AI学习笔记--Tensorflow--用 tf.data 加载 CSV 数据

这篇教程通过一个示例展示了怎样将 CSV 格式的数据加载进 tf.data.Dataset。

这篇教程使用的是泰坦尼克号乘客的数据。模型会根据乘客的年龄、性别、票务舱和是否独自旅行等特征来预测乘客生还的可能性。首先运行需要配置额外组件。按照下面链接的教程说明下载 TensorFlow-io 模块。

https://github.com/tensorflow/io

```
(venv) ➜  Tensorflow pip install tensorflow-io-nightly
```

安装成功后，在 running 的时候，不会出现很多 warniing。也避免程序在运行过程中出现异常情况。

首先导入训练数据，使用 tf.keras.utils 中的方法导入数据，如果未下载过数据可以使用这个下载相应的测试数据：

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

import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
import ssl

ssl._create_default_https_context = ssl._create_unverified_context

TRAIN_DATA_URL = "https://storage.googleapis.com/tf-
datasets/titanic/train.csv"
TEST_DATA_URL = "https://storage.googleapis.com/tf-
datasets/titanic/eval.csv"

train_file_path = tf.keras.utils.get_file("train.csv", TRAIN_DATA_URL)
test_file_path = tf.keras.utils.get_file("eval.csv", TEST_DATA_URL)

np.set_printoptions(precision=3, suppress=True)
```

然后编辑读取 CSV 的方法，使其变为 TensorFlow.data 的可用数据，调用 tf.data.experimental.make_csv_dataset方法：

```
LABEL_COLUMN = 'survived'

def get_dataset(file_path):
  dataset = tf.data.experimental.make_csv_dataset(
      file_path,
      batch_size=12, # 为了示例更容易展示，手动设置较小的值
      label_name=LABEL_COLUMN,
      na_value="?",
      num_epochs=1,
      ignore_errors=True)
  return dataset
```

这里我们重点的标签项目是 survived 这个输出目标。可以看下原始数据的列名称：

survived	sex	age	n_siblings_spouses	
0	male	35.0	0	
0	male	54.0	0	
1	female	58.0	0	
1	female	55.0	0	

为了让数据更加容易读取，可以用下面的代码：

```
np.set_printoptions(precision=3, suppress=True)
```

正如你看到的那样，CSV 文件的每列都会有一个列名。dataset 的构造函数会自动识别这些列名。如果你使用的文件的第一行不包含列名，那么需要将列名通过字符串列表传给 make_csv_dataset 函数的 column_names 参数。可以在 dataset 方法中加入：

```
CSV_COLUMNS = ['survived', 'sex', 'age', 'n_siblings_spouses', 'parch',
'fare', 'class', 'deck', 'embark_town', 'alone']
dataset = tf.data.experimental.make_csv_dataset(
     ...,
     column_names=CSV_COLUMNS,
     ...)
```

这个示例使用了所有的列。如果你需要忽略数据集中的某些列，创建一个包含你需要使用的列的列表，然后传给构造器的（可选）参数 select_columns。

```
dataset = tf.data.experimental.make_csv_dataset(
  ...,
  select_columns = columns_to_use,
  ...)
```

对于包含模型需要预测的值的列是你需要显式指定的。

```
LABEL_COLUMN = 'survived'
```

```
LABELS = [0, 1]
```

然后开始读取 CSV 具体的那些项目，并且将其保存到元组中。

```
raw_train_data = get_dataset(train_file_path)
raw_test_data = get_dataset(test_file_path)

# 如果是离线的文件
# raw_train_data = get_dataset("database/train.csv")
# raw_test_data = get_dataset("database/eval.csv")
```

dataset 中的每个条目都是一个批次，用一个元组（多个样本，多个标签）表示。样本中的数据组织形式是以列为主的张量（而不是以行为主的张量），每条数据中包含的元素个数就是批次大小（这个示例中是 12）。

阅读下面的示例有助于你的理解。

```
examples, labels = next(iter(raw_train_data)) # 第一个批次
print("EXAMPLES: \n", examples, "\n")
print("LABELS: \n", labels)
```

输出：

```
EXAMPLES:
OrderedDict([('sex', <tf.Tensor: id=176, shape=(12,), dtype=string, numpy=
array([b'male', b'male', b'female', b'male', b'male', b'male', b'female',
       b'male', b'male', b'male', b'female', b'male'], dtype=object)>),
('age', <tf.Tensor: id=168, shape=(12,), dtype=float32, numpy=
array([ 7. , 28. , 23. , 28. , 20. , 32. , 17. , 28. , 17. , 28.5, 19. ,
       28. ], dtype=float32)>), ('n_siblings_spouses', <tf.Tensor: id=174,
shape=(12,), dtype=int32, numpy=array([4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
dtype=int32)>), ('parch', <tf.Tensor: id=175, shape=(12,), dtype=int32,
numpy=array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)>), ('fare',
<tf.Tensor: id=173, shape=(12,), dtype=float32, numpy=
array([29.125, 26.55 ,  7.55 ,  7.896,  9.846, 10.5  , 14.458,  7.829,
        8.663,  7.229, 26.   ,  8.05 ], dtype=float32)>), ('class',
<tf.Tensor: id=170, shape=(12,), dtype=string, numpy=
array([b'Third', b'First', b'Third', b'Third', b'Third', b'Second',
       b'Third', b'Third', b'Third', b'Third', b'Second', b'Third'],
      dtype=object)>), ('deck', <tf.Tensor: id=171, shape=(12,),
dtype=string, numpy=
array([b'unknown', b'C', b'unknown', b'unknown', b'unknown', b'unknown',
       b'unknown', b'unknown', b'unknown', b'unknown', b'unknown',
       b'unknown'], dtype=object)>), ('embark_town', <tf.Tensor: id=172,
shape=(12,), dtype=string, numpy=
array([b'Queenstown', b'Southampton', b'Southampton', b'Southampton',
       b'Southampton', b'Southampton', b'Cherbourg', b'Queenstown',
       b'Southampton', b'Cherbourg', b'Southampton', b'Southampton'],
      dtype=object)>), ('alone', <tf.Tensor: id=169, shape=(12,),
dtype=string, numpy=
```

```
array([b'n', b'y', b'y', b'y', b'y', b'y', b'y', b'y', b'y', b'y', b'y',
       b'y'], dtype=object)>)])

LABELS:
tf.Tensor([0 1 1 0 0 0 0 0 0 0 1 0], shape=(12,), dtype=int32)
```

- 数据预处理

分类数据

CSV 数据中的有些列是分类的列。也就是说，这些列只能在有限的集合中取值。使用 tf.feature_column API 创建一个 tf.feature_column.indicator_column 集合，每个 tf.feature_column.indicator_column 对应一个分类的列。实际则是数据限定条件，保存有效的数据。

```
CATEGORIES = {
    'sex': ['male', 'female'],
    'class' : ['First', 'Second', 'Third'],
    'deck' : ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'],
    'embark_town' : ['Cherbourg', 'Southhampton', 'Queenstown'],
    'alone' : ['y', 'n']
}
categorical_columns = []

for feature, vocab in CATEGORIES.items():
  cat_col = tf.feature_column.categorical_column_with_vocabulary_list(
        key=feature, vocabulary_list=vocab)
  categorical_columns.append(tf.feature_column.indicator_column(cat_col))
```

执行完成后，categorical_columns 中存储的元组数据：

```
columns :
[IndicatorColumn(categorical_column=VocabularyListCategoricalColumn(key='se
x', vocabulary_list=('male', 'female'), dtype=tf.string, default_value=-1,
num_oov_buckets=0)),
IndicatorColumn(categorical_column=VocabularyListCategoricalColumn(key='cla
ss', vocabulary_list=('First', 'Second', 'Third'), dtype=tf.string,
default_value=-1, num_oov_buckets=0)),
IndicatorColumn(categorical_column=VocabularyListCategoricalColumn(key='dec
k', vocabulary_list=('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'),
dtype=tf.string, default_value=-1, num_oov_buckets=0)),
IndicatorColumn(categorical_column=VocabularyListCategoricalColumn(key='emb
ark_town', vocabulary_list=('Cherbourg', 'Southhampton', 'Queenstown'),
dtype=tf.string, default_value=-1, num_oov_buckets=0)),
IndicatorColumn(categorical_column=VocabularyListCategoricalColumn(key='alo
ne', vocabulary_list=('y', 'n'), dtype=tf.string, default_value=-1,
num_oov_buckets=0))]
```

- 连续数据

连续数据需要标准化。
写一个函数标准化这些值，然后将这些值改造成 2 维的张量。

```
def process_continuous_data(mean, data):
  # 标准化数据
  data = tf.cast(data, tf.float32) * 1/(2*mean)
  return tf.reshape(data, [-1, 1])


# 现在创建一个数值列的集合。tf.feature_columns.numeric_column API 会使用
normalizer_fn 参数。在传参的时候使用 functools.partial，functools.partial 由使用
每个列的均值进行标准化的函数构成。
MEANS = {
    'age' : 29.631308,
    'n_siblings_spouses' : 0.545455,
    'parch' : 0.379585,
    'fare' : 34.385399
}

numerical_columns = []

for feature in MEANS.keys():
  num_col = tf.feature_column.numeric_column(feature,
normalizer_fn=functools.partial(process_continuous_data, MEANS[feature]))
  numerical_columns.append(num_col)
```

随后打印 numerical_columns:

```
[NumericColumn(key='age', shape=(1,), default_value=None, dtype=tf.float32,
normalizer_fn=functools.partial(<function process_continuous_data at
0x7fba7b3fc158>, 29.631308)),
NumericColumn(key='fare', shape=(1,), default_value=None, dtype=tf.float32,
normalizer_fn=functools.partial(<function process_continuous_data at
0x7fba7b3fc158>, 34.385399)),
NumericColumn(key='n_siblings_spouses', shape=(1,), default_value=None,
dtype=tf.float32, normalizer_fn=functools.partial(<function
process_continuous_data at 0x7fba7b3fc158>, 0.545455)),
NumericColumn(key='parch', shape=(1,), default_value=None,
dtype=tf.float32, normalizer_fn=functools.partial(<function
process_continuous_data at 0x7fba7b3fc158>, 0.379585))]
```

这里使用标准化的方法需要提前知道每列的均值。如果需要计算连续的数据流的标准化的值可以使用 TensorFlow Transform。

- 创建预处理层并且构建、训练模型

将这两个特征列的集合相加，并且传给 tf.keras.layers.DenseFeatures 从而创建一个进行预处理的输入层。

```
# 预处理层
preprocessing_layer =
tf.keras.layers.DenseFeatures(categorical_columns+numerical_columns)
# 构建模型
model = tf.keras.Sequential([
  preprocessing_layer,
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid'),
])

model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy'])
# 训练数据
train_data = raw_train_data.shuffle(500)
test_data = raw_test_data
model.fit(train_data, epochs=18)
test_loss, test_accuracy = model.evaluate(test_data)
print('\n\nTest Loss {}, Test Accuracy {}'.format(test_loss,
test_accuracy))
```

得到准确度和损失值：

```
Test Loss 0.45212358033115213, Test Accuracy 0.810606062412262
```

之后开始用测试数据验证模型并且输出结果：

```
predictions = model.predict(test_data)

# 显示部分结果
for prediction, survived in zip(predictions[:10], list(test_data)[0][1]
[:10]):
  print("Predicted survival: {:.2%}".format(prediction[0]),
        " | Actual outcome: ",
        ("SURVIVED" if bool(survived) else "DIED"))
```

输出结果：

```
Predicted survival: 99.93%  | Actual outcome:  DIED
Predicted survival: 44.35%  | Actual outcome:  DIED
Predicted survival: 88.58%  | Actual outcome:  DIED
Predicted survival: 4.71%  | Actual outcome:  SURVIVED
Predicted survival: 87.09%  | Actual outcome:  DIED
Predicted survival: 99.44%  | Actual outcome:  DIED
```

```
Predicted survival: 39.48%  | Actual outcome:  DIED
Predicted survival: 85.02%  | Actual outcome:  SURVIVED
Predicted survival: 42.41%  | Actual outcome:  DIED
Predicted survival: 1.27%  | Actual outcome:  DIED
```

前十条数据：

survived	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone
0	male	35.0	0	0	8.05	Third	unknown	Southampton	y
0	male	54.0	0	0	51.8625	First	E	Southampton	y
1	female	58.0	0	0	26.55	First	C	Southampton	y
1	female	55.0	0	0	16.0	Second	unknown	Southampton	y
1	male	34.0	0	0	13.0	Second	D	Southampton	y
1	female	15.0	0	0	8.0292	Third	unknown	Queenstown	y
0	female	8.0	3	1	21.075	Third	unknown	Southampton	n
0	male	21.0	0	0	8.05	Third	unknown	Southampton	y
0	female	18.0	2	0	18.0	Third	unknown	Southampton	n
1	female	19.0	0	0	7.8792	Third	unknown	Queenstown	y
1	female	28.0	0	0	7.75	Third	unknown	Queenstown	y
0	male	21.0	0	0	7.8	Third	unknown	Southampton	y
1	female	5.0	1	2	27.75	Second	unknown	Southampton	n
0	male	28.0	0	0	27.7208	First	unknown	Cherbourg	y