CS3920/CS5920 Lab Worksheet 7: Kernel methods in `scikit-learn`

Volodymyr Vovk

November 7, 2022

This lab worksheet should be completed during the lab session of 14 November and your independent study time.

The topics that are briefly covered in this worksheet are:

- Nearest Neighbours with user-defined distances.
- Kernel methods in combination with Nearest Neighbours.
- Creating your own estimator.
- Uncertainty estimates for Nearest Neighbours.

For further details of some functions used in this worksheet, see [1, Chapters 2 and 8] and [2].

If you are getting results that are not exactly the same as given below, usually there is no need to worry: you may be using a different version of `scikit-learn`.

1 Nearest Neighbours with user-defined distances

In this section we will be using the `ionosphere` dataset, the one that you used in Assignment 1.

```
In[1]:
  import numpy as np
  from sklearn.model_selection import train_test_split
  X = np.genfromtxt("ionosphere.txt", delimiter=",",
    usecols=np.arange(34))
  y = np.genfromtxt("ionosphere.txt", delimiter=",",
    usecols=34, dtype='int')
  X_train, X_test, y_train, y_test = train_test_split(X, y,
    random_state=0)
```

Let us fit a default Nearest Neighbour algorithm to it:

```
In[2]:
  from sklearn.neighbors import KNeighborsClassifier
  knn = KNeighborsClassifier(n_neighbors=1)
  knn.fit(X_train, y_train)

  KNeighborsClassifier(algorithm='auto', leaf_size=30,
    metric='minkowski', metric_params=None, n_jobs=1,
    n_neighbors=1, p=2, weights='uniform')
```

You have already seen such a description of a Nearest Neighbour model, but now we will discuss two further parameters, `metric='minkowski'` and `p=2`.

Hermann Minkowski (1864–1909) was a German mathematician, and his name is sometimes attached to the family of vector norms

$$\|w\|_{\mathbf{p}} = \left(\sum_{j=0}^{p-1} |w[j]|^{\mathbf{p}} \right)^{1/\mathbf{p}} ,$$

where $\mathbf{p}$ is a positive parameter. They are also known as $L_{\mathbf{p}}$ norms, and you saw two examples in Chapter 5: L_2 (used in Ridge Regression) and L_1 (used in Lasso). The $L_{\mathbf{p}}$ norm defines the $L_{\mathbf{p}}$ distance

$$d(u, v) = \|u - v\|_{\mathbf{p}} .$$

The case $\mathbf{p} = 2$ corresponds to Euclidean distance, or simply distance, and is the default in `scikit-learn`: see `In[2]`.

Remark 1. Remember that the parameter $\mathbf{p}$ (which is set in boldface to make it less confusing) has nothing to do with the number p of features.

To specify a user-defined metric, we use the option `metric` setting it to our own function for computing the metric:

```
In[3]:
  def my_dist(x, y):
    return np.sum((x-y)**2)
  knn = KNeighborsClassifier(n_neighbors=1, metric=my_dist)
  knn.fit(X_train, y_train)

  KNeighborsClassifier(algorithm='auto', leaf_size=30,
    metric=<function my_dist at 0x0000017587939C80>, metric_params=None,
    n_jobs=1, n_neighbors=1, p=2, weights='uniform')
```

In this example `my_dist` is just squared Euclidean distance, and so we obtain the same accuracy as before:

```
In[4]:
  np.mean(knn.predict(X_test)==y_test)

  0.852273
```

But using the Manhattan metric L_1 gives a better result:

```
In[5]:
  knn = KNeighborsClassifier(n_neighbors=1, p=1)
  knn.fit(X_train, y_train)
  np.mean(knn.predict(X_test)==y_test)

0.920455
```

Exercise 1. *Use cross-validation on the training set to choose the best value of* **p** *for Nearest Neighbour. What is the error of the Nearest Neighbour with this value of* **p** *on the test set?*

2 Kernel methods

Let us now try to use distances defined by kernels. For the polynomial kernel of degree 2 we get:

```
In[6]:
  def poly_kernel(x, y, d):
    return (1+np.dot(x,y))**d
  d = 2  # trying the polynomial kernel of degree d
  def poly_dist(x, y):  # squared distance
    return poly_kernel(x,x,d) + poly_kernel(y,y,d)\
      - 2*poly_kernel(x,y,d)
  knn = KNeighborsClassifier(n_neighbors=1, metric=poly_dist)
  knn.fit(X_train, y_train)
  np.mean(knn.predict(X_test)==y_test)

  0.886364
```

(Notice the use of a backslash \ to split a line of code in Python.) The result is intermediate between the L_2 and L_1 metric. The rbf kernel gives a similar result:

```
In[7]:
  def rbf_kernel(x, y, gamma):
    return np.exp(-gamma*np.sum((x-y)**2))
  gamma = 1  # the parameter gamma of the rbf kernel
  def rbf_dist(x, y):  # squared distance
    return rbf_kernel(x,x,gamma) + rbf_kernel(y,y,gamma)\
      - 2*rbf_kernel(x,y,gamma)
  knn = KNeighborsClassifier(n_neighbors=1, metric=rbf_dist)
  knn.fit(X_train, y_train)
  np.mean(knn.predict(X_test)==y_test)

  0.852273
```

How good is our chosen value $\gamma = 1$ of the parameter γ for the rbf kernel? To answer this question, let's use parameter selection with cross-validation, as in Lab Worksheet 6.

```
In[8]:
  from sklearn.model_selection import cross_val_score
  best_score = 0
  for gamma in [0.01, 0.1, 1, 10, 100]:
    # for each parameter, train a model
    def rbf_dist(x, y):  # squared distance
      return rbf_kernel(x,x,gamma) + rbf_kernel(y,y,gamma)\
        - 2*rbf_kernel(x,y,gamma)
    knn = KNeighborsClassifier(n_neighbors=1, metric=rbf_dist)
    # perform cross-validation
    scores = cross_val_score(knn, X_train, y_train, cv=5)
    # compute mean cross-validation accuracy
    score = np.mean(scores)
    # if we got a better score, store the score and parameters
    if score > best_score:
      best_score = score
      best_gamma = gamma
  # rebuild a model on the full training set
  def rbf_dist(x, y):  # squared distance
    return rbf_kernel(x,x,best_gamma) + rbf_kernel(y,y,best_gamma)\
      - 2*rbf_kernel(x,y,best_gamma)
  knn = KNeighborsClassifier(n_neighbors=1, metric=rbf_dist)
  knn.fit(X_train, y_train)
  test_score = knn.score(X_test, y_test)
  print("Best CV score:", best_score)
  print("Best parameter gamma:", best_gamma)
  print("Test set score with best parameters:", test_score)

  Best CV score: 0.844340
  Best parameter gamma: 0.01
  Test set score with best parameters: 0.852273
```

We can see that the K Nearest Neighbours algorithm with the rbf kernel does not perform better on this dataset.

3 Creating your own estimator

In this section we will create our own estimator, Kernel K Nearest Neighbours. We will obtain it by slightly modifying `KNeighborsClassifier`.

Our very first estimator is not particularly original:

```
In[9]:
  class My_Classifier(KNeighborsClassifier):
    """My first example of a classifier"""
    def __init__(self, n_neighbors=1):
      KNeighborsClassifier.__init__(self, n_neighbors=n_neighbors)
    def fit(self, X, y):
      KNeighborsClassifier.fit(self, X, y)
      return self
    def predict(self, X, y=None):
      return KNeighborsClassifier.predict(self, X)
    def score(self, X, y):
      return KNeighborsClassifier.score(self, X, y)
```

We create a subclass of `KNeighborsClassifier` with essentially the same methods `__init__`, `fit`, and `predict`. The only difference is that now the default value for the number of nearest neighbours used is `n_neighbors=1`, whereas it was `n_neighbors=5` for `KNeighborsClassifier`.

Let's test our new classifier:

```
In[10]:
  knn = My_Classifier()
  knn.fit(X_train, y_train)
  knn.score(X_test, y_test)

  0.852273
```

We get the same result as before.

Let us now modify the classifier by replacing Euclidean distance by the rbf distance.

```
In[11]:
  class rbfClassifier(KNeighborsClassifier):
    """Kernel K Nearest Neighbours classifier"""
    def __init__(self, n_neighbors=1, gamma=1):
      def rbf_dist(x, y):  # squared distance
        return rbf_kernel(x,x,gamma) + rbf_kernel(y,y,gamma)\
          - 2*rbf_kernel(x,y,gamma)
      KNeighborsClassifier.__init__(self, n_neighbors=n_neighbors,
        metric=rbf_dist)
      self.gamma = gamma
      self.n_neighbors=n_neighbors
    def fit(self, X, y):
      KNeighborsClassifier.fit(self, X, y)
      return self
    def predict(self, X, y=None):
      return KNeighborsClassifier.predict(self, X)
    def score(self, X, y):
      return KNeighborsClassifier.score(self, X, y)
```

We can use our new estimator in the same way as `KNeighborsClassifier`.

```
In[12]:
  knn = rbfClassifier()
  knn.fit(X_train, y_train)
  knn.score(X_test,y_test)

  0.852273
```

Unfortunately, string kernels are not implemented in `scikit-learn`.

4 Uncertainty estimates for Nearest Neighbours

Remember that the method `predict` gives predicted labels of the test set:

```
In[13]:
  from sklearn.datasets import load_iris
  iris = load_iris()
```

```
X_train, X_test, y_train, y_test = train_test_split(iris.data,
  iris.target, random_state=0)
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
knn.predict(X_test)

array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       2, 1, 0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 2])
```

(To make the prediction problem more interesting we are now using a multiclass dataset, `iris`.) There is a useful modification of this method that outputs predicted probabilities for the labels:

```
In[14]:
  knn.predict_proba(X_test)

  array([[0. , 0. , 1. ],
         [0. , 1. , 0. ],
         [1. , 0. , 0. ],
         [0. , 0. , 1. ],
         [1. , 0. , 0. ],
         [0. , 0. , 1. ],
         [1. , 0. , 0. ],
         [0. , 1. , 0. ],
         [0. , 1. , 0. ],
         [0. , 1. , 0. ],
         [0. , 0. , 1. ],
         [0. , 1. , 0. ],
         [0. , 1. , 0. ],
         [0. , 1. , 0. ],
         [0. , 0.6, 0.4],
         [1. , 0. , 0. ],
         [0. , 0.8, 0.2],
         [0. , 1. , 0. ],
         [1. , 0. , 0. ],
         [1. , 0. , 0. ],
         [0. , 0. , 1. ],
         [0. , 1. , 0. ],
         [1. , 0. , 0. ],
         [1. , 0. , 0. ],
         [0. , 0.2, 0.8],
         [1. , 0. , 0. ],
         [1. , 0. , 0. ],
         [0. , 1. , 0. ],
         [0. , 1. , 0. ],
         [1. , 0. , 0. ],
         [0. , 0. , 1. ],
         [0. , 1. , 0. ],
         [1. , 0. , 0. ],
         [0. , 0.2, 0.8],
         [0. , 0. , 1. ],
         [0. , 1. , 0. ],
         [1. , 0. , 0. ],
         [0. , 0. , 1. ]])
```

The probability of a class (0, 1, or 2) is defined as the percentage of this class among the 5 nearest neighbors (remember that the default value of `n_neighbors` in `scikit-learn` is 5).

5 More exercises

As usual, all your answers should be written in your Jupyter notebook.

Exercise 2. *Explain how the array in `In[13]` can be obtained from the array in `In[14]`.*

Exercise 3. *Implement a new method, called* `predict_proba`*, for the class* `rbfClassifier`*, which should output probabilities for various labels for the test samples.* HINT: *Emulate what we did for the method* `predict` *in that class.*

Exercise 4. *Test your new method* `predict_proba` *for the class* `rbfClassifier`*.*

References

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

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