D8_ml

March 11, 2024

1 Workbook : Machine Learning

For our last section workbook (so that next week you can ask questions about and work on your final projects in section), we're going to work with a dataset all about craft beer. We'll work to predict what type of beer each is based on the characteristics of that beer.

Disclaimer: Working with data about beer does *NOT* mean that I'm encouraging the drinking of beer by students. In fact, your professor doesn't even like beer (blech). Specifically, individuals under the age of 21 are not legally allowed to consume alcoholic beverages, but lucky for you all, that doesn't stop us from working with data on the topic!

The data we'll use here come from a publicly-available Kaggle dataset on craft beer.

2 Part I : Data, Wrangling, & EDA

To get started, you'll need to **import the following**: * `pandas` as `pd` * `numpy` as `np` * from `sklearn.svm`: `SVC` * from `sklearn.metrics`: `confusion_matrix`, `classification_report`, `precision_recall_fscore_support`

[1]:
```
# YOUR CODE HERE
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report,␣
↪precision_recall_fscore_support
```

[2]:
```
assert pd
assert np
assert SVC
assert confusion_matrix
assert classification_report
assert precision_recall_fscore_support
```

Now that you're setup to go in Python, **read in the `'breweries.csv'` file from the `data/` directory. Assign this to the variable `breweries`**. Then, **read in the file `beers.csv` from the `data/` directory. Assign this to the variable `beers`.**

[3]:
```
# YOUR CODE HERE
breweries = pd.read_csv('data/breweries.csv')
```

```python
beers = pd.read_csv('data/beers.csv')
```

[4]:
```python
assert breweries.shape == (558, 4)
assert beers.shape == (2410, 8)
```

Run the code below to take a **look at the first few rows of each dataset** to give yourself an idea of what data are inclued in each dataset. Notice if there are any common columns between the two datasets.

[5]:
```python
breweries.head()
```

[5]:
```
   Unnamed: 0                       name           city state
0           0          NorthGate Brewing    Minneapolis    MN
1           1  Against the Grain Brewery     Louisville    KY
2           2   Jack's Abby Craft Lagers     Framingham    MA
3           3  Mike Hess Brewing Company      San Diego    CA
4           4    Fort Point Beer Company  San Francisco    CA
```

[6]:
```python
beers.head()
```

[6]:
```
   Unnamed: 0    abv  ibu    id                 name  \
0           0  0.050  NaN  1436             Pub Beer
1           1  0.066  NaN  2265          Devil's Cup
2           2  0.071  NaN  2264  Rise of the Phoenix
3           3  0.090  NaN  2263             Sinister
4           4  0.075  NaN  2262        Sex and Candy

                            style  brewery_id  ounces
0             American Pale Lager         408    12.0
1         American Pale Ale (APA)         177    12.0
2                    American IPA         177    12.0
3  American Double / Imperial IPA         177    12.0
4                    American IPA         177    12.0
```

To get a quick handle on what's going on these data, **save the number of missing values in each variable of the variables in the `beers` dataset to `null_beers`.** Hint: use `.isnull()`

[7]:
```python
# YOUR CODE HERE
null_beers = beers.isnull().sum()
```

[8]:
```python
assert null_beers.sum() == 1072
```

We're going to try to predict the `style` of beer from its alcohol by volume (`abv`) and its international bitterness unites (`ibu`). To do this, **remove any beers from our `beers` dataset where data are missing for any of these three values. Store this back into hte `beers` dataset.**

Note that you may not always want to take this approach and removing samples from your dataset will not always be appropriate, but for this example, it's a reasonable approach.

[9]:
```
# YOUR CODE HERE
beers.dropna(subset=['style','abv', 'ibu'], inplace=True)
print(beers.shape)
```

```
(1403, 8)
```

[10]:
```
assert beers.shape == (1403, 8)
```

Using the `beers` dataset you've not got, **merge `beers` and `breweries` together using a left join. Assign this to the variable `beer_df`. Be sure to look at the first few rows of `beer_df`.**

[11]:
```
# YOUR CODE HERE
beers = beers.drop('Unnamed: 0', axis = 1)
beer_df = pd.merge(beers, breweries, how='left')
print(beer_df.head())
print(beers.shape)
```

```
     abv   ibu    id                                name  \
0  0.061  60.0  1979                        Bitter Bitch
1  0.099  92.0  1036                       Lower De Boom
2  0.079  45.0  1024                       Fireside Chat
3  0.044  42.0   876                     Bitter American
4  0.049  17.0   802  Hell or High Watermelon Wheat (2009)

                     style  brewery_id  ounces  Unnamed: 0 city state
0  American Pale Ale (APA)         177    12.0         NaN  NaN   NaN
1       American Barleywine        368     8.4         NaN  NaN   NaN
2             Winter Warmer        368    12.0         NaN  NaN   NaN
3  American Pale Ale (APA)         368    12.0         NaN  NaN   NaN
4    Fruit / Vegetable Beer        368    12.0         NaN  NaN   NaN
(1403, 7)
```

[12]:
```
assert beer_df.shape == (1403, 10)
```

Use and take a look at the output of the `describe()` method to describe the quantitative variables in your `beer_df` dataset.

[13]:
```
beer_df.describe()
```

[13]:
```
               abv          ibu           id   brewery_id       ounces  \
count  1403.000000  1403.000000  1403.000000  1403.000000  1403.000000
mean      0.059919    42.739843  1413.888810   223.375624    13.510264
std       0.013585    25.962692   757.572191   150.387510     2.254112
min       0.027000     4.000000     1.000000     0.000000     8.400000
25%       0.050000    21.000000   771.000000    95.500000    12.000000
50%       0.057000    35.000000  1435.000000   198.000000    12.000000
75%       0.068000    64.000000  2068.500000   350.000000    16.000000
```

```
max       0.125000   138.000000  2692.000000   546.000000    32.000000

       Unnamed: 0
count         0.0
mean          NaN
std           NaN
min           NaN
25%           NaN
50%           NaN
75%           NaN
max           NaN
```

Be sure to look at the output you just generated. What do you learn? Do any values surprise you? Are there any with really big standard deviations? Does this make sense? (Feel free to edit this cell with any observations/notes)

Now, let's take a look and **see how many different styles of beer we have in our datset.** The `value_counts` method may help you accomplish this. Assign it to `beer_counts` and print it.

[14]:
```
# YOUR CODE HERE
beer_counts = beer_df['style'].value_counts()
print(beer_counts)
```

```
style
American IPA                          301
American Pale Ale (APA)               153
American Amber / Red Ale               77
American Double / Imperial IPA         75
American Blonde Ale                    61
                                     …
Roggenbier                              1
Smoked Beer                             1
Euro Pale Lager                         1
Other                                   1
American Double / Imperial Pilsner      1
Name: count, Length: 90, dtype: int64
```

[15]:
```
assert beer_counts[0] == 301
assert len(beer_counts) == 90
```

Due to limitations in time here in section, let's just try to predict the four most common `styles` of beer. **Filter your `beer_df` dataset to only include entries from the four most common styles of beer.** Store this filtered dataset into `beer_df`.

[16]:
```
# YOUR CODE HERE
top_four_styles = beer_df['style'].value_counts().index.tolist()[:4]

beer_df = beer_df[beer_df['style'].isin(top_four_styles)]
```

```
print(beer_df.shape)
styles = beer_df['style'].unique()
print(styles)
print(len(styles))
```

```
(606, 10)
['American Pale Ale (APA)' 'American IPA' 'American Double / Imperial IPA'
 'American Amber / Red Ale']
4
```

[17]:
```
assert beer_df.shape == (606, 10)
styles = beer_df['style' ].value_counts().index.tolist()
assert len(styles) == 4
```

3 Part II : Prediction Model

Let's start to build our model! To do so, **create a variable `num_training` that includes the number of samples that corresponds to 80% of our total samples in our `beer_df` dataset. Be sure that this is an integer. Also, create a variable `num_testing` including the number corresponding to 20% of our total samples.**

[18]:
```
# YOUR CODE HERE
num_training = int(0.8 * len(beer_df))

num_testing = len(beer_df) - num_training

print(num_training)
print(num_testing)
```

```
484
122
```

[19]:
```
assert num_training == 484
assert num_testing == 122
```

To model these data, **split your data into `beer_X`, which includes the `abv` and `ibu` columns from `beer_df` (predictors). This should be a pandas DataFrame. The outcome variable will be `style`. Assign the outcome variable to the variable `beer_Y`. This should be a numpy array.**

[20]:
```
# YOUR CODE HERE
beer_X = beer_df[['abv', 'ibu']]

beer_Y = beer_df['style'].values

print(type(beer_X), type(beer_Y))
```

```python
print(beer_X.shape, beer_Y.shape)
```

```
<class 'pandas.core.frame.DataFrame'> <class 'numpy.ndarray'>
(606, 2) (606,)
```

[21]:
```python
assert type(beer_Y) == np.ndarray
assert beer_Y.shape == (606,)
assert beer_X.shape == (606, 2)
```

Before running our model, we'll need to **split our data into a training and test set. Use `num_training` (created above) to extract the following variables**: * from `beer_X`, generate : `beer_train_X`, `beer_test_X` * from `beer_Y`, generate: `beer_train_Y`, `beer_test_Y`

[22]:
```python
# YOUR CODE HERE
beer_train_X = beer_X[:num_training]
beer_test_X = beer_X[-num_testing:]

beer_train_Y = beer_Y[:num_training]
beer_test_Y = beer_Y[-num_testing:]
```

[23]:
```python
assert len(beer_train_X) == 484
assert len(beer_test_X) == 122
```

To train our model, we'll use a linear SVM classifier. Here a function has been defined for you. **Run the following cell, but be sure you understand what the function is doing.**

[24]:
```python
def train_SVM(X, y, kernel='linear'):
    clf = SVC(kernel=kernel)
    clf.fit(X, y)

    return clf
```

Using the `train_SVM` function defined above, **train your model. Assign this output to `beer_clf`.**

[25]:
```python
# YOUR CODE HERE
beer_clf = train_SVM(beer_train_X, beer_train_Y)

print(isinstance(beer_clf, SVC))
print(hasattr(beer_clf, "predict"))
```

```
True
True
```

[26]:
```python
assert isinstance(beer_clf, SVC)
assert hasattr(beer_clf, "predict")
```

Now, **generate predictions from your training and test sets of predictors using the `predict` method. Assign your predictions from the training data to `beer_predicted_train_Y`. Assign your predictison from the test data to `beer_predicted_test_Y`.**

[27]:
```
# YOUR CODE HERE
beer_clf = train_SVM(beer_train_X, beer_train_Y, kernel='linear')

beer_predicted_train_Y = beer_clf.predict(beer_train_X)
beer_predicted_test_Y = beer_clf.predict(beer_test_X)

print(beer_predicted_train_Y.shape, beer_predicted_test_Y.shape)
```

```
(484,) (122,)
```

[28]:
```
assert beer_predicted_train_Y.shape == (484,)
assert beer_predicted_test_Y.shape == (122,)
```

4 Part III : Model Assessment

At this point, you should have built your model and generated predictions using that model for both your training and test datasets.

Let's determine how our predictor did. **Generate a `classification_report` from sklearn for the predictions generated for your training data relative to the truth (from the original beers dataset). Save the output to `class_report_pred` and print it.**

[29]:
```
class_report_train = None
# YOUR CODE HERE
class_report_train = classification_report(beer_train_Y, beer_predicted_train_Y)
print(class_report_train)
```

```
                               precision    recall  f1-score   support

      American Amber / Red Ale       0.82      0.45      0.58        69
American Double / Imperial IPA       0.76      0.25      0.37        53
                  American IPA       0.69      0.84      0.76       236
         American Pale Ale (APA)       0.57      0.64      0.60       126

                      accuracy                           0.67       484
                     macro avg       0.71      0.54      0.58       484
                  weighted avg       0.69      0.67      0.65       484
```

[30]:
```
assert len(class_report_train) == 578
```

What are precision and recall? What do these numbers represent? How accurate are our predictions?

Generate a `classification_report_test` for the predictions generated for your *test* data relative to the truth (from the original beers dataset). Save the output to `class_report_test` and print it.

[31]:
```
class_report_test = None
# YOUR CODE HERE
class_report_test = classification_report(beer_test_Y, beer_predicted_test_Y)
print(class_report_test)
```

```
                               precision    recall  f1-score   support

      American Amber / Red Ale       0.62      0.62      0.62         8
American Double / Imperial IPA       0.78      0.32      0.45        22
                  American IPA       0.70      0.72      0.71        65
      American Pale Ale (APA)        0.55      0.78      0.65        27

                      accuracy                           0.66       122
                     macro avg       0.66      0.61      0.61       122
                  weighted avg       0.68      0.66      0.64       122
```

[32]:
```
assert len(class_report_test) == 578
```

How is our model performing? Does this dffer between training and test data? Where does it have trouble? Where does it perform well? Do we have thoughts as to why? One way to determine where a model is going wrong is to look at a confusion matrix. **Generate a confusion matrix for the training data predictions as well as the ground truth from the `beer_df` dataset. Save this to `conf_mat_train`**

[33]:
```
conf_mat_train = None
# YOUR CODE HERE
conf_mat_train = confusion_matrix(beer_train_Y, beer_predicted_train_Y)
print(conf_mat_train)
```

```
[[ 31   1  10  27]
 [  0  13  40   0]
 [  0   3 198  35]
 [  7   0  38  81]]
```

[34]:
```
assert conf_mat_train[0,0] == 31
assert conf_mat_train[-1,-1] == 81
assert conf_mat_train.shape == (4,4)
```

Generate a confusion matrix for the testing data. Save this to `conf_mat_test`

[35]:
```
# YOUR CODE HERE
conf_mat_test = confusion_matrix(beer_test_Y, beer_predicted_test_Y)
print(conf_mat_test)
```

```
[[ 5  0  2  1]
 [ 1  7 14  0]
 [ 0  2 47 16]
 [ 2  0  4 21]]
```

[36]:
```
assert conf_mat_test[-1,-1] == 21
assert conf_mat_test.shape == (4,4)
assert conf_mat_test[0,0] == 5
```

While this is a somewhat small example using a limited dataset for prediction, we hope you have a better understanding of how to approach a machine learning question, knowing specifically what training and test datasets are used for, how to build a model, and how to assess model/prediction performance. **Feel free to try different models, include more beer types in your analysis or ask a completely different prediction question!**