataset_path

column_names = ['MPG','Cylinders','Displacement','Horsepower','Weight',
                'Acceleration', 'Model Year', 'Origin']
raw_dataset = pd.read_csv(dataset_path, names=column_names,
                      na_values = "?", comment='\t',
                      sep=" ", skipinitialspace=True)

dataset = raw_dataset.copy()
dataset.tail()
# 过滤未知数据
dataset.isna().sum()
```

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Origin
393	27.0	4	140.0	86.0	2790.0	15.6	82	1
394	44.0	4	97.0	52.0	2130.0	24.6	82	2
395	32.0	4	135.0	84.0	2295.0	11.6	82	1
396	28.0	4	120.0	79.0	2625.0	18.6	82	1
397	31.0	4	119.0	82.0	2720.0	19.4	82	1

为了保证简单，我们需要删除一些未知数据，可以用这句代码过滤删除：

```
#为了保证这个初始示例的简单性，删除这些行。
dataset = dataset.dropna()
```

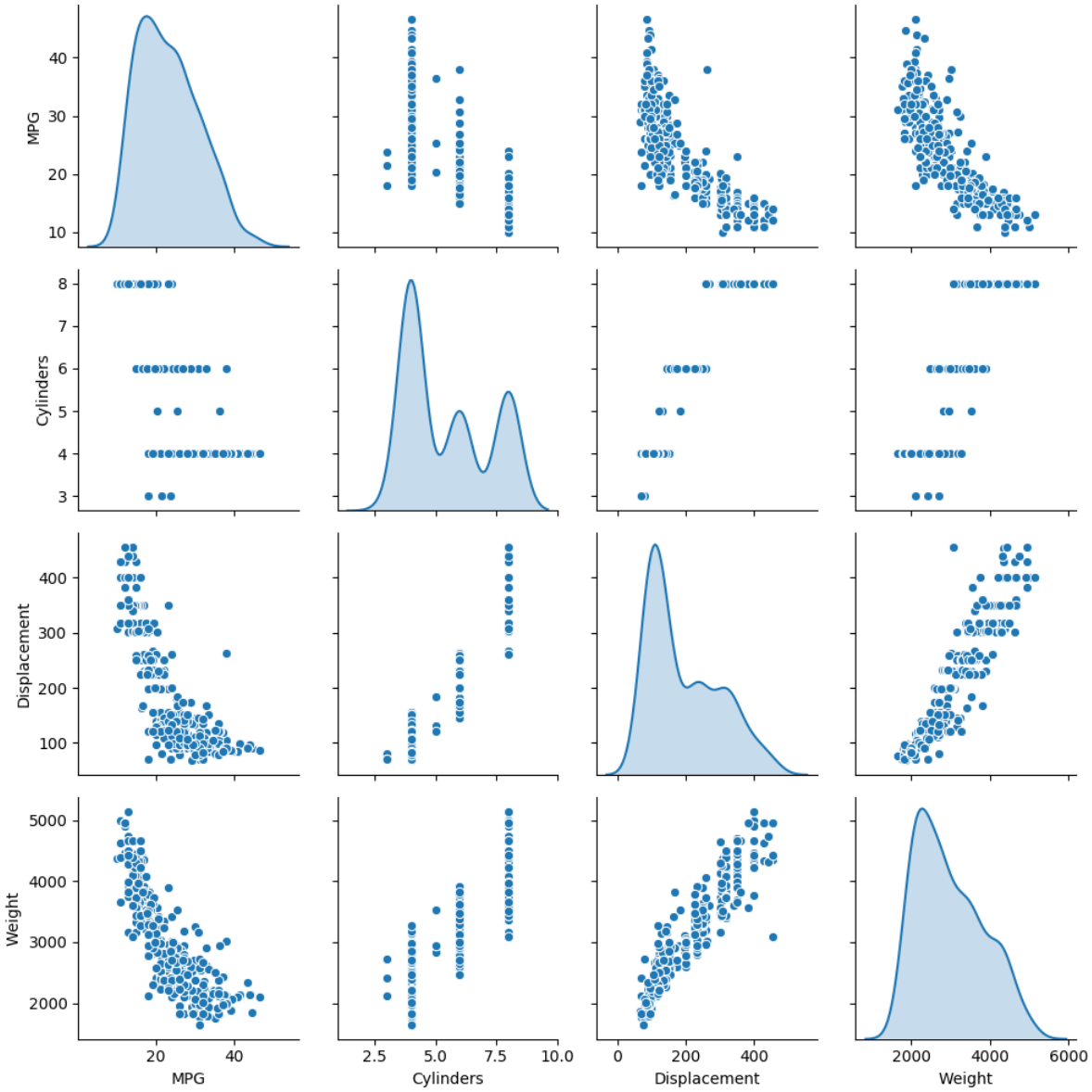
"Origin" 列实际上代表分类，而不仅仅是一个数字。所以把它转换为独热码 （one-hot）：

```
origin = dataset.pop('Origin')

dataset['USA'] = (origin == 1)*1.0
dataset['Europe'] = (origin == 2)*1.0
dataset['Japan'] = (origin == 3)*1.0
dataset.tail()
```

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	USA	Europe	Japan
393	27.0	4	140.0	86.0	2790.0	15.6	82	1.0	0.0	0.0
394	44.0	4	97.0	52.0	2130.0	24.6	82	0.0	1.0	0.0
395	32.0	4	135.0	84.0	2295.0	11.6	82	1.0	0.0	0.0
396	28.0	4	120.0	79.0	2625.0	18.6	82	1.0	0.0	0.0
397	31.0	4	119.0	82.0	2720.0	19.4	82	1.0	0.0	0.0

随后，对数据进行拆分，把原始数据的部分作为训练数据，并且用一部分作为测试数据。并且显示到图中：

```
train_dataset = dataset.sample(frac=0.8,random_state=0)
test_dataset = dataset.drop(train_dataset.index)
# 显示矩阵图
sns.pairplot(train_dataset[["MPG", "Cylinders", "Displacement", "Weight"]], diag_kind="kde")
```



也可以看一下总体的数据统计：

```
train_stats = train_dataset.describe()
train_stats.pop("MPG")
train_stats = train_stats.transpose()
train_stats
```

	count	mean	std	min	25%	50%	75%	max
Cylinders	314.0	5.477707	1.699788	3.0	4.00	4.0	8.00	8.0
Displacement	314.0	195.318471	104.331589	68.0	105.50	151.0	265.75	455.0
Horsepower	314.0	104.869427	38.096214	46.0	76.25	94.5	128.00	225.0
Weight	314.0	2990.251592	843.898596	1649.0	2256.50	2822.5	3608.00	5140.0

Acceleration	314.0	15.559236	2.789230	8.0	13.80	15.5	17.20	24.8
Model Year	314.0	75.898089	3.675642	70.0	73.00	76.0	79.00	82.0
USA	314.0	0.624204	0.485101	0.0	0.00	1.0	1.00	1.0
Europe	314.0	0.178344	0.383413	0.0	0.00	0.0	0.00	1.0
Japan	314.0	0.197452	0.398712	0.0	0.00	0.0	0.00	1.0

接着，从标签中分离特征，并且这个特征是用于训练模型进行预测的数值。

```
train_labels = train_dataset.pop('MPG')
test_labels = test_dataset.pop('MPG')
```

处理一下数据规范化，使用不同的尺度和范围对数据进行归一化处理。注意：尽管我们仅仅从训练集中有意生成这些统计数据，但是这些统计信息也会用于归一化的测试数据集。我们需要这样做，将测试数据集放入到与已经训练过的模型相同的分布中。

```
normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)
```

我们将会使用这个已经归一化的数据来训练模型。

警告: 用于归一化输入的数据统计（均值和标准差）需要反馈给模型从而应用于任何其他数据，以及我们之前所获得独热码。这些数据包含测试数据集以及生产环境中所使用的实时数据。

- 构建模型

让我们来构建我们自己的模型。这里，我们将会使用一个“顺序”模型，其中包含两个紧密相连的隐藏层，以及返回单个、连续值得输出层。模型的构建步骤包含于一个名叫 'build_model' 的函数中，稍后我们将会创建第二个模型。两个密集连接的隐藏层。

```
model = build_model()
model.summary()
```

现在试用下这个模型。从训练数据中批量获取‘10’条例子并对这些例子调用 model.predict

```
example_batch = normed_train_data[:10]
example_result = model.predict(example_batch)
example_result
```

预测的十条数据结果如下：

```
WARNING:tensorflow:Falling back from v2 loop because of error: Failed to find data adapter that
can handle input: <class 'pandas.core.frame.DataFrame'>, <class 'NoneType'>

array([[ 0.01557792],
       [-0.09677997],
       [ 0.09817091],
       [-0.17441332],
       [-0.2521149 ],
       [-0.12406033],
       [-0.23704767],
       [-0.1016776 ],
       [-0.1110839 ],
       [-0.23353352]], dtype=float32)
```

对模型进行1000个周期的训练，并在 history 对象中记录训练和验证的准确性。

```
EPOCHS = 1000

history = model.fit(
  normed_train_data, train_labels,
  epochs=EPOCHS, validation_split = 0.2, verbose=0,
  callbacks=[PrintDot()])
```

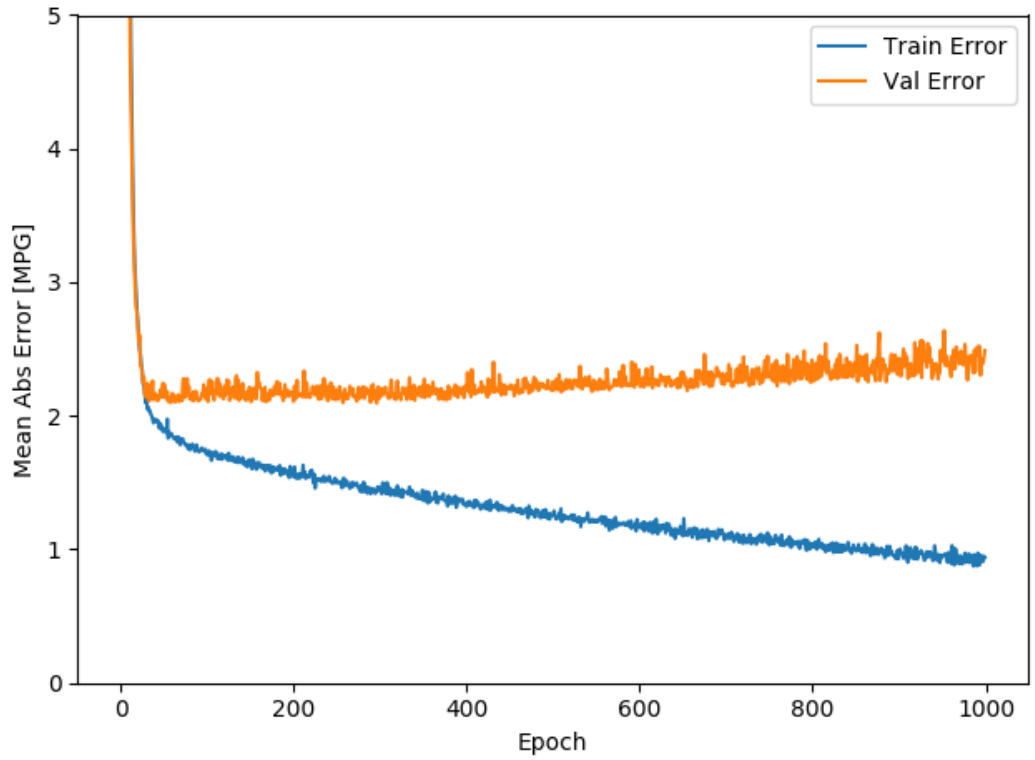
使用 history 对象中存储的统计信息可视化模型的训练进度。

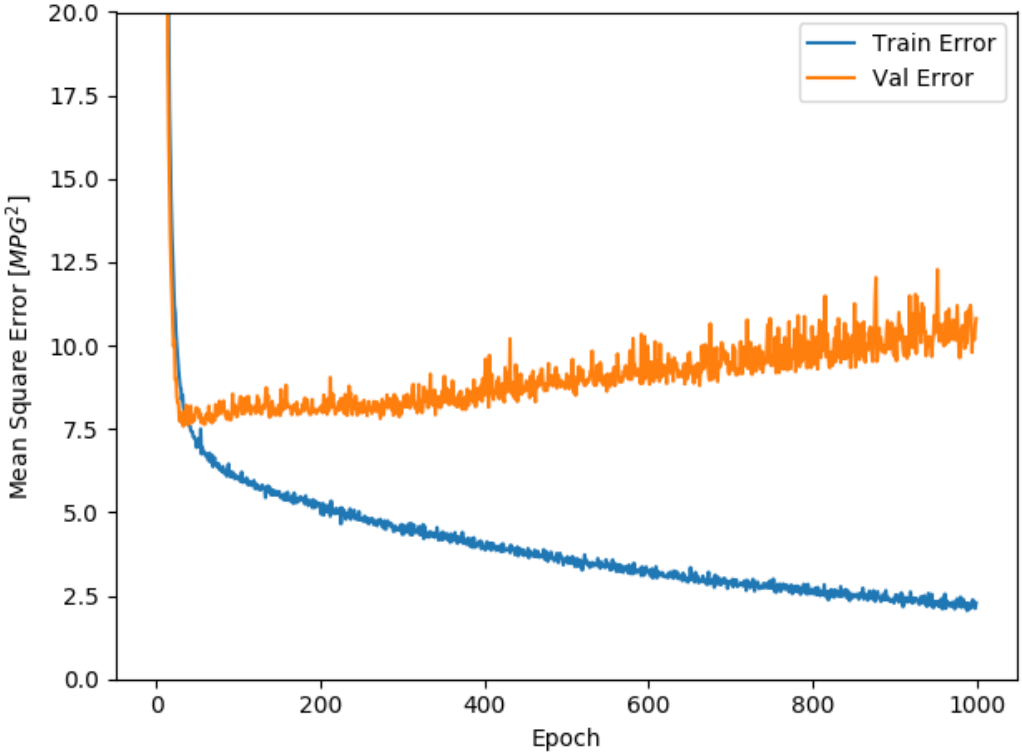
```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
```

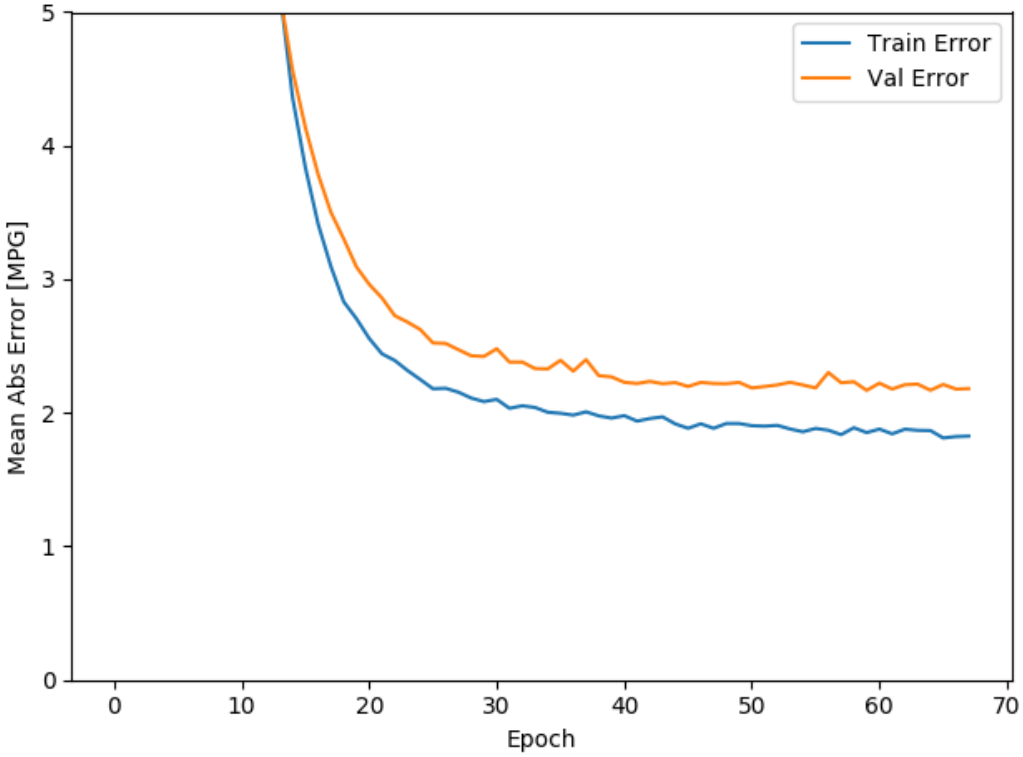
	loss	mae	mse	val_loss	val_mae	val_mse	epoch
995	2.377308	0.972564	2.377308	9.812141	2.277914	9.812140	995
996	2.089346	0.881897	2.089346	11.054218	2.500603	11.054217	996
997	2.168902	0.915549	2.168902	9.853918	2.314816	9.853918	997
998	2.340033	0.944049	2.340033	10.559951	2.421101	10.559950	998
999	2.254203	0.926744	2.254203	10.530814	2.451076	10.530814	999

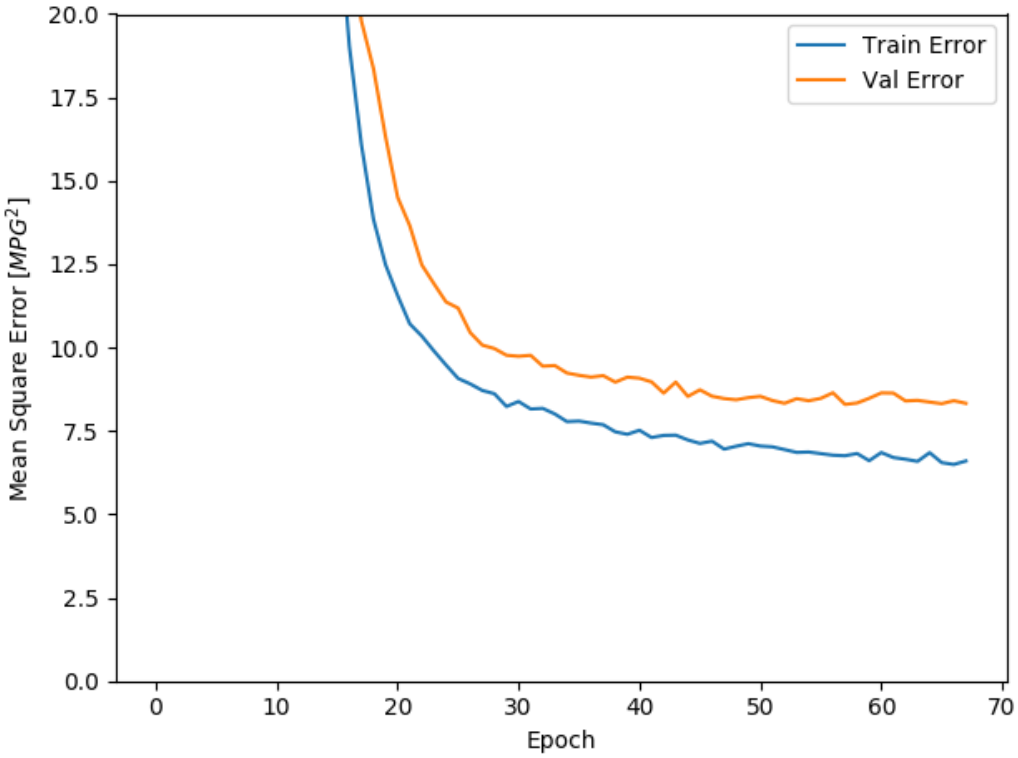
打印可视化数据：

```
plot_history(history)
```




Train Error
Val Error
Mean Square Error [MPG^2]
Epoch
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
200
400
600
800
1000


Train Error
Val Error
Mean Abs Error [MPG]
Epoch
5
4
3
2
1
0
0
10
20
30
40
50
60
70



如图所示，验证集中的平均的误差通常在 +/- 2 MPG左右。

让我们看看通过使用 测试集 来泛化模型的效果如何，我们在训练模型时没有使用测试集。这告诉我们，当我们在现实世界中使用这个模型时，我们可以期望它预测得有多好。

```
loss, mae, mse = model.evaluate(normed_test_data, test_labels, verbose=2)
print("\n Testing set Mean Abs Error: {:5.2f} Mean squared error :{:5.2f} loss :
{:5.2f}MPG".format(mae,mse,loss))
```

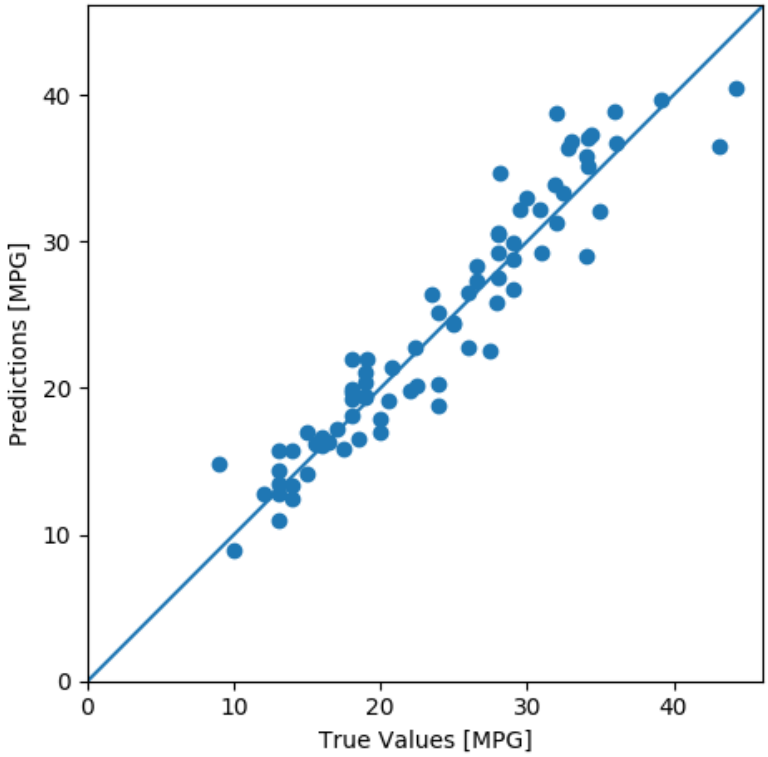
结果：

```
Testing set Mean Abs Error:  1.98 Mean squared error : 5.97 loss : 5.97MPG
```

- 预测

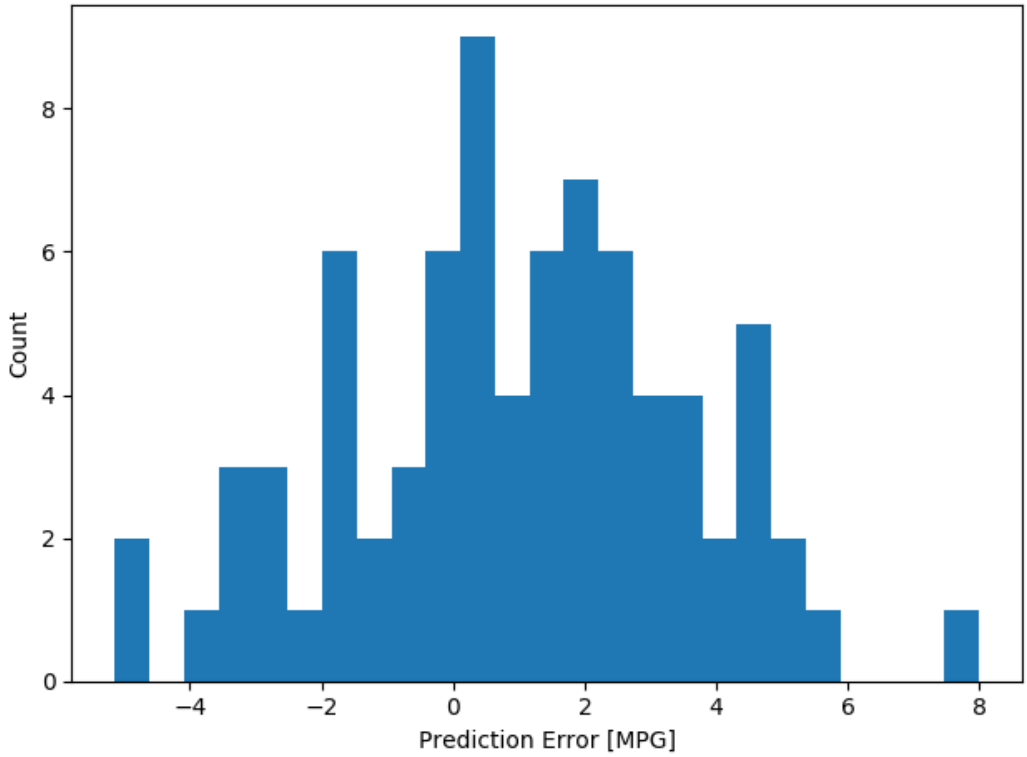
最后，使用测试集中的数据预测 MPG 值:

```
test_predictions = model.predict(normed_test_data).flatten()

plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values [MPG]')
plt.ylabel('Predictions [MPG]')
plt.axis('equal')
plt.axis('square')
plt.xlim([0,plt.xlim()[1]])
plt.ylim([0,plt.ylim()[1]])
plt.plot([-100, 100], [-100, 100])
plt.show()
```



上图是预测结果和实际的结果。还可以分析一下误差分布：

```
error = test_predictions - test_labels
plt.hist(error, bins = 25)
plt.xlabel("Prediction Error [MPG]")
plt.ylabel("Count")

plt.show()
```



可能由于样本数量不足导致误差较大。

- 结论

本笔记本 (notebook) 介绍了一些处理回归问题的技术。

- 均方误差（MSE）是用于回归问题的常见损失函数（分类问题中使用不同的损失函数）。
- 类似的，用于回归的评估指标与分类不同。 常见的回归指标是平均绝对误差（MAE）。
- 当数字输入数据特征的值存在不同范围时，每个特征应独立缩放到相同范围。
- 如果训练数据不多，一种方法是选择隐藏层较少的小网络，以避免过度拟合。
- 早期停止是一种防止过度拟合的有效技术。