Exercise 1b

Deadline: 17.11.2021, 16:00.
Ask questions to #ask-your-tutor-peter

In this exercise, you will implement the k-nearest-neighbor classifier and estimate its accuracy on unseen test data and via cross-validation. Moreover, you learn how to speed-up Python code using "vectorization".

Regulations

Please read this section! Instructions are slightly different to last time.

Solutions for this week's tasks shall be handed in as `nearest-neighbor.ipynb` and exported to `nearest-neighbor.html`. Zip all files into a single archive `ex01b.zip` and upload this file to your assigned tutor on MaMPF before the given deadline.
Note: Each team creates only a single upload, and all team members must *join* it as described in the MaMPF documentation at `https://mampf.blog/zettelabgaben-fur-studierende/`.
Important: Make sure that your MaMPF name is the same as your name on Muesli. We now identify submissions purely from the MaMPF name. If we are unable to identify your submission you will not receive points for the exercise!

1 Nearest Neighbor Classification on Real Data

1.1 Exploring the Data (3 points)

Scikit-learn (usually abbreviated `sklearn`) provides a collection of standard datasets that are suitable for testing a classification algorithm (see `https://scikit-learn.org/stable/datasets.html` for a list of the available datasets and usage instructions). In this exercise, we want to recognize handwritten digits, which is a typical machine learning application. The dataset `digits` consists of 1797 small images with one digit per image.

Load the dataset from sklearn and extract the data:

```
from sklearn.datasets import load_digits

digits = load_digits()

print(digits.keys())

data         = digits["data"]
images       = digits["images"]
target       = digits["target"]
target_names = digits["target_names"]

print(data.dtype)
```

Note that `data` is a flattened (1-dimensional) version of `images`. What is the size of these images (the `numpy` attribute `shape` might come in handy)? Visualize one image of a **3** using the `imshow` function from `matplotlib.pyplot`, trying the two interpolation methods in the code:

```
import numpy as np
import matplotlib.pyplot as plt

img = ...

assert 2 == len(img.shape)
```

```
plt.figure()
plt.gray()
plt.imshow(img, interpolation="nearest") # also try interpolation="bicubic"
plt.show()
```

Moreover, sklearn provides a convenient function to separate the data into a training and a test set.

```
from sklearn import model_selection

X_all = data
y_all = target

X_train, X_test, y_train, y_test =\
    model_selection.train_test_split(digits.data, digits.target,
                                     test_size = 0.4, random_state = 0)
```

1.2 Distance function computation using loops (3 points)

A naive implementation of the nearest neighbor classifier uses loops to determine the required minimum distances. Implement this approach in a python function `dist_loop(training, test)`, which computes the Euclidean distance between all instances in the training and test set (in the feature space). The input should be the $N \times D$ and $M \times D$ training and test matrices with D pixels per image and N respectively M instances in the training and test set. The output should be a $N \times M$ distance matrix. For the calculation of the Euclidean distance you might want to use `numpy.square()`, `numpy.sum()` and `numpy.sqrt()` or `numpy.linalg.norm()`.

1.3 Distance function computation using vectorization (8 points)

Since loops are rather slow in python, and we will need efficient code later in the semester, write a second python function `dist_vec(training, test)` for computing the distance function which relies on vectorization and does not use `for` loops. Consult `https://www.safaribooksonline.com/library/view/python-for-data/9781449323592/ch04.html` and `https://softwareengineering.stackexchange.com/questions/254475/how-do-i-move-away-from-the-for-loop-school-of-thought` for information on how to do this. Verify that the new function returns the same distances as the loop-based version. Now compare the run times of the two implementations using jupyter's `%timeit` command (the vectorized version should be significantly faster).

Note: It is absolutely critical that you understand vectorization, because the code for subsequent homeworks will otherwise be too slow for meaningful experimentation. We will therefore grade this task very strictly.

1.4 Implement the k-nearest neighbor classifier (6 points)

Revise your code from the previous homework to implement a k-nearest neighbor classifier. It should work for arbitrary k (number of neighbors to include in the majority vote), N (training set size) and D (number of features). Use your classifier to distinguish the digit **3** from the digit **9**. To do so, filter out these digits from the training and test sets and use your function from subproblems 1.2 or 1.3 to compute distances. Vary the value for k (try the values 1, 3, 5, 9, 17 and 33) and compute the error rates. Describe the dependency of the classification performance on k.

2 Cross-validation (8 points)

Note: Please use the whole digits dataset for this part of the exercise (not just 3 and 9).

In subproblem 1.4, we measured the performance of the nearest neighbor classifier on a predefined test set. To be able to do this, we had to put aside test data and thus reduced the size of the

training set. This may lead to increased error because some relevant training instances might not end up in the training set. Another way to estimate whether the trained classifier is able to generalize to unseen data is cross-validation (see `https://en.wikipedia.org/wiki/Cross-validation_(statistics)`).

Write a function

```
X_folds, y_folds = split_folds(data, target, L)
```

to randomly split the given data and labels into L *folds* (parts of roughly equal size). The numpy-functions `numpy.random.permutation()` and `numpy.array_split()` may be useful here. In each subsequent cross-validation iteration, you use one of the L subparts as test set and the remaining ones as corresponding training set.

Use the same splits to evaluate your implementation from subtask 1.4 and the pre-defined solution in `sklearn.neighbors.KNeighborsClassifier()`. The latter is trained with the `fit()` function and classifies new data via the `predict()` function (see `http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html` for more details).

Return the mean error rate as well as its standard deviation over the L repetitions. Cross-validate your k-nearest neighbor classifier with $k = 1$ and a suitable bigger k, as well as its `sklearn` counterparts, on the full digits dataset for $L \in \{2, 5, 10\}$. Compare the algorithms' performance. How do the results depend on L?