11752 Machine Learning
Master in Intelligent Systems
Universitat de les Illes Balears

Handout #3: **Instance-based learning**:
Support Vector Machines & k-Nearest Neighbour Classifiers
(graded assignment)

NOTE 1: Tasks T1, T2 and T3 respectively require loading datasets `dsgg1.txt`, `dsgg2.txt` and `dsgg3tr.txt`/ `dsgg3te.txt`, where `gg` is the group number:

```
import numpy as np
group = '01' # assuming group 1
ds = 1       # assuming task 1
data = np.loadtxt('ds'+group+str(ds)+'.txt')
X = data[:, 0:2]
y = data[:, 2:3]
```

Class labels are 1 for ω_1 and 0 for ω_2.

NOTE 2: Tasks T1 and T2 require the use of a Quadratic Programming solver. You are advised to use the Python library `cvxpy` (`https://www.cvxpy.org/`). This library can be installed by means of:

```
pip install cvxpy                  # if you use pip
conda install -c conda-forge cvxpy # if you use conda (inside an anaconda installation)
```

(2.5p) T1. **Given dataset** `dsgg1.txt`:

a) Solve for the SVM analytically using the Karush-Kuhn-Tucker conditions and the Wolfe dual representation making use of the quadratic programming solver and

1. find and report the *support vectors* (NOTE: due to round-off errors, it is likely none of the λ_i are exactly 0, but close, e.g. 10^{-6}), and
2. calculate and report the resulting *decision function* $g(x) = w^T x + w_0$ [1].

b) Generate the following plots:

1. a first plot with the *training samples*, highlighting the *support vectors* and plotting the 2D *decision curve*
2. a second plot with the *classification map*, i.e. evaluate the *decision function* for a 'regular' subset (grid) of points of the feature space

Use different markers and/or colours for each class.

c) Compare the results obtained with the ones resulting from the `scikit-learn SVC` object: i.e. report the *support vectors* returned by `SVC` and the corresponding *decision function*.

NOTE: The SVC object solves the soft-margin kernel-based problem, hence you will have to select the *linear* kernel and set constant C to a high value, e.g. 10^{16}, to force a perfect classification of the training set. See `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html` for more detail.

(4.5p) T2. **Given dataset** `dsgg2.txt`:

a) Mapping the *training samples* $x = (x_1, x_2)$ onto an alternative 2-dimensional space using $\Phi(x) = (x_1 x_2, x_1^2 + x_2^2)$, solve for the SVM analytically using the quadratic programming solver and

1. find and report the *support vectors* in the original space, and

[1] In https://jupyterbook.org/content/math.html you can find tools for typesetting mathematical expressions in notebooks

2. calculate and report the resulting *decision function* both in the transformed space $g_1(x') = w^T x' + w_0$ $[x' = \Phi(x)]$ and in the original space $g_2(x) = w^T \Phi(x) + w_0$.

b) Generate the following plots:

1. a first plot with the *training samples* in the transformed space, highlighting the *support vectors* and plotting the 2D *decision curve*;
2. a second plot with the *training samples* in the original space, highlighting the *support vectors* and plotting the 2D *decision curve*; and
3. a third plot with the *classification map* in the original space, i.e. evaluate the *decision function* for a 'regular' subset (grid) of points.

Use different markers and/or colours for each class.

c) Compare the results obtained with the ones resulting from the `scikit-learn` `SVC` object: i.e. report the *support vectors* returned by `SVC` and the corresponding *decision function*.

NOTE: You will have to specify `kernel = 'precomputed'` when creating the `SVC` object, and supply the *Gram matrix* —a square matrix $G(i,j) = x_i \cdot x_j$, i.e. the H matrix without the y labels— to the **fit** method instead of the data matrix X. As before, set constant C to a high value, e.g. 10^{16}, to force a perfect classification of the training set. See `https://scikit-learn.org/stable/modules/svm.html#custom-kernels` regarding the Gram matrix.

d) Also by means of the `scikit-learn` `SVC` object, repeat point c) for the `'rbf'` kernel. Additionally, draw the corresponding RBF network (slide 42 of the SVM lecture notes, replacing $K(x_i, x)$, λ_i and y_i by your values).

NOTE: As before, set constant C to a high value, e.g. 10^{16}, to force a perfect classification of the training set, and also set $\gamma = 1$.

(3p) T3. **Given dataset** `dsgg3tr.txt/dsgg3te.txt`, find a suitable k-Nearest Neighbour (k-NN) classifier to adequately address the involved task (`KNeighborsClassifier` object of `scikit-learn`, `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`).

a) Define the design strategy: combinations of hyper-parameters considered (number of neighbours and distance function), tuning approach (folds and repetitions).

b) Using the *training dataset*, find the best performing classifier according to the design strategy and employing the accuracy as performance metric for the cross-validation process.

c) Plot the training samples on top of the *classification map*, i.e. evaluate the decision function for a 'regular' subset (grid) of points of the feature space (use different markers and/or colours for each class).

d) Report on the classifier performance using the *test dataset*:

1. measure the *test accuracy*, *test precision*, *test recall* and *test f1-score*, and
2. in a single figure, plot the *test samples* over the already calculated *classification map* (use different markers and/or colours for each class).

e) Obtain an improved estimation of the *accuracy*, *precision*, *recall* and *f1-score* measures by means of *repeated, n-fold cross-validation*. To this end, put together the *training* and *test* datasets, so that the corresponding function can build the *folds* from all available data.

DELIVERY INSTRUCTIONS:

- To implement the solutions to tasks T1, T2 and T3, you can either use a notebook file (`.ipynb`) or separate python files (`.py`). In the latter case, use a python file for each task and include inside all the source code that is needed to run the solution to the task.

  The name of the python files has to be `alltasks.ipynb`, or `task1.py`, `task2.py`, etc.

- Brief/suitable comments are expected in the source code.

- A report of the work done has to be delivered by/on January 17, 2024 in PDF form. The report can be generated by exporting the notebook file (after full execution) or using a separate text editor; you can find a template in `.docx` format in the course web page that you can adapt for the `.ipynb` case.

  Upload a Zip container to package the report (with name `report.pdf`) and the source code files (.ipynb or .py file(s)).

- This work can be done in groups of 2 students (contact me with the name of the members of the group for getting a group number; the members of each group and their group number will be included in a list available in the course web page).

- IMPORTANT NOTICE: An excessive similarity between the reports/source code released can be considered a kind of plagiarism.