CS3920/CS5920 Lab Worksheet 4: General principles of machine learning and the validity of a primitive conformal predictor

Volodymyr Vovk

October 22, 2022

This lab worksheet should be completed during the lab session of 24 October and your independent study time.

To maximally benefit from these lab sessions, it's important that you write the code yourself rather than copying and pasting. One strategy is to read a section, or a significant part of a section, and after you completely understand it, try reproducing the code from your memory. It's OK if your code is slightly, or even significantly, different from what you saw; writing your own code is much more useful than seeing somebody else's.

The topics that are briefly covered in this worksheet are:

- Nearest Neighbours Regression.
- Inverted U-shaped curves.
- Learning curves.
- The optional part concerns the validity of a simple conformal predictor.

For further details of some functions used in this worksheet, see [1, Chapters 1 and 2], [3], and [2].

1 Nearest Neighbours Regression and an inverted U shape

In Lab 2 we saw how the K Nearest Neighbours classifier works for the `iris` dataset. In Exercise 2 you saw that the error rate of the K Nearest Neighbours classifier behaved somewhat anomalously. Normally we expect a U-shaped curve similar to the green curve on slide 6 of Chapter 4 (but flipped over). But the curve in Exercise 2 of Lab 2 was degenerate: the best error rate was attained already for $K = 1$. We will get a less degenerate figure for a new dataset, which will be a regression rather than classification dataset.

Namely, we will use the real-world **diabetes** dataset that has 442 samples and 10 features. The task associated with this dataset is to predict a quantitative measure of disease progression after one year, using age, sex, body mass index, average blood pressure, and six blood serum measurements as features. Let us check that there are 442 samples described by 10 features:

```
In[1]:
  from sklearn.datasets import load_diabetes
  diabetes = load_diabetes()
  diabetes.data.shape
Out[1]:
  (442, 10)
In[2]:
  print(diabetes.DESCR)
  [read quickly the description
  (read it in detail at home)]
```

Let us now split the `diabetes` dataset into training and test sets, for `K_max` values of K starting from 3 create a K Nearest Neighbours regressor, evaluate the accuracy of each of these regressors, and plot the accuracy vs K:

```
In[3]:
  from sklearn.model_selection import train_test_split
  X_train, X_test, y_train, y_test = train_test_split(diabetes['data'],
    diabetes['target'], random_state=42)
  from sklearn.neighbors import KNeighborsRegressor
  K_max = 100  # maximal number of nearest neighbours to consider
  import numpy as np
  results = np.empty(K_max)
  for k in range(K_max):
    knn = KNeighborsRegressor(n_neighbors=k+3)
    knn.fit(X_train, y_train)
    results[k] = knn.score(X_test, y_test)
  %matplotlib inline
  import matplotlib.pyplot as plt
  plt.plot(np.arange(K_max)+3,results)
```

Can you see an inverted U shape now?

As you can see in `In[3]`, the accuracy of a `KNeighborsRegressor` is still assessed using a method called `score`, as in the case of a `KNeighborsClassifier`. However, the meaning of `score` is now different. Remember that in the case of a `KNeighborsClassifier`, the function `score` reported one minus the test error rate. This is the standard meaning of `score` for classifiers. The accuracy of a `KNeighborsRegressor` is measured in `scikit-learn` by R^2, which will be defined and discussed in detail in Chapter 5. Have a quick look at its definition now:

```
In[4]:
  help(KNeighborsRegressor.score)
  [read the description quickly]
```

But do not worry if the definition is not 100% clear.

In future lab sessions we will encounter many other classifiers and regressors, and each of them will have its own `score` method. For all classifiers `score` returns one minus the test error rate, and for all regressors `score` returns R^2.

2 Using cross-validation to get an inverted U-shaped curve

To get a slightly nicer U-shaped curve we can use cross-validation. The function for cross validation in the case of regression is similar to what you saw in Chapter 4 for the case of classification:

```
In[5]:
  from sklearn.model_selection import cross_val_score
  knn = KNeighborsRegressor(n_neighbors=3)
  cross_val_score(knn, X_train, y_train)
```

You should be aware of a pitfall of `scikit-learn`'s implementation of cross-validation: the dataset on which you do cross-validation is split into folds in a systematic manner rather than randomly, and this may lead to counterintuitive results unless you shuffle the dataset randomly. In `In[5]`, there was no need to perform shuffling since it has already been performed by the function `train_test_split` that created `X_train` and `y_train`.

The function `shuffle` in `scikit-learn` performs random shuffling of several arrays of the same length; each of those arrays is shuffled in the same way.

```
In[6]:
  from sklearn.utils import shuffle
  X, y = shuffle(diabetes.data, diabetes.target, random_state=42)
  print(cross_val_score(knn, X, y))

  [0.365 0.323 0.267 0.432 0.406]
```

The best (i.e., largest) possible value for R^2 is 1, so we can see that the performance of Nearest Neighbour is less than ideal. We get similar numbers for its performance on a test set:

```
In[7]:
  knn.fit(X_train, y_train)
  knn.score(X_test, y_test)

  0.372
```

Now let us draw another inverted U-shaped curve, this time using cross-validation:

```
In[8]:
  K_max = 100  # maximal number of nearest neighbours to consider
  for k in range(K_max):
    knn = KNeighborsRegressor(n_neighbors=k+3)
    results[k] = np.mean(cross_val_score(knn, X, y))
  plt.plot(np.arange(K_max)+3,results)
```

Can you see an inverted U shape now?

Exercise 1. *Explain the role of the* `np.mean` *function in* `In[8]`*.*

Exercise 2. *Try different ranges for the numbers of nearest neighbours K in* `In[3]` *and* `In[8]`*. What is the optimal value of the number K of nearest neighbours? Compare the behaviour of the two inverted U-shaped curves for large values of K.*

3 Learning curves

Remember the definition of a learning curve: the prediction accuracy on a large test set of the model trained on a training set of size n as function of n.

```
In[9]:
  knn = KNeighborsRegressor(n_neighbors=10)
  train_sizes = np.array([50,100,200,300]) # sizes of training sets
                                           # that we will try
  results = np.empty(train_sizes.size)
  for k in range(train_sizes.size):
    X_train, X_test, y_train, y_test = train_test_split(diabetes.data,
      diabetes.target, train_size=train_sizes[k], random_state=42)
    knn.fit(X_train, y_train)
    results[k] = knn.score(X_test, y_test)
  plt.plot(train_sizes,results)
```

Explain what you see (there is no need to write down your explanation). The argument `train_size` of the function `train_test_split` specifies the size of the training set, to be used in place of the default 3 : 1 split. Notice the alternative way to access `diabetes['data']` and `diabetes['target']` used in the code snippet.

Exercise 3. *Try other arrays of* `train_sizes` *in* `In[9]`*. Write one sentence summarizing what you see.*

Exercise 4. *What is the size of the test set that you obtain when using the function* `train_test_split` *if you only specify the parameter* `train_size` *and do not specify* `test_size`*? What happens if you only specify* `test_size` *and do not specify* `train_size`*? Can you specify both? To make sure your answers are correct you should use the scikit-learn documentation.*

4 Value at Risk (optional)

This section is optional and will be of particular interest to students implementing a conformal predictor for Assignment 1. We will implement another conformal predictor (a very different one, since now our learning problem is regression rather than classification).

Suppose we work for a bank and our task is each morning to estimate our bank's Value at Risk (VaR). The bank's preferred confidence level is 80%, which means that the probability that the bank's loss on a given day exceeds the VaR that you produce in the morning of that day should be 20% or less. In the morning of day $n+1$ you are only given the losses $L_1, \ldots, L_n$ for the previous n days.

Remark 1. Computing VaR is an important area in practical finance, but in real life banks have lots of other information in addition to $L_1, \ldots, L_n$. This other information is so important that the primitive conformal predictor described in this section is useless in practice (unless it is extended to take account of the extra information).

To develop a conformal predictor for our problem, consider different possible values for L_{n+1}. Suppose, for simplicity, that the losses in the sequence $L_1, \ldots, L_n, L_{n+1}$ are all different.

- As the nonconformity scores we will take the losses themselves: $\alpha_i = L_i$ for all i.
- Rank all losses (= nonconformity scores) according to their value: the largest loss among $L_1, \ldots, L_{n+1}$ gets rank 1, the second largest gets rank 2, etc.
- Notice that the p-value is the rank of L_{n+1} divided by $n+1$.
- Deduce that the prediction set $\Gamma^{20\%} = \Gamma^{1/5}$ is the semi-infinite interval $(-\infty, L]$, where L is the smallest element of $\{L_1, \ldots, L_n\}$ whose rank in this set is $(n+1)/5$ or less. (If this appears too difficult in general, do this in the special case where $n = 99$.)

The conclusion is that the VaR in the morning of the $(n+1)$st day is the smallest element of $\{L_1, \ldots, L_n\}$ whose rank in this set is $(n+1)/5$ or less. This can be computed as follows.

First we simulate the bank losses.

```
In[10]:
  n = 99    # we are in the morning of the 100th day
  L = 10**6 * np.random.random((n)) - 10**6/2
  print(L)
```

The `NumPy` function `np.random.random` (i.e., the function `random` in the `random` module) generates independent random numbers in the interval $[0, 1)$. We make the bank's losses more respectable by multiplying them by £1,000,000 and acknowledge the fact that the bank does not suffer losses every day (the losses are negative on the days when the bank's operations are profitable) by subtracting £500,000.

Now we can compute the VaR; for simplicity let's use the very powerful function `np.sort` sorting the array.

```
In[11]:
  sorted_L = np.sort(L)
  VaR = sorted_L[-int(np.floor((n+1)/5))]
  print(VaR)
```

Notice the minus sign; it's needed because **np.sort** sorts the array in the ascending order. The NumPy function **floor** computes the floor of a real number (the *floor* of x is the largest integer i such that $i \leq x$, usually denoted as $\lfloor x \rfloor$):

```
In[12]:
  np.floor(4.5)

  4.0
```

Notice that **np.floor(4.5)** is still a float, so we have to apply **int()** in order to be able to use it for indexing.

Let's package this way of computing VaR as a function:

```
In[13]:
  import math
  def VaR(L):
    """Value at Risk (a primitive conformal predictor)"""
    if L.size>=4:
      return np.sort(L)[-int(np.floor((n+1)/5))]
    else:
      return math.inf
```

This definition uses the fact that we can obtain a p-value of 20% only if the training set contains at least 4 samples; if the number of samples is below 4, we have to output the vacuous VaR of ∞.

Validity of conformal prediction: an empirical test

In this subsection we will check empirically the validity of our conformal predictor in the online mode.

Suppose we have been working for our bank for 2 years, or 500 working days, and the losses were as follows:

```
In[14]:
  N = 500    # the number of days
  # The losses on those days:
  L = 10**6 * np.random.random_sample((N)) - 10**6/2
```

Our performance over those 500 days is described by the following snippet:

```
In[15]:
  successes = np.empty(N)
  for n in range(N):
    # This is what happens on the (n+1)st day:
    V = VaR(L[:n])  # VaR for the (n+1)st day
    if L[n]<=V:
      successes[n] = 1  # our VaR worked
    else:
      successes[n] = 0  # it didn't
  print(np.mean(successes))

  0.784
```

We create an array `successes` recording our success; an entry of 1 in it means that our VaR was successful (it was a valid upper bound on the actual loss on that day) and an entry of 0 means that our VaR was exceeded. In the end `successes` contains a full record of our successes and failures.

We can see that the percentage of successful days (i.e., the days when our VaR was correct) is 78.4%, which is fairly close to our nominal confidence level 80%. (And if you generate another sequence L of losses, you will get another percentage, still close to 80%.)

What about the intermediate days? For example, if your boss checks the percentage of your successes by day 300, will it still be close to 80%?

```
In[16]:
  plt.plot(np.arange(N)+1,np.cumsum(successes))
```

This graph plots the number of successes by day n vs n. It is close to the linear function with slope 80%.

Exercise 5. *What exactly is the function `np.cumsum` ("cumulative sum") doing in `In[16]`? Check [2] or Google for it. (Being in an optional section, this exercise is also optional.)*

5 One more exercise

Exercise 6. *Plot a learning curve for the `iris` dataset.*

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/`, 2008–2022.

[3] `scikit-learn` tutorials. `http://scikit-learn.org/stable/tutorial/`, 2007–2022.