Lab 6: Classification in Scikit-Learn

Data Science for Biologists • University of Washington • BIOL 419/519 • Winter 2019

Course design and lecture material by [Bingni Brunton (https://github.com/bwbrunton)](https://github.com/bwbrunton) and [Kameron Harris (https://github.com/kharris/)](https://github.com/kharris/). Lab design and materials by [Eleanor Lutz (https://github.com/eleanorlutz/)](https://github.com/eleanorlutz/), with helpful comments and suggestions from Bing and Kam.

Table of Contents

1. Review of importing and inspecting data
2. Split a dataset into a training and test set
3. Train a machine learning classifier using scikit-learn
4. Bonus exercises

Helpful resources

- [Python Data Science Handbook (http://shop.oreilly.com/product/0636920034919.do)](http://shop.oreilly.com/product/0636920034919.do) by Jake VanderPlas
- [An introduction to machine learning with Scikit-Learn (https://scikit-learn.org/stable/tutorial/basic/tutorial.html)](https://scikit-learn.org/stable/tutorial/basic/tutorial.html)
- [Scikit-Learn user guide (https://scikit-learn.org/stable/user_guide.html)](https://scikit-learn.org/stable/user_guide.html)
- [Scikit-Learn Cheat Sheet (https://datacamp-community-prod.s3.amazonaws.com/5433fa18-9f43-44cc-b228-74672efcd116)](https://datacamp-community-prod.s3.amazonaws.com/5433fa18-9f43-44cc-b228-74672efcd116) by Python for Data Science

Data

- The data in this lab is originally from [USA Forensic Science Service (https://archive.ics.uci.edu/ml/datasets/Glass+Identification)](https://archive.ics.uci.edu/ml/datasets/Glass+Identification) and was edited for teaching purposes.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

This week's lab requires plotting several different classifications in different colors, so set the default matplotlib style to a colorblind-friendly setting:

In [ ]:
```
plt.style.use("seaborn-colorblind")
```

Lab 6 Part 1: Review of importing and inspecting data

This week's data is from the USA Forensic Science Service, and contains information about 214 samples from seven different types of glass (vehicle windows, tableware, headlamps, etc). By analyzing various properties of the glass, such as the refractive index or the proportion of different chemical elements (Na, Mg, Al, etc.), this dataset can be used to predict the source of unknown glass found at crime scenes.

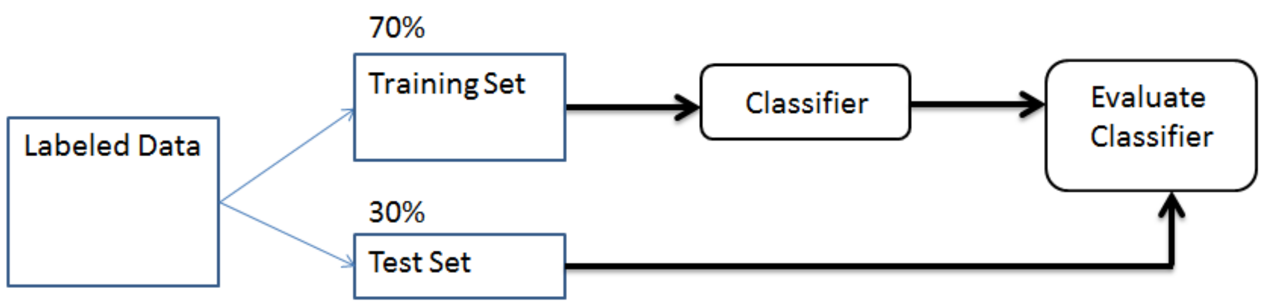
Exercise 1: Read in the dataset in the file `"./data/Lab_06/glass_properties_data.csv"` as a `Pandas` dataframe called `df`. Display the `head()` of the data and `describe()` the dataframe to make sure your `df` variable was imported properly and has the expected columns.

In [ ]:
```
```

Lab 6 Part 2: Split the dataset into a training set and test set

Using this dataset, we would like to create a machine learning classifier that can be used to categorize unknown glass samples.

A machine learning classifier is trained based on one particular set of data. A separate dataset is used to evaluate the accuracy of the classifier. For this lab we'll use 70% of the data for training and save the remaining 30% for testing the classifier. This means that we first need to separate a random 70% of the data into a `train_data` variable, and the other 30% into a different `test_data` variable.



Credit: A schematic overview of the classification process, by Ahmet Taspinar

Exercise 2: Each sample in this dataset has an `ID` number from 1 to 214. Create a `numpy` vector called `indices` that contains all of the index numbers (all integers from 1 to 214). We will use this vector to shuffle the data `ID` s and separate the `ID` s into training and testing groups. Print the vector.

In [ ]:

Exercise 3: The `np.random.shuffle()` function can be used to randomly shuffle everything stored in a list or `numpy` array. Run the following code block and confirm that the ID numbers stored in `indices` has indeed been shuffled:

In [ ]:

Exercise 4: If we take the first 70% of the shuffled `indices` array, this should contain all of the data ID numbers to use for our training dataset. Create a variable called `train_indices` that contains just the first 70% of `indices`. Create another variable called `test_indices` that contains the rest of `indices`. If `indices` is not divisible into clean 70%-30% segments, you can round to the nearest number. Print the length of `train_indices` and `test_indices`. Print `test_indices`.

In [ ]:

```

```

Now we have two variables that hold the `ID` numbers for a randomized selection of 70% and 30% of the data. We can now use these `ID` numbers to separate the data into training and test sets:

In [ ]:

```
train_data = df[df["ID"].isin(train_indices)]
test_data = df[df["ID"].isin(test_indices)]
```

Exercise 5: Evaluate the `train_data` dataframe you just constructed to make sure it looks correct, using the following steps:

- Print the number of rows in `train_data` and the original data.
- Print the percentage of rows in the original data is included in `train_data`. Is this close to the expected 70%?
- Display the `head()` of the `train_data` dataframe and see if the data is a random selection of the original. *Hint:* Look at the index of the dataframe.

In [ ]:

```

```

Exercise 6: Evaluate the `test_data` dataframe similarly to make sure it looks correct:

- Print the number of rows in `test_data`.
- Print the percentage of rows in the original data is included in `test_data`. Is this close to the expected 30%?
- Display the `head()` of the `test_data` dataframe and see if the data is a random selection of the original.
- Does the number of rows in `test_data` and `train_data` add up to the number of rows in the full data?

In [ ]:

```

```

Splitting data using Pandas

Instead of dividing the data into training and test sets manually, `pandas` also has a builtin function to sample a random subset of the dataset (as do other libraries including `scikit-learn`. The previous data splitting can also be done in two lines (for future reference):

In [ ]:
```
train_data = df.sample(frac=0.7)
test_data = df.drop(train_data.index)
```

Lab 6 Part 3: Train a machine learning classifier using scikit-learn

Like `numpy` or `pandas`, [scikit-Learn (https://scikit-learn.org/stable/)](https://scikit-learn.org/stable/) is a python library for machine learning. In today's lab we will make a Linear Discriminant Analysis classifier by importing this function from the `sklearn.discriminant_analysis` module.

Linear Discriminant Analysis

In class we discussed the Linear Discriminant Analysis method for two variables. The scikit-learn library can also extrapolate this principle to run a LDA on many variables. To run a LDA on our dataset we will import the `LDA` function from the Scikit learn (`sklearn`) library.

In [ ]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

To use the `LinearDiscriminantAnalysis` function we need to first separate the data into the data itself and the classifications:

In [ ]:
```
train_class = train_data["Glass_Type"] # known classification answers
train_vals = train_data.drop(["Glass_Type", "ID"], axis=1) # correspondi
# make sure to drop the ID since it is not a characteristic of the glass

display(train_class.head())
display(train_vals.head())
```

Exercise 7: Similarly, split the `test_data` test dataset into a variable called `test_class` that holds all of the `Glass_Type` classifications, and another variable called `test_vals` that contains the corresponding measurements.

In [ ]:
```
```

Now that the training data is separated into the values and the known classifier values, these two dataframes can be used to train the classifier:

In [ ]:
```
classifier = LinearDiscriminantAnalysis()
classifier.fit(train_vals, train_class)
```

To test the accuracy of the classifier, we can use it to predict the types of glass (classes) for the training data, and see what proportion of the predictions matched the correct answers:

In [ ]:
```
train_score = classifier.score(train_vals, train_class)
print("Training score is:", train_score)
```

The classifier can choose between 7 different glass types, so anything above 1/7 (or 14.29%) is better than chance. The classifier should be performing much better than 14% on the training dataset we scored above.

But the real evaluation of the classifier is to test its accuracy in predicting the glass classes for data it has never seen before (the test dataset):

In [ ]:
```
test_score = classifier.score(test_vals, test_class)
print("Test score is:", test_score)
```

Exercise 8: Compare your test score with two other people in the class.

- Did you get the same accuracy values? If not, why do you think this might be the case? Explain briefly as a comment below:
- Is the accuracy of your classifier better or worse for the test data or the training data? Why do you think this is?

In [ ]:
```

```

Lab 6 Bonus exercises

Comparing different classifiers: LDA vs K-Nearest Neighbors

In class we discussed Nearest Neighbor classification, which classifies an unknown data point as belonging to the same class as its nearest neighboring point. K-Nearest Neighbors extrapolates this approach to more than one neighbor. For example, a 3-nearest neighbor classification looks at the three nearest data points, and picks the most common class from among those three neighbors.

In `sklearn` a KNN classifier can be imported from the `sklearn.neighbors` module:

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
```

Bonus Exercise 1: Evaluate the accuracy of `KNeighborsClassifier()` using the same training and test dataset as used previously in the lab. Is this classification method better than LDA for this dataset?

In [ ]:
```
```

Bonus Exercise 2: Look up the documentation for `sklearn KNeighborsClassifier()` to find out how to specify the number of neighbors to use. Make a scatterplot showing the accuracy of the classifier on the test dataset for a range of n-nearest neighbors (1 to 50). How does the number of nearest neighbors affect classifier accuracy?

In [ ]:
```
```

In [ ]:
```
```