BuildingABalancedDataSet

August 8, 2023

1 Building a Balanced Data Set

Run the code cells below to import the packages.

[1]:
```
import pandas as pd
import numpy as np
import os
```

1.1 Step 1: Inspect the Data

Run the code cell below to load the dataset and save it as a Pandas DataFrame.

[2]:
```
filename = os.path.join(os.getcwd(), "data", "adult.data.partial")
df = pd.read_csv(filename, header=0)
```

Run the code cell below to glance at the data. Note that one column has the name "label". This column will serve as the label (value we want to predict). The `label` column contains two possible values (or two possible classes) `<=50k` and `>50k`. This corresponds to income. Since the label can be one of two classes, this dataset is suitable for a binary classification problem. The other columns will serve as the features.

Note: Throughout the exercises you will see some terms used interchangeably. You will see features and labels referred to as "variables." Since our data will be in the form of a DataFrame, we will often use the term "column" interchangeably with "feature" or "label," depending on the column in question. For example, in our `adult` dataset, `age` can be referred to as both a "column" and a "feature."

[3]:
```
df.head()
```

[3]:
```
   age  workclass  fnlwgt     education  education-num      marital-status  \
0   36  State-gov  112074     Doctorate             16       Never-married
1   35    Private   32528       HS-grad              9  Married-civ-spouse
2   21    Private  270043  Some-college             10       Never-married
3   45    Private  168837  Some-college             10  Married-civ-spouse
4   39    Private  297449     Bachelors             13  Married-civ-spouse

          occupation   relationship   race  sex_selfID  capital-gain  \
0     Prof-specialty  Not-in-family  White  Non-Female             0
1  Handlers-cleaners        Husband  White  Non-Female             0
2      Other-service      Own-child  White      Female             0
3       Adm-clerical           Wife  White      Female             0
```

```
4     Prof-specialty        Husband  White  Non-Female             0

   capital-loss  hours-per-week native-country  label
0             0              45  United-States  <=50K
1             0              45  United-States  <=50K
2             0              16  United-States  <=50K
3             0              24         Canada   >50K
4             0              40  United-States   >50K
```

Run the code cell below to see the shape of the data.

[4]:
```
df.shape
```

[4]:
```
(7000, 15)
```

1.2 Step 2: Random Sampling From the Data

Complete the code in the cell below to randomly select 30% of the examples (rows) and save them to new DataFrame `df_subset`. To accomplish this, use `np.random.choice()` to obtain 30% of row indices and save the result to variable `indices`.

1.2.1 Graded Cell

The cell below will be graded. Remove the line "raise NotImplementedError()" before writing your code.

[6]:
```
percentage = 0.3
num_rows = df.shape[0]

# YOUR CODE HERE
indices=np.random.choice(df.index,size=int(percentage*num_rows),replace=False)
df_subset=df.loc[indices]
```

1.2.2 Self-Check

Run the cell below to test the correctness of your code above before submitting for grading. Do not add code or delete code in the cell.

[7]:
```
# Run this self-test cell to check your code;
# do not add code or delete code in this cell
from jn import testSubset

try:
    p, err = testSubset(df, df_subset, indices)
    print(err)
except Exception as e:
    print("Error!\n" + str(e))
```

```
Correct!
```

The code cell below displays the shape (number of rows and columns) of the resulting sample. Run the cell and examine the results. DataFrame `df_subset` should contain 30% of the row number in the original DataFrame.

[8]:
```
print(df.shape)
print(df_subset.shape)
```

```
(7000, 15)
(2100, 15)
```

1.3 Step 3: Verifying (im)balance

Is our sample *balanced* with respect to (self-reported) sex? In order to answer that, first we'd like to know how many categories exist for the 'sex_selfID' values in our data.

1.3.1 Listing unique values of a column using Pandas `unique()` Method.

You will be using the `pandas` `unique()` method to display all unique values from the column `sex_selfID`. To see how to use the `unique()` method, run the cell below and examine the documentation.

You can also access the online documentation.

[9]:
```
pd.unique?
```

The `unique()` method follows the `pandas` series after a dot, like so: `<pandas_series>.unique()`. We want to apply `unique` to the entire column with the name `sex_selfID` and save the result to the variable `unique_ssID`.

1. To select a column, simply write `df['<column_name>']`.
2. To call the `unique() method`, write `df['<column_name>'].unique()`.

Complete the code in the cell below. Note that the `unique()` method returns a `numpy` array.

1.3.2 Graded Cell

The cell below will be graded. Remove the line "raise NotImplementedError()" before writing your code.

[11]:
```
# YOUR CODE HERE
unique_ssID=df['sex_selfID'].unique()
```

1.3.3 Self-Check

Run the cell below to test the correctness of your code above before submitting for grading. Do not add code or delete code in the cell.

[12]:
```
# Run this self-test cell to check your code;
# do not add code or delete code in this cell
from jn import testUnique
```

```
try:
    p, err = testUnique(df, unique_ssID)
    print(err)
except Exception as e:
    print("Error!\n" + str(e))
```

```
Correct!
```

It is good to examine whether there are any non-standard spellings or unexpected missing values in the columns. In this case, we have exactly two unique values, (`Non-Female` and `Female`) so we can proceed. Note that this dataset, although still widely used in ML research, is from 1994 and is a bit outdated.

1.3.4 Calculating the Proportion of Each Class

How many 'Female' examples are in our data sample?

The code cell below uses `np.sum()` to sum up the `True` values that indicate whether a row has `Female` in the `sex_selfID` field. It divides that sum by the total number of rows in the DataFrame `df_subset`. Run the code to display the results. Note that the sample is not balanced with respect to self-reported sex (assuming that we want balance for the two classes).

[13]:
```
percent_female = np.sum(df_subset['sex_selfID']=='Female')/
 ↪df_subset['sex_selfID'].shape[0]
percent_female
```

[13]:
```
0.3252380952380952
```

For a column that has a large amount of categories, doing the above computation for each value would be tedious. One of the more efficient ways to compute class proportions would be to use the `value_counts()` method from `Pandas`. Run the cells below.

[14]:
```
counts = df_subset['sex_selfID'].value_counts()
counts
```

[14]:
```
Non-Female    1417
Female         683
Name: sex_selfID, dtype: int64
```

[15]:
```
counts['Female']/sum(counts.values)
```

[15]:
```
0.3252380952380952
```

Now let's examine balance with respect to race. In the code cell below, display the total number of examples belonging to each `race` column in DataFrame `df_subset`. Use the more efficient Pandas `value_counts()` method, as demonstrated above.

1. Get the `race` column from `df_subset` using bracket notation.
2. Apply the `value_counts()` method as demonstrated above.
3. Save the results to variable `num_examples`.

Run the code cell and examine the results. You'll note that the sample is unbalanced with respect to the race categories present in the data sample.

1.3.5 Graded Cell

The cell below will be graded. Remove the line "raise NotImplementedError()" before writing your code.

[16]:
```
# YOUR CODE HERE
num_examples=df_subset['race'].value_counts()
```

1.3.6 Self-Check

Run the cell below to test the correctness of your code above before submitting for grading. Do not add code or delete code in the cell.

[17]:
```
# Run this self-test cell to check your code;
# do not add code or delete code in this cell
from jn import testNumExamples

try:
    p, err = testNumExamples(df_subset, num_examples)
    print(err)
except Exception as e:
    print("Error!\n" + str(e))
```

```
Correct!
```

1.3.7 Detecting group (im)balance with respect to the label using Pandas `groupby()` Method.

Generally, there are many different kinds of balance. The simplest way to define 'balance' is to require that the total number of representatives in each category is the same. For the purposes of fairness, this may mean that the number of females is the same as the number of non-females.

But what about the label? The usual kind of balance that machine learning engineers seek is that of labels. In our dataset, the label is one of two income values: <=50k or >50k. We want the dataset to have equal representation of 'high income' and 'low income' examples. A more nuanced approach, however, is to look for balance in each intersection of labels and sensitive feature values. Indeed, imagine a dataset in which the number of white and non-white people is the same, yet the label values are not balanced among the two groups. Training an ML model on such a dataset would likely produce a biased model (we will discuss this in more detail further in the course). In other words, we would like to see that for each value of the label, there is an equal representation between the demographic subgroups.

Establish whether there is a balance between the two categories of `sex_selfID` with respect to the label. In other words, check if in our sample, the number of females who have one kind of label is the same as the number of non-females with that label (i.e., the value in the column `label` is '<=50K'). Do the same for the other label (the value in the column `label` is '>50K'). You can do this by using the Pandas `groupby()` method to aggregate the subsample data by `sex_selfID` and `label`. Then, use the `size()` method on the resulting object. For more information about the method `groupby()`, consult the online documentation.

The code cell below accomplishes this task. Run the cell and inspect the results.

[18]:
```
df_subset.groupby(['sex_selfID', 'label']).size()
```

[18]:
```
sex_selfID  label
Female      <=50K     608
            >50K       75
Non-Female  <=50K    1016
            >50K      401
dtype: int64
```

1.3.8 Addressing imbalance: upsampling the underrepresented group.

It seems that the females are underrepresented in the 'higher income' group, compared to non-females. What can we do about this? There are many ways to go about dealing with this imbalance. For the purposes of this exercise, we will sample additional points from the original full DataFrame `df` into the group of Females with income >50k. We will sample until the ratio of the two subgroup sizes (Females with income >50k and Females with income <=50K) is the same as the ratio of higher to lower income non-females.

The next two cells are non-graded. Simply run the cells below and inspect the results.

[19]:
```
low_income_nonfemale, high_income_nonfemale = df_subset.groupby(['sex_selfID',␣
↪'label']).size()['Non-Female']
class_balance_nonfemale = high_income_nonfemale / low_income_nonfemale

low_income_female, high_income_female = df_subset.groupby(['sex_selfID',␣
↪'label']).size()['Female']

add_sample_size = int(class_balance_nonfemale*low_income_female -␣
↪high_income_female)
add_sample_size # we need this many more points in (Female)&(>50K) group for␣
↪balance
```

[19]:
```
164
```

[20]:
```
# Subset the original data: exclude entries that are already in our sample:
df_never_sampled = df.drop(labels=df_subset.index, axis=0, inplace=False)

# Filter that subset to include only the type of examples that we want to␣
↪upsample: Females, higher income
condition = (df_never_sampled['label']=='>50K') &␣
↪(df_never_sampled['sex_selfID']=='Female')
df_never_sampled_target = df_never_sampled[condition]

# Sample from the resulting set
size=min(add_sample_size, df_never_sampled_target.shape[0])
indices = np.random.choice(df_never_sampled_target.index, size=size,␣
↪replace=False)

# Append the selected examples to our original sample
rows = df.loc[indices]
df_balanced_subset = df_subset.append(rows)
```

```
df_balanced_subset.head()
```

[20]:
```
      age  workclass  fnlwgt     education  education-num      marital-status  \
1431   18    Private  123856          11th              7       Never-married
5500   24    Private  235071          11th              7  Married-civ-spouse
6132   36  Local-gov  212856          11th              7       Never-married
707    28    Private  188236  Some-college             10       Never-married
2399   40    Private  404573  Some-college             10       Never-married

         occupation   relationship   race  sex_selfID  capital-gain  \
1431          Sales      Own-child  White      Female             0
5500   Craft-repair        Husband  White  Non-Female             0
6132  Other-service      Unmarried  White      Female             0
707    Adm-clerical  Not-in-family  White      Female             0
2399          Sales  Not-in-family  White      Female             0

      capital-loss  hours-per-week native-country  label
1431             0              49  United-States  <=50K
5500             0              50  United-States   >50K
6132             0              23  United-States  <=50K
707              0              40  United-States  <=50K
2399             0              44  United-States  <=50K
```

The code cell below checks the balance of this new DataFrame `balanced_subset_df`. Run the cell below and examine the results.

[21]:
```
df_balanced_subset.groupby(['sex_selfID', 'label']).size()
```

[21]:
```
sex_selfID  label
Female      <=50K     608
            >50K      239
Non-Female  <=50K    1016
            >50K      401
dtype: int64
```

The resulting balance is not perfect, but it is better than before!