CS3920/CS5920 Lab Worksheet 2: Machine learning and Nearest Neighbours

Volodya Vovk

October 5, 2019

This lab worksheet should be completed during the lab session of 10 October and your independent study time, and it should be demonstrated during the lab session of 17 October[1]. You should present your work in the form of a Jupyter notebook. Please demonstrate your work even if you do not finish all exercises.

The topics that are briefly covered in this worksheet are:

- Splitting a dataset into training and test sets.
- Running Nearest Neighbours.
- Evaluating the quality of your model.
- Loading a dataset from a file.

For further details of this worksheet, see [1, Chapter 1] and [3].

1 Splitting the dataset

First we split the `iris` dataset into training and test sets.

```
In[1]:
  from sklearn.datasets import load_iris
  iris = load_iris()
  from sklearn.model_selection import train_test_split
  X_train, X_test, y_train, y_test = train_test_split(iris['data'],
    iris['target'], random_state=0)
```

The function `train_test_split` shuffles the dataset and splits it for you. It extracts 75% of the rows in the data as the training set, together with the corresponding labels for this data. The remaining 25% of the data, together with the remaining labels are declared as the test set. How much data you want to put into the training and the test set respectively is somewhat arbitrary, but using a test set containing 25% of the data is a good rule of thumb.

[1] The day release students and other students having their lab on Monday should subtract 3 days from these dates.

It is important that the `train_test_split` function shuffles the dataset using a pseudo random number generator before making the split. If we just took the last 25% of the data as a test set, all the data point would have the label 2, as the data points are sorted by the label (as we saw in Lab 1).

To make sure that we will get the same output if we run the same function several times, we provide the pseudo random number generator with a fixed seed using the `random_state` parameter. This will make the outcome deterministic, so this command will always have the same outcome.

The output of the `train_test_split` function is `X_train`, `X_test`, `y_train`, and `y_test`, which are all `NumPy` arrays. `X_train` contains 75% of the rows of the dataset, and `X_test` contains the remaining 25%:

```
In[2]:
  X_train.shape
Out[2]:
  (112, 4)
In[3]:
  X_test.shape
Out[3]:
  (38, 4)
```

To save space, I will sometimes use the function `print`:

```
In[4]:
  print(X_train.shape)
  print(X_test.shape)

  (112, 4)
  (38, 4)
```

The empty space indicates that now Jupyter notebook's output follows. We can save even more space:

```
In[5]:
  print(X_train.shape,X_test.shape)

  (112, 4) (38, 4)
```

2 Building your first model

All machine learning models in `scikit-learn` are implemented in their own *classes*, which are parts of *modules*. The K Nearest Neighbours classification algorithm is implemented in the `KNeighborsClassifier` class in the `neighbors` module. (Remember to use the American spelling "neighbor" in Python code.)

Before we can use the model, we need to instantiate the class into an object. This is when we will set any parameters of the model. The single parameter of the `KNeighborsClassifier` is the number of neighbours, which we set to one:

```
In[6]:
  from sklearn.neighbors import KNeighborsClassifier
  knn = KNeighborsClassifier(n_neighbors=1)
```

The `knn` object encapsulates the algorithm to build the model from the training set, as well the algorithm to make predictions on test samples. It will also hold the information the algorithm has extracted from the training set. In the case of `KNeighborsClassifier`, it will just store the training set (since it is a transductive algorithm).

To build the model on the training set, we call the `fit` method of the `knn` object, which takes as arguments the `NumPy` array `X_train` containing the training data and the `NumPy` array `y_train` of the corresponding training labels:

```
In[7]:
  knn.fit(X_train, y_train)
Out[7]:
  KNeighborsClassifier(algorithm='auto', leaf_size=30,
         metric='minkowski', metric_params=None,
         n_jobs=1, n_neighbors=1, p=2, weights='uniform')
```

The `fit` method returns the `knn` object, so we get a string representation of our classifier. The representation shows us which parameters were used in creating the model. Nearly all of them are the default values, but you can also find `n_neighbors=1`, which is the parameter that we passed. Most models in `scikit-learn` have many parameters, but the majority of them are either speed optimizations or for very special use cases. You don't have to worry about the other parameters shown in this representation. Printing a `scikit-learn` model can yield very long strings, but don't be intimidated by these. All the important parameters will be covered in this course.

3 Making predictions

We can now make predictions using this model on new data, for which we might not know the correct labels. Imagine we found an iris in the wild with a sepal length of 5 cm, a sepal width of 2.9 cm, a petal length of 1 cm, and a petal width of 0.2 cm. What species of iris would this be? We can put this data into a `NumPy` array, again with the shape number of samples (1) times number of features (4):

```
In[8]:
  X_new = np.array([[5, 2.9, 1, 0.2]])
  X_new.shape
Out[8]:
  (1, 4)
```

Did you get a syntactic error? If so, correct it (see Lab 1). To make prediction we call the `predict` method of the `knn` object:

```
In[9]:
  prediction = knn.predict(X_new)
  print(prediction)
Out[9]:
  [0]
In[10]:
  print(iris['target_names'][prediction])
Out[10]:
  ['setosa']
```

Our model predicts that this new iris belongs to the class `0`, meaning its species is `Setosa`.

4 Evaluating the model

We can make a prediction for an iris in the test data, and compare it against its label (the known species). We can measure how well the model works by computing the accuracy, which is the fraction of flowers for which the right species was predicted:

```
In[11]:
  y_pred = knn.predict(X_test)
  np.mean(y_pred == y_test)
Out[11]:
  0.973684
```

Here `mean` is a `NumPy` function computing the mean. We can also use the `score` method of the `knn` object, which will compute the test set accuracy for us:

```
In[12]:
  knn.score(X_test,y_test)
Out[12]:
  0.973684
```

For this model, the test set accuracy is about 0.97, which means we made the right prediction for 97% of the irises in the test set. It's not unreasonable to expect our model to be correct 97% of the time for new irises. For our hobby botanist application, this high level of accuracy means that our models may be trustworthy enough to use, but there are lots of caveats, such as:

- is our training set large enough to make this kind of conclusions?
- can't it be a statistical fluke?
- is the 97% average success rate applicable to the specific iris that you have just found? Imagine we are talking not about irises and their species but you and a label you really care about (whether you find a job, whether you have a serious disease,...).

Partial answers to these questions are provided by conformal prediction (the topic of Chapter 3).

Full procedure

Here is a summary of the code needed for the whole training and evaluation procedure:

```
In[13]:
  X_train, X_test, y_train, y_test = train_test_split(iris['data'],
      iris['target'], random_state=0)
  knn = KNeighborsClassifier(n_neighbors=1)
  knn.fit(X_train, y_train)
  knn.score(X_test, y_test)
Out[13]:
  0.973684
```

This snippet contains the core code for applying any machine learning algorithm using `scikit-learn`. The `fit`, `predict`, and `score` methods are the common interface to supervised models in `scikit-learn`. You can apply them to many machine learning tasks using many machine learning algorithms, as you will see later in this course.

5 Loading data from file

Have a look at the `iris_data.txt` and `iris_target.txt` files on the course's Moodle page. In this section we will learn how you can load the data in `iris` from the former file. This will be important for you in future when you will have to work with datasets that are not necessarily already available in `scikit-learn` (such as `ionosphere` in Assignment 1).

Put `iris_data.txt` in your current folder. Read it into Python:

```
In[14]:
  X = np.genfromtxt("iris_data.txt")
  X[:3,]
```

`genfromtxt` is a very powerful function. You will learn more about using this function in the next lab, but if you get a chance please read about it beforehand in [2].

6 Exercises

The following exercises involve the `iris` dataset.

1. Briefly explain the way

   ```
   np.mean(y_pred == y_test)
   ```

   is computed (see `In[11]` above). It might help to run its part:

   ```
   y_pred == y_test
   ```

You may need to make an educated guess.

2. Draw the test error rate of the K Nearest Neighbours algorithm on the same training set against K. Use the same test set for all K.

   If you need to remember the scores you are getting for various K, you may use `NumPy` commands such as

   ```
   results = np.empty(19)
   ```

   (creates a `NumPy` array of size 19 with random initial values) or

   ```
   results = np.zeros(19)
   ```

   (creates an array of size 19 containing all 0s). Whereas `np.empty` is marginally faster, `np.zeros` is safer.

3. Check that the dataset that you loaded from file in Section 5 is identical to the one that you loaded using `load_iris` in Section 1. You may want to use your answer to Exercise 1. If the two data sets are not identical, please explore the difference.

If you have completed all exercises before the end of the lab session, you should demonstrate them (as part of your Jupyter notebook) and the rest of your code to a lab supervisor and may then leave early.

References

[1] Andreas C. Müller and Sarah Guido. *An introduction to machine learning with Python*. O'Reilly, Beijing, 2017.

[2] `NumPy` manual. `https://docs.scipy.org/doc/numpy/`, 2018.

[3] `scikit-learn` tutorials. `http://scikit-learn.org/stable/tutorial/`, 2017.