AKADEMIA GÓRNICZO-HUTNICZA

Project Documentation

k-NN

From the course

Programming in Python Language

EiT 4

Antoni Bajor
Dawid Olchawa

Group 4, Friday 17:30

30.11.2025

Table of contents

1. Project description

The aim of this project is to design, implement and evaluate a simple k-Nearest Neighbors (k-NN) classification algorithm in Python. The main goal is to understand how the algorithm works by creating a custom implementation, rather than using only ready-made functions from external libraries.

The project focuses on the following tasks:

- implementing the k-NN algorithm from scratch as a reusable Python class
- enabling the user to choose the number of neighbours *k*
- applying the classifier to selected datasets and evaluating its performance
- comparing the custom implementation with the KNeighborsClassifier available in the scikit-learn library

The documentation presents the theoretical background of the k-NN algorithm, describes the structure of the implementation, explains how to run and use the program, and summarises the results of experiments performed on example data.

2. Theoretical background

The k-Nearest Neighbors (k-NN) algorithm is a simple, instance-based method used mainly for classification and, in some cases, regression. It belongs to the group of lazy learning algorithms, because it does not build an explicit model during the training phase. Instead, it stores the training data and performs all computations when a new sample needs to be classified.

In the classification setting, the basic idea of k-NN is as follows:

- Each data point is represented as feature vector in an n-dimensional space,
- For a new (unknown) sample, the algorithm finds the k closest training samples according to a Euclidean distance metric,
- The class label is assigned based on the majority class among these k neighbours (majority voting).

The most commonly used distance measure in k-NN is the Euclidean distance. For two point and $y = (y_1, y_2, \dots, y_n)$, the Euclidean distance is defined as:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

The choice of the parameter *k* is very important:

- a small *k* (e.g. 1 or 3) makes the classifier more flexible but also more sensitive to noise and outliers,
- a large *k* leads to smoother decision boundaries but may ignore local structure and minority classes.

The k-NN algorithm has several advantages: it is conceptually simple and easy to implement, it can achieve good performance for many practical problems, and it naturally supports multi-class classification. However, it also has some limitations: the prediction time can be high for large datasets because distances must be computed to many training points, the algorithm is sensitive to the scale of features and usually requires normalisation, and it may perform poorly in very high-dimensional spaces (the "curse of dimensionality").

3. Project environment and tools

The project was implemented in the Python programming language and follows the basic recommendations for structuring a small software project.

The main elelments of the environment are:

- Programming language: Python 3.13.5
- Development environment: PyCharm 2025.2.4
- Github repository: *https://github.com/Toniqo/kNN*

To simplify installation and isolate dependencies, a virtual environment is used: venv or VirtualEnv is employed to create a separate Python environment for the project, and all required packages can be installed using pip.

The source files are organised into a simple module structure (for example src/knn/abdo_knn.py for the classifier and separate scripts for tests and experiments). This structure allows the classifier to be reused in other scripts and makes the project easier to maintain and extend in the future.

4. Implementation

4.1 Project structure

- *src/knn/* - main Python package
    - *abdo_knn.py* - implementations of ***ABDOKNNClassifier***
- *tests/* - scripts for experiments and comparison
    - *compare_knn.py* - compares our k-NN with scikit-learn
    - *plot_knn_results.py* - plots accuracy vs k and confusion matrices

4.2 *ABDOKNNClassifier* implementation

The file *src/knn/abdo_knn.py* contains the ***ABDOKNNClassifier*** class, which implements the k-Nearest Neighbors algorithm. The constructor takes the parameter k (number of neighbours) and checks that it is greater than zero.

- The *fit(X, y)* method stores the training data and verifies that the dataset is non-empty and that the numbers of samples and labels match.
- The private method *_distance(p1, p2)* computes the Euclidean distance between two feature vectors.

- The *_predict_one(x)* method finds all distances to training samples, selects the k nearest neighbours and returns the most frequent label using majority voting.
- The public *predict(X)* method applies *_predict_one* to each sample in X and returns a list of predicted class labels.

5. Datasets

In this project two standard datasets from scikit-learn were used: ***load_digits*** and ***load_iris***. The *load_iris* dataset contains measurements of iris flowers from three different species, while the the *load_digits* dataset contains images of handwritten digits represented as numerical feature vectors. Both datasets were loaded using the built-in functions load_iris() and load_digits() from scikit-learn and then split into training and test sets to evaluate the k-NN classifier.

6. User manual

The k-NN classifier is implemented as the ABDOKNNClassifier class in src/knn/abdo_knn.py. To use it, import the class in Python, create an object with a chosen value of *k*, call fit(X_train, y_train) on the training data, and then predict(X_test) to obtain labels for new samples.

Example experiments can be run with the scripts in the tests folder: compare_knn.py compares accuracy between the custom classifier with scikit-learn, and plot_knn_results.py plots accuracy for different values of *k* and confusion matrices.

7. Results and analysis

My implementation of k-NN classifier, has appeared very similar to Scikit-learn classifier. Accuracy is mostly the same, with some differences, when changing number of neighbors.

First Test was performed on the Iris dataset. For small datasets like this, typical number of neighbors is 3 or 5. Below, there are listed accuracies for ABDO classifier and Scikit-learn version, with three decimal places of result accuracy, for 3, 5, and 8 neighbors.

Iris accuracy	**k = 3**	**k = 5**	**k = 8**
ABDO	0.956	0.978	0.933
Scikit-learn	0.956	0.978	0.933

We can see that the results are identical for both classifiers. That is completely reasonable, because they are working on the same principle, only with differences in implementation.

Next step is testing both implementations, over a range of nieghbor value. Diagram below shows accuracy in function of k, which is number of neighbors. The range is 1 to 16.

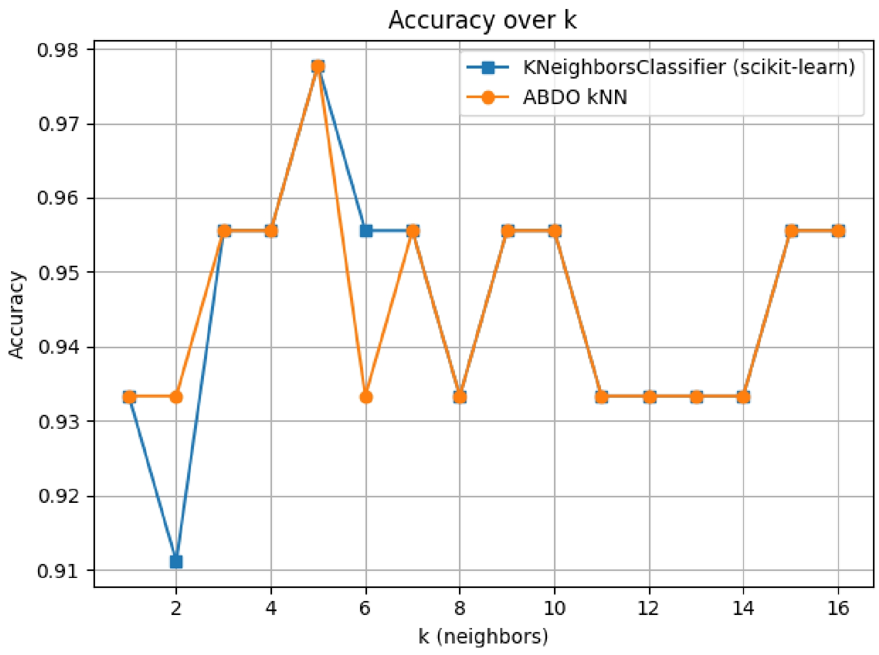
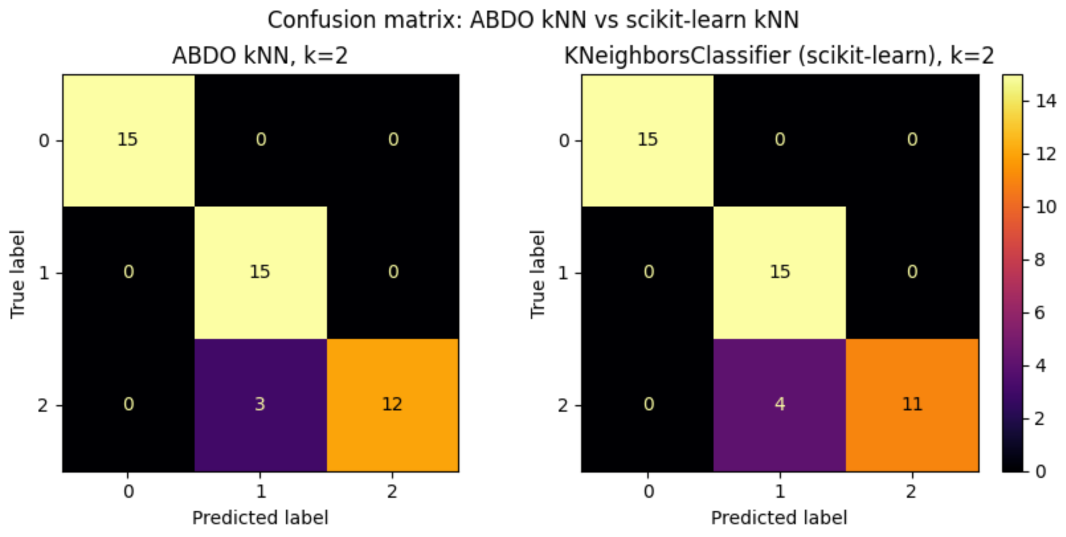


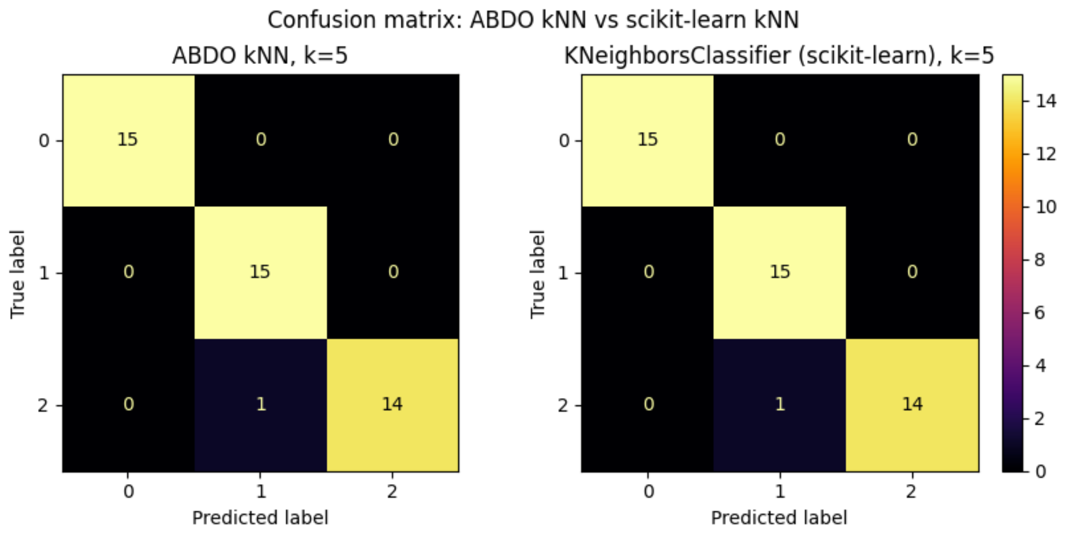
Diagram of accuracy over k for Iris dataset

The difference between classifiers is now visible for 2 and 6 neighbors. For the former our classifier appears to have better accuracy than Scikit-learn, but for the latter, our classifier has worse accuracy. The rest are the same results.

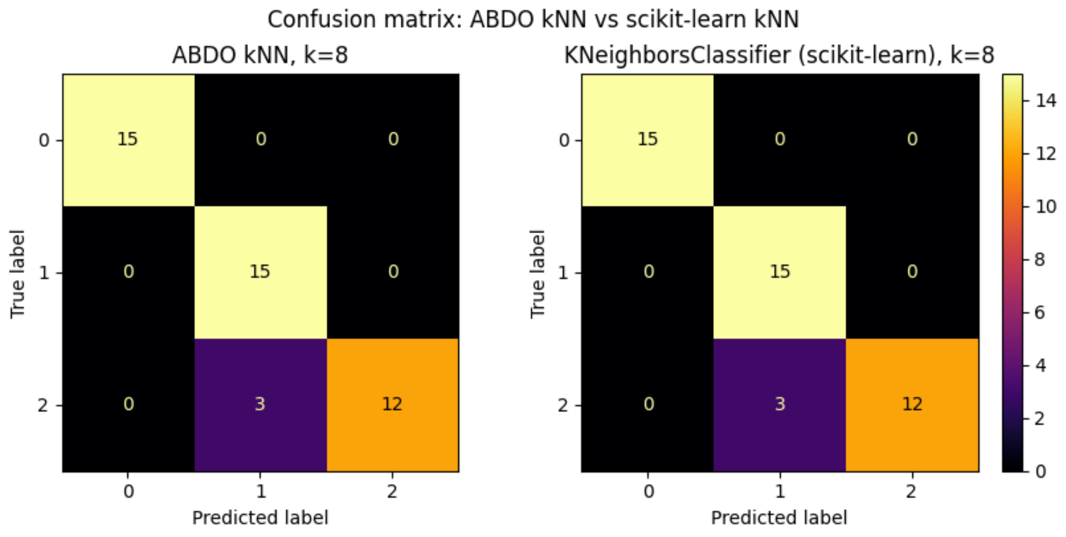
Another way to compare classifiers is confusion matrix. The X axis shows us value predicted by k-NN classifier, and the Y axis shows real value of class. The main diagonal of matrix in result, shows number of instances, that are predicted correctly.



Confusion matrices of k=2 for Iris dataset



Confusion matrices of k=5 for Iris dataset



Confusion matrices of k=8 for Iris dataset

We can see difference for k=2, where one class more was classified correctly in ABDO classifier. In Scikit-Learn version, it was falsely classified as 1, when it should be 2.

To better benchmark my implementation of k-NN classifier, the dataset was changed to Digits, then the same test were ran, with identical parameters as before.

Digits accuracy	**k = 3**	**k = 5**	**k = 8**
ABDO	0.987	0.987	0.985
Scikit-learn	0.987	0.987	0.980

Accuracies are identical except when k=8. ABDO classifier has slightly higher value.

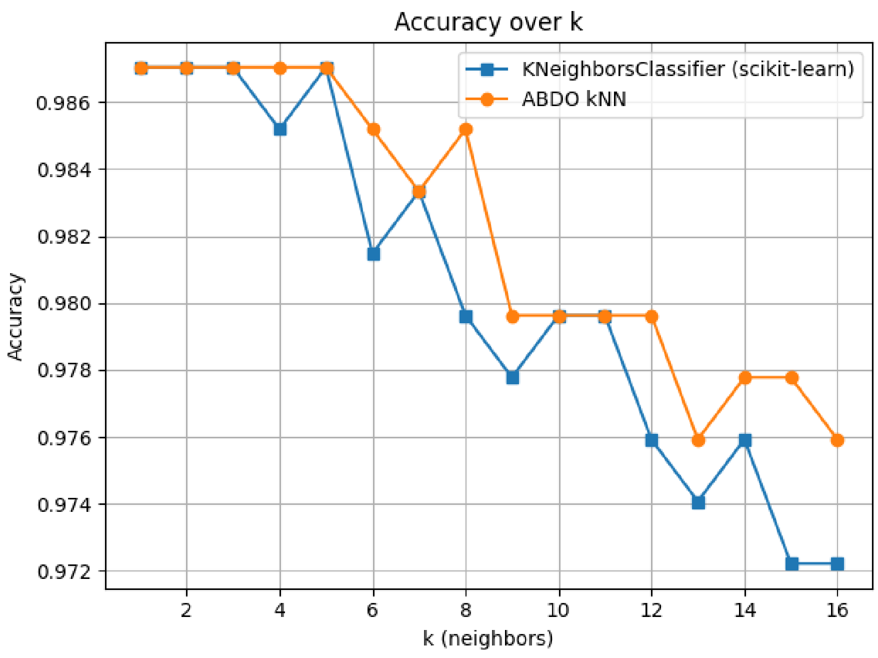
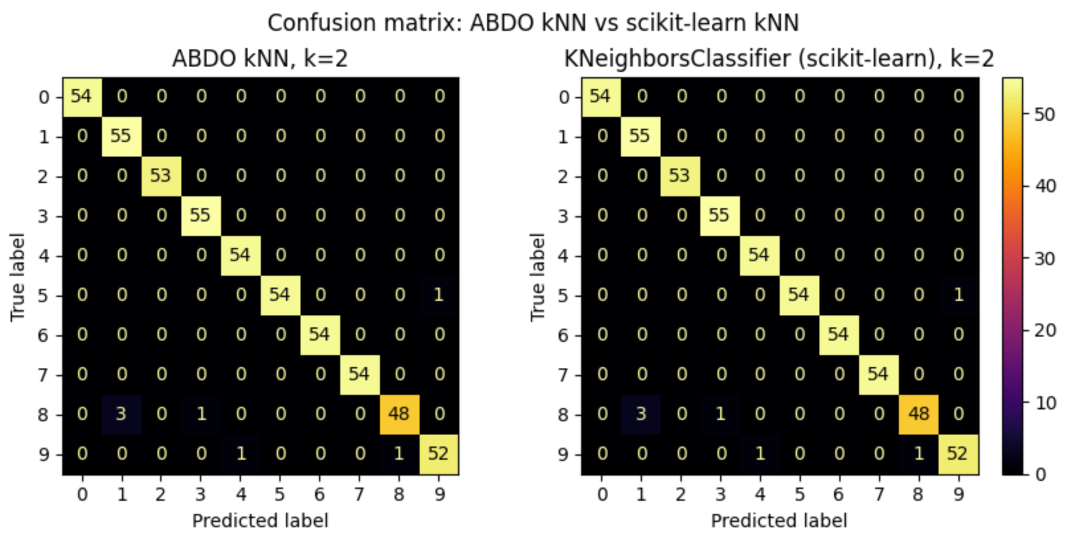


Diagram of accuracy over k for Digits dataset

On the diagram of accuracy over k, the difference is even more visible. ABDO classifier has higher accuracies half of the time. Also, we can see that the accuracy is gradually dropping in both classifiers with increasing number of neighbors.



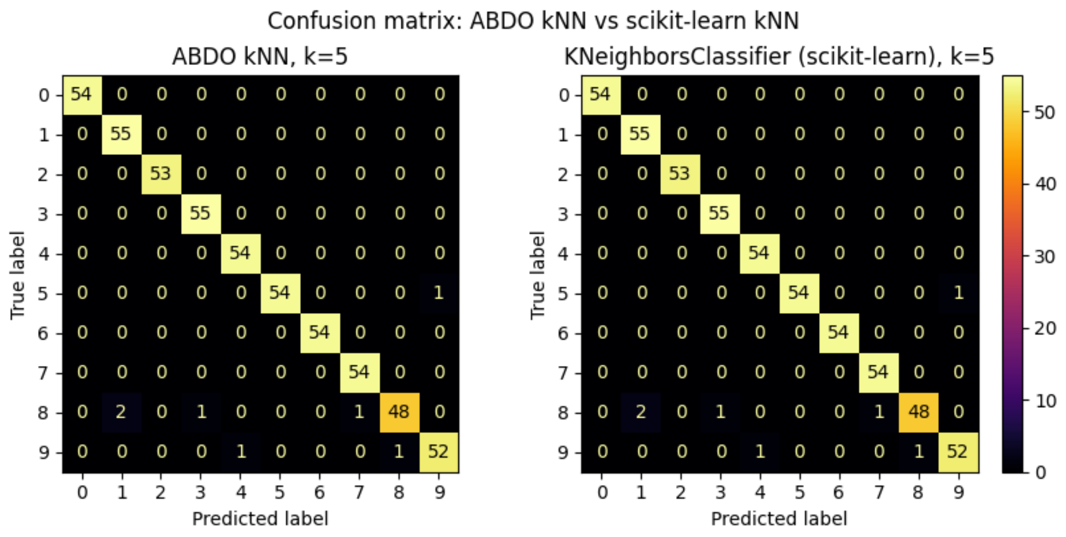
ABDO kNN, k=2

	0	1	2	3	4	5	6	7	8	9
0	54	0	0	0	0	0	0	0	0	0
1	0	55	0	0	0	0	0	0	0	0
2	0	0	53	0	0	0	0	0	0	0
3	0	0	0	55	0	0	0	0	0	0
4	0	0	0	0	54	0	0	0	0	0
5	0	0	0	0	0	54	0	0	0	1
6	0	0	0	0	0	0	54	0	0	0
7	0	0	0	0	0	0	0	54	0	0
8	0	3	0	1	0	0	0	0	48	0
9	0	0	0	0	1	0	0	0	1	52

KNeighborsClassifier (scikit-learn), k=2

	0	1	2	3	4	5	6	7	8	9
0	54	0	0	0	0	0	0	0	0	0
1	0	55	0	0	0	0	0	0	0	0
2	0	0	53	0	0	0	0	0	0	0
3	0	0	0	55	0	0	0	0	0	0
4	0	0	0	0	54	0	0	0	0	0
5	0	0	0	0	0	54	0	0	0	1
6	0	0	0	0	0	0	54	0	0	0
7	0	0	0	0	0	0	0	54	0	0
8	0	3	0	1	0	0	0	0	48	0
9	0	0	0	0	1	0	0	0	1	52

Confusion matrices of k=2 for Digits dataset



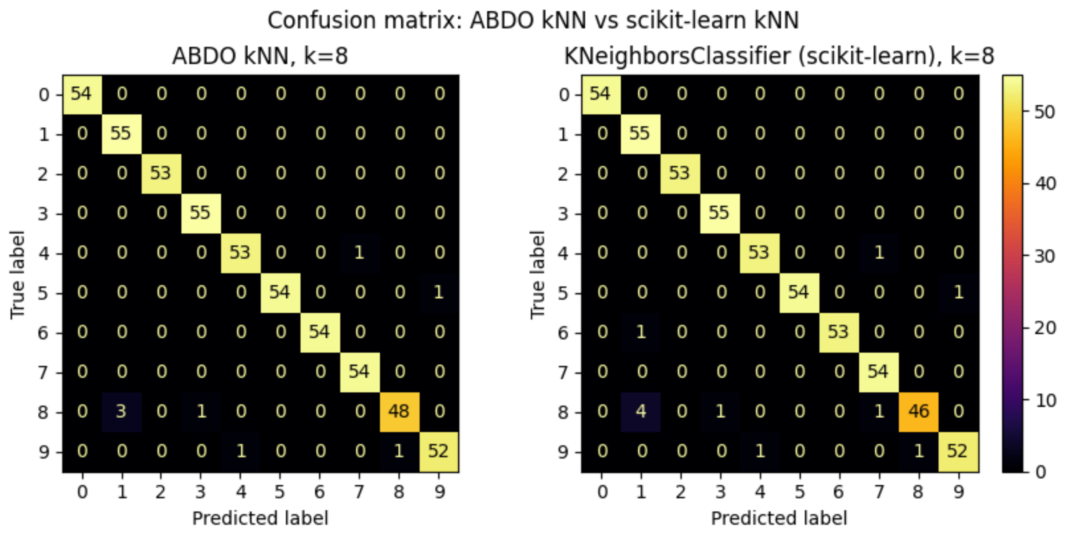
ABDO kNN, k=5

	0	1	2	3	4	5	6	7	8	9
0	54	0	0	0	0	0	0	0	0	0
1	0	55	0	0	0	0	0	0	0	0
2	0	0	53	0	0	0	0	0	0	0
3	0	0	0	55	0	0	0	0	0	0
4	0	0	0	0	54	0	0	0	0	0
5	0	0	0	0	0	54	0	0	0	1
6	0	0	0	0	0	0	54	0	0	0
7	0	0	0	0	0	0	0	54	0	0
8	0	2	0	1	0	0	0	1	48	0
9	0	0	0	0	1	0	0	0	1	52

KNeighborsClassifier (scikit-learn), k=5

	0	1	2	3	4	5	6	7	8	9
0	54	0	0	0	0	0	0	0	0	0
1	0	55	0	0	0	0	0	0	0	0
2	0	0	53	0	0	0	0	0	0	0
3	0	0	0	55	0	0	0	0	0	0
4	0	0	0	0	54	0	0	0	0	0
5	0	0	0	0	0	54	0	0	0	1
6	0	0	0	0	0	0	54	0	0	0
7	0	0	0	0	0	0	0	54	0	0
8	0	2	0	1	0	0	0	1	48	0
9	0	0	0	0	1	0	0	0	1	52

Confusion matrices of k=5 for Digits dataset



ABDO kNN, k=8

	0	1	2	3	4	5	6	7	8	9
0	54	0	0	0	0	0	0	0	0	0
1	0	55	0	0	0	0	0	0	0	0
2	0	0	53	0	0	0	0	0	0	0
3	0	0	0	55	0	0	0	0	0	0
4	0	0	0	0	53	0	0	1	0	0
5	0	0	0	0	0	54	0	0	0	1
6	0	0	0	0	0	0	54	0	0	0
7	0	0	0	0	0	0	0	54	0	0
8	0	3	0	1	0	0	0	0	48	0
9	0	0	0	0	1	0	0	0	1	52

KNeighborsClassifier (scikit-learn), k=8

	0	1	2	3	4	5	6	7	8	9
0	54	0	0	0	0	0	0	0	0	0
1	0	55	0	0	0	0	0	0	0	0
2	0	0	53	0	0	0	0	0	0	0
3	0	0	0	55	0	0	0	0	0	0
4	0	0	0	0	53	0	0	1	0	0
5	0	0	0	0	0	54	0	0	0	1
6	0	1	0	0	0	0	53	0	0	0
7	0	0	0	0	0	0	0	54	0	0
8	0	4	0	1	0	0	0	1	46	0
9	0	0	0	0	1	0	0	0	1	52

Confusion matrices of k=8 for Digits dataset

The confusion matrices now are bigger because of the dataset features. The only difference is when k=8, when ABDO classifier is slightly better.

Lastly, the computation time of both classifiers was tested on these two datasets.

Iris time	**k = 3**	**k = 5**	**k = 8**	**k = 16**
ABDO	0.00310s	0.00310s	0.00302s	0.00320s
Scikit-learn	0.00139s	0.00150s	0.00147s	0.00149s

Apparently the difference between number of neighbors is negligible in computation time, but the difference between ABDO and Scikit-learn is well visible. Scikit-learn k-NN classifier is about two times faster than ABDO k-NN classifier.

Digits time	**k = 3**	**k = 5**	**k = 8**	**k = 16**
ABDO	2.44812s	2.42847s	2.40850s	2.42202s
Scikit-learn	1.25032s	1.25942s	1.25146s	1.24723s

In Digits dataset, the same tendency is visible, computation time is almost the same for k = 3 and k = 16, but ABDO classifier is again about two times slower.

8. Bibilography

- load_digits - scikit-learn 1.8.0 documentation
- load_iris - scikit-learn 1.8.0 documentation
- KNeighborsClassifier - scikit-learn 1.8.0 documentation
- wikipedia.org/wiki/K-nearest_neighbors_algorithm