Open in Colab

T81-558: Applications of Deep Neural Networks

Module 2: Python for Machine Learning

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 2 Material

Main video lecture:

- Part 2.1: Introduction to Pandas [Video] [Notebook]
- Part 2.2: Categorical Values [Video] [Notebook]
- **Part 2.3: Grouping, Sorting, and Shuffling in Python Pandas** [Video] [Notebook]
- Part 2.4: Using Apply and Map in Pandas for Keras [Video] [Notebook]
- Part 2.5: Feature Engineering in Pandas for Deep Learning in Keras [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

In [1]:
```
try:
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: not using Google CoLab
```

Part 2.3: Grouping, Sorting, and Shuffling

We will take a look at a few ways to affect an entire Pandas data frame. These techniques will allow us to group, sort, and shuffle data sets. These are all essential operations for both data preprocessing and evaluation.

Shuffling a Dataset

There may be information lurking in the order of the rows of your dataset. Unless you are dealing with time-series data, the order of the rows should not be significant. Consider if your training set included employees in a company. Perhaps this dataset is ordered by the number of years the employees were with the company. It is okay to have an individual column that specifies years of service. However, having the data in this order might be problematic.

Consider if you were to split the data into training and validation. You could end up with your validation set having only the newer employees and the training set longer-term employees. Separating the data into a k-fold cross validation could have similar problems. Because of these issues, it is important to shuffle the data set.

Often shuffling and reindexing are both performed together. Shuffling randomizes the order of the data set. However, it does not change the Pandas row numbers. The following code demonstrates a reshuffle. Notice that the program has not reset the row indexes' first column. Generally, this will not cause any issues and allows tracing back to the original order of the data. However, I usually prefer to reset this index. I reason that I typically do not care about the initial position, and there are a few instances where this unordered index can cause issues.

In [2]:
```
import os
import pandas as pd
import numpy as np

df = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/auto-mpg.csv",
    na_values=['NA', '?'])

#np.random.seed(42) # Uncomment this line to get the same shuffle each time
df = df.reindex(np.random.permutation(df.index))

pd.set_option('display.max_columns', 7)
pd.set_option('display.max_rows', 5)
display(df)
```

	mpg	cylinders	displacement	...	year	origin	name
117	29.0	4	68.0	...	73	2	fiat 128
245	36.1	4	98.0	...	78	1	ford fiesta
...	...	...	...	...	...	...	...
88	14.0	8	302.0	...	73	1	ford gran torino
26	10.0	8	307.0	...	70	1	chevy c20

398 rows × 9 columns

The following code demonstrates a reindex. Notice how the reindex orders the row indexes.

In [3]:
```
pd.set_option('display.max_columns', 7)
pd.set_option('display.max_rows', 5)

df.reset_index(inplace=True, drop=True)
display(df)
```

	mpg	cylinders	displacement	...	year	origin	name
0	29.0	4	68.0	...	73	2	fiat 128
1	36.1	4	98.0	...	78	1	ford fiesta
...	...	...	...	...	...	...	...
396	14.0	8	302.0	...	73	1	ford gran torino
397	10.0	8	307.0	...	70	1	chevy c20

398 rows × 9 columns

Sorting a Data Set

While it is always good to shuffle a data set before training, during training and preprocessing, you may also wish to sort the data set. Sorting the data set allows you to order the rows in either ascending or descending order for one or more columns. The following code sorts the MPG dataset by name and displays the first car.

In [4]:
```
import os
import pandas as pd

df = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/auto-mpg.csv",
    na_values=['NA', '?'])

df = df.sort_values(by='name', ascending=True)
print(f"The first car is: {df['name'].iloc[0]}")
```

```
pd.set_option('display.max_columns', 7)
pd.set_option('display.max_rows', 5)
display(df)
```

```
The first car is: amc ambassador brougham
```

	mpg	cylinders	displacement	...	year	origin	name
96	13.0	8	360.0	...	73	1	amc ambassador brougham
9	15.0	8	390.0	...	70	1	amc ambassador dpl
...	...	...	...	...	...	...	...
325	44.3	4	90.0	...	80	2	vw rabbit c (diesel)
293	31.9	4	89.0	...	79	2	vw rabbit custom

398 rows × 9 columns

Grouping a Data Set

Grouping is a typical operation on data sets. Structured Query Language (SQL) calls this operation a "GROUP BY." Programmers use grouping to summarize data. Because of this, the summarization row count will usually shrink, and you cannot undo the grouping. Because of this loss of information, it is essential to keep your original data before the grouping.

We use the Auto MPG dataset to demonstrate grouping.

In [5]:
```
import os
import pandas as pd

df = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/auto-mpg.csv",
    na_values=['NA', '?'])

pd.set_option('display.max_columns', 7)
pd.set_option('display.max_rows', 5)
display(df)
```

	mpg	cylinders	displacement	...	year	origin	name
0	18.0	8	307.0	...	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	...	70	1	buick skylark 320
...	...	...	...	...	...	...	...
396	28.0	4	120.0	...	82	1	ford ranger
397	31.0	4	119.0	...	82	1	chevy s-10

398 rows × 9 columns

You can use the above data set with the group to perform summaries. For example, the following code will group cylinders by the average (mean). This code will provide the grouping. In addition to **mean**, you can use other aggregating functions, such as **sum** or **count**.

In [6]:
```
g = df.groupby('cylinders')['mpg'].mean()
g
```

Out[6]:
```
cylinders
3    20.550000
4    29.286765
5    27.366667
6    19.985714
8    14.963107
Name: mpg, dtype: float64
```

It might be useful to have these **mean** values as a dictionary.

In [7]:
```
d = g.to_dict()
d
```

Out[7]:
```
{3: 20.55,
 4: 29.28676470588236,
 5: 27.366666666666664,
 6: 19.985714285714284,
 8: 14.963106796116508}
```

A dictionary allows you to access an individual element quickly. For example, you could quickly look up the mean for six-cylinder cars. You will see that target encoding, introduced later in this module, uses this technique.

In [8]:
```
d[6]
```

Out[8]:
```
19.985714285714284
```

The code below shows how to count the number of rows that match each cylinder count.

In [9]:
```
df.groupby('cylinders')['mpg'].count().to_dict()
```

Out[9]:
```
{3: 4, 4: 204, 5: 3, 6: 84, 8: 103}
```