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

Handout #1: **Instance-based Learning**

<u>NOTE 1</u>: The following problems require loading dataset `dsggp.txt` where `gg` is the group number and `p` is the problem number:

```
import numpy as np
group = '01' # assuming group 1
ds = 1       # assuming problem 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.

Problems P3 and P4 require *training* and *test datasets*. They are, respectively, stored in `dsxx34tr.txt` and `dsxx34te.txt` files.

<u>NOTE 2</u>: Problems P1 and P2 require the use of a Quadratic Programming solver, which can be obtained from library `qpsolvers` (`https://pypi.org/project/qpsolvers/`). This library can be installed by means of:

```
pip install cvxopt --user
pip install qpsolvers
```

When calling function `solve_qp`, choose solver `'cvxopt'`.

<u>NOTE 3</u>: All problems will require the use of `scikit-learn` (`https://scikit-learn.org`) and `matplotlib` (`https://matplotlib.org/`). Apart from considering the library functions suggested at certain points, you can make use of others which may be relevant at each point (to this end, page `https://scikit-learn.org/stable/modules/classes.html` will be useful; sections `sklearn.svm`, `sklearn.neighbors`, `sklearn.preprocessing`, `sklearn.metrics` and `sklearn.model_selection` are of particular relevance).

P1. **Given dataset** `dsxx1.txt`:

a) Solve for the SVM analytically using the Karush-Kuhn-Tucker conditions and the Wolfe dual representation making use of a quadratic programming solver and
   1. find and report the *support vectors* (<u>NOTE</u>: 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* $g(x) = w^T x + w_0$.

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. See the appendix for examples of the requested plots.

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*, and provide the same kind of plots requested before.

   <u>NOTE</u>: the SVC object solves the soft-margin kernel-based problem, hence you will have to select the *linear* kernel and set constant C with a high value, e.g. 10^{16}, to force a perfect classification of the training set.

P2. Given dataset `dsxx2.txt`:

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

   1. find and report the *support vectors* in the original space (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* 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. See the appendix for examples of the requested plots.

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*, and provide the same kind of plots requested before.

   NOTE: the SVC object solves the soft-margin kernel-based problem, hence you will have to supply the kernel specified in a) –use either `kernel = 'precomputed'` and compute the *gram matrix*, or supply a *callable* kernel when invoking the SVC object constructor– and set constant C with a high value, e.g. 10^{16}, to force a perfect classification of the training set.

d) Also by means of `scikit-learn SVC` object, repeat point c) for the `'rbf'` kernel ($\gamma = 1$). 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.

P3. Given datasets `dsxx34tr.txt` and `dsxx34te.txt`, find a suitable SVM classifier adopting a *soft-marging* approach. You have to define the classifier design strategy, including data normalization, e.g. *min-max* scaling, and setting up the classifier hyper-parameters, e.g. by means of *grid-search*, as well as estimate the classifier performance by means of *n-fold cross validation*.

a) Define the design strategy: input data normalization, combinations of hyper-parameters considered (kernel and its parameters, and C), number of folds, performance metric employed in the cross-validation process.

   NOTE: typical values for C are $10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}$ and 10^{3}.

b) Using the *training dataset*, find the best performing classifier according to the design strategy.

c) Generate the following plots in the original space:

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

   Use different markers and/or colours for each class. See the appendix for examples of the requested plots.

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* and *recall* measures by means of *5-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.

P4. **Given datasets `dsxx34tr.txt` and `dsxx34te.txt`**, find a suitable k-NN classifier (`KNeighborsClassifier` object of `scikit-learn`). You have to define the classifier design strategy, including data normalization, e.g. *min-max* scaling, and setting up the classifier hyper-parameters, e.g. by means of *grid-search*, as well as estimate the classifier performance by means of *n-fold cross validation*.

a) Define the design strategy: input data normalization, combinations of hyper-parameters considered (number of neighbours and distance function), number of folds, performance metric employed in the cross-validation process.

b) Using the *training dataset*, find the best performing classifier according to the design strategy.

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* and *recall* measures by means of *5-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.

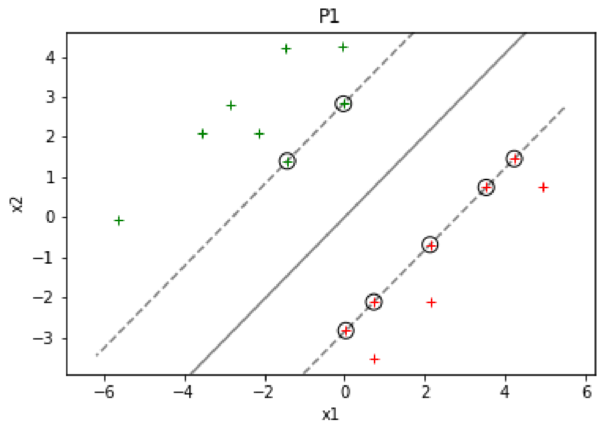
- A report of the work done, problem by problem and point by point, has to be released by January 10, 2021 in electronic form using the templates provided through *Aula Digital*. Notice that there is a *template for results* and another *template for the source code*. Zip the corresponding PDF files, together with an executable source file, either a notebook file (.ipynb) or a python file (.py), whatever you have used for solving the different problems (3 files in total).
- Provide the requested data and plots/figures at each point above. For figures, use appropriate titles, axis labels and legends to clarify the results reported.
- Suitable comments are expected in the source code.
- This work has to be done individually (see the number of group in *Aula Digital*).
- IMPORTANT NOTICE: An excessive similarity between the reports released can be considered a kind of plagiarism.

Appendix:

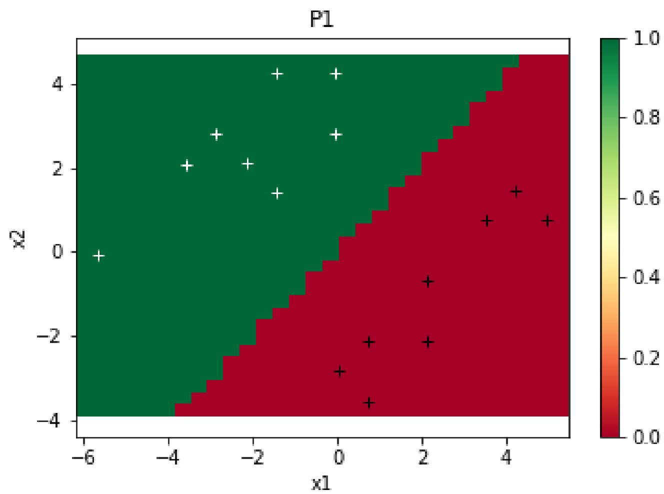
These are the kind of graphical results which are expected for P1.b and P1.c (linear SVM), P2.b and P2.c (non-linear SVM), etc.

<u>LINEAR CASE</u>

example of plot with training samples and the 2D decision curve, highlighting the support vectors

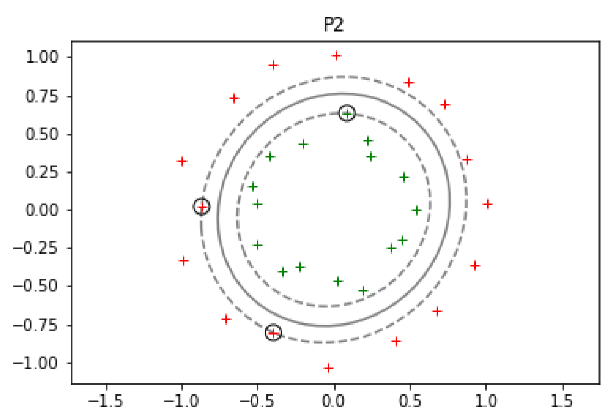


example of classification map (in this case, label 0/*black crosses* corresponds to the class at the "negative" side of the hyperplane)

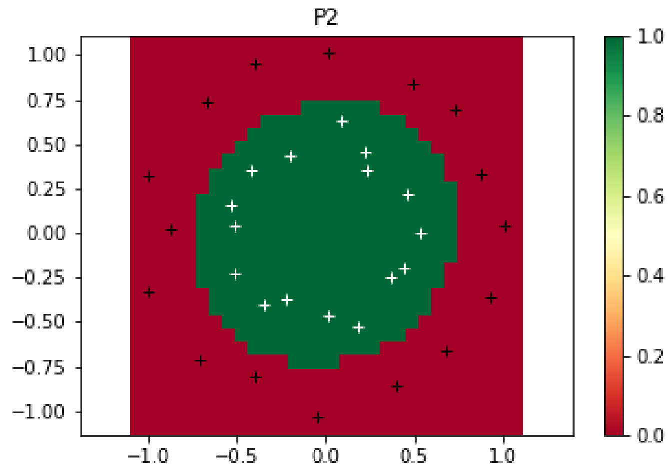


<u>NON-LINEAR CASE</u>

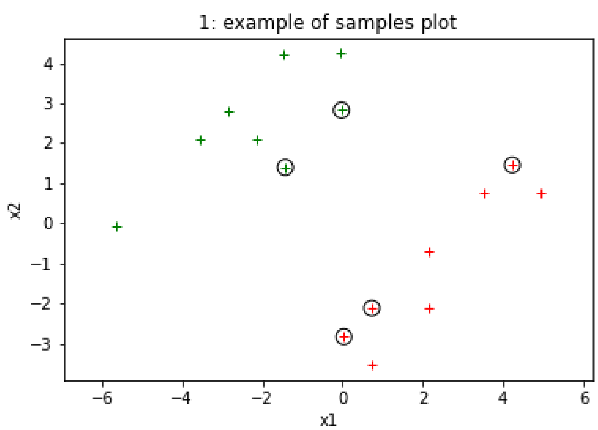
example of plot with training samples and the 2D decision curve, highlighting the support vectors

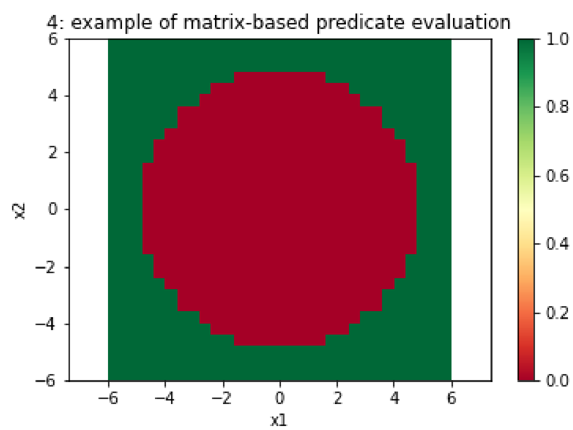


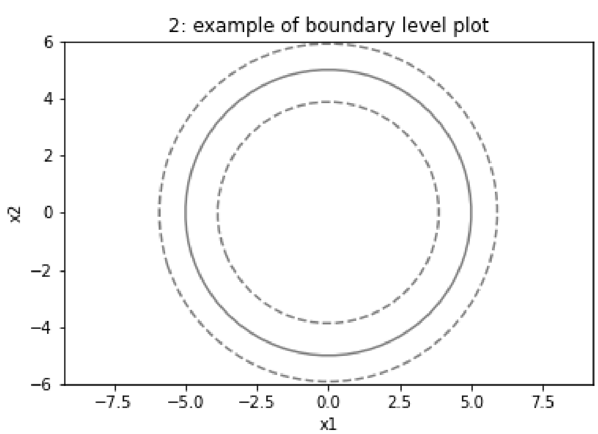
example of classification map (in this case, label 0/*black crosses* corresponds to the class at the "negative" side of the hyperplane)

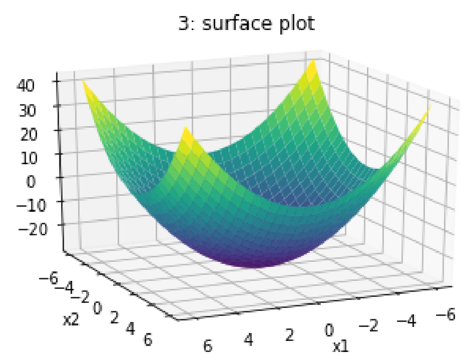


By way of example, the next plots:









have been generated by means of the following source code (assuming X has been properly loaded):

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
```

```
# plot samples, highlighting some of them
plt.figure(1)
plt.plot(X[0:8,0],X[0:8,1],'+r')   # class w1
plt.plot(X[8:16,0],X[8:16,1],'+g') # class w2
hil = [1,  2,  5, 10, 11] # samples to highlight
ax = plt.gca()
ax.scatter(X[hil,0], X[hil,1], s=100, linewidth=1, facecolors='none', edgecolors='k')
plt.xlabel('x1')
plt.ylabel('x2')
plt.axis('equal')
plt.title('example of samples plot')
plt.show(block=False) # to force visualization

# create grid to evaluate function
xx = np.linspace(-6, 6, 30)
yy = np.linspace(-6, 6, 30)
YY, XX = np.meshgrid(yy, xx)
Z = np.zeros((30 * 30,1))
k = 0
for x1 in xx:
    for x2 in yy:
        Z[k] = x1 ** 2 + x2 ** 2 - 30
        k += 1

# plot boundary of levels -15, -5 and +5
plt.figure()
ax = plt.gca()
ax.contour(XX,YY,Z.reshape(XX.shape),colors='k',levels=[-15, -5, 5],alpha=0.5,linestyles=['--', '-', '--'])
plt.xlabel('x1')
plt.ylabel('x2')
plt.axis('equal')
plt.title('example of boundary level plot')
plt.show(block=False) # to force visualization

plt.figure(3)
ax = plt.axes(projection='3d')
ax.plot_surface(XX, YY, Z.reshape(XX.shape), rstride=1, cstride=1, cmap='viridis', edgecolor='none')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('surface plot')
plt.show(block=False) # to force visualization

# matrix-based predicate evaluation
C = np.where(Z >= -5, 1, 0)
plt.figure(4)
plt.imshow(C.reshape(XX.shape), origin='lower', extent=(-6, 6, -6, 6), cmap='RdYlGn')
plt.colorbar()
plt.xlabel('x1')
plt.ylabel('x2')
plt.axis('equal')
plt.title('example of matrix-based predicate evaluation')
plt.show(block=False) # to force visualization
```