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

Handout #2: **Supervised learning** (GRADED)

(0.5p) T0. (a) For this assignment, you will need a dataset that you can load using the following source code, where `gg` is the group number (you can find your group number in the course web page):

```
import pandas as pd
df = pd.read_csv('ds%02d.csv' % (gg))
```

This dataset is a subset of the *Spotify Songs* dataset (`https://www.kaggle.com/datasets/joebeachcapital/30000-spotify-songs/`). As you can observe in the dataset description, each entry contains identification data of the track and a number of features attributed to the song. Your subsets contain the following 12 features (you can find the description in the web page specified above):

danceability	*energy*	*key*	*loudness*	*mode*	*speechiness*
acousticness	*instrumentalness*	*liveness*	*valence*	*tempo*	*duration_ms*

The problem consists in being able to predict whether a song will become *highly popular* or not. To this end, a 13-th column has been added to the dataset to define two classes: *highly popular* (ω_1, class label 1) and *not popular enough* (ω_2, class label 0). Class labels have been generated using the feature *track_popularity* (*tp*) of the original dataset, which takes values between 0 and 100 (the higher, the more popular): tracks with $tp > 90$ have been classified as *highly popular* and the others have been considered as *not enough popular*.

(b) Normalize the dataset samples using *max-min normalization* and consider the following cases:

i. **case A**. All features.

ii. **case B**. Best two features according to PCA.

(1.5p) T1. **(only for case B)** Assuming that class data follow a 2D Gaussian distribution, consider the **quadratic Bayesian classifier** case, i.e. different covariance matrices for each class, find and report the discrimination function $g_i(x)$ for each class ω_i, i.e. $g_i(x) = a_i x_1^2 + b_i x_2^2 + c_i x_1 x_2 + d_i x_1 + e_i x_2 + f_i$, and evaluate its performance. HINT: You need to calculate the mean and a covariance matrix for each class.

(1.5p) T2. **(only for case B)** Assuming that class data follow a 2D Gaussian distribution, consider the **linear Bayesian classifier** case, i.e. same covariance matrix for each class, find and report the discrimination function $g_i(x)$ for each class ω_i, i.e. $g_i(x) = a_i x_1 + b_i x_2 + c_i$, and evaluate its performance. HINT: You need to calculate the mean for each class and a single covariance matrix for the full dataset, shared by all classes.

(1p) T3. **(cases A and B)** Implement a **generic Bayesian classifier** (BC) estimating the probability $p(x|\omega_i)$ using the *probability density estimator* (PDE) available in *scikit-learn* for a Gaussian kernel, and evaluate its performance. HINT: The *scikit-learn* implementation of a PDE is available as object *KernelDensity*[1]. For the *bandwidth* h use the recommendation of slide 47 with $h_1 = 1$. Notice that the method *score_samples* of the density estimator provides you with the log-likelihood $\log_e p(x|\omega_i)$ instead of directly $p(x|\omega_i)$.

(1p) T4. **(cases A and B)** Make use of the implementation of the **naive Bayes** classifier (NB)[2] available in *scikit-learn* and evaluate the resulting classifier. HINT: The *scikit-learn* implements several variations of the NB classifier, use the implementation for Gaussian classes that is available as object *GaussianNB*[3].

(1p) T5. **(cases A and B)** Make use of the implementation of **logistic regression** (LR) available in *scikit-learn* and evaluate the resulting classifier. HINT: The *scikit-learn* implementation of LR is available as object *LogisticRegression*[4]. Do not incorporate any regularization term (`penalty = None`).

[1] `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KernelDensity.html`
[2] `https://scikit-learn.org/stable/modules/naive_bayes.html`
[3] `https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html`
[4] `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression`

(1.5p) T6. **(cases A and B)** Adopt a classifier ensemble approach using the **stacking** approach[5], comprising NB, LR and decision tree (DT) classifiers. HINT: The *scikit-learn* implementation of the DT classifier is available as object *DecisionTreeClassifier*[6]. Use also a DT for the second layer. For the first layer, regularize the DT imposing a minimum of 10 samples per leaf, while, for the second layer (the blender), impose a maximum depth of 3.

(1.5p) T7. **(cases A and B)**

(a) Adopt a classifier ensemble approach using the implementation of **random forests** (RF) available in *scikit-learn*. HINT: The *scikit-learn* implementation of RF is available as object *RandomForestClassifier*[7].

(b) Tune the RF model by means of *grid search* (with `cv = 3`) as follows: (1) number of trees among 20, 40 and 60, (2) minimum samples per leaf equal to 5 or 10, and (3) consider the two impurity criteria *gini* and *entropy*. HINT: The *scikit-learn* implementation of grid search is available as object *GridSearchCV*[8].

(0.5p) T8. For the best model (according to the F_1-score) that you have obtained for your dataset:

(a) Generate the classification map for **case B** using the following source code given the *model* object, and the test data (X_{te}, y_{te}):

```
def plot_class(c, X, y):
  m1 = ['k','w']
  m2 = ['x','o']
  i = np.where(y == c)[0]
  plt.scatter(X[i,0], X[i,1], c=m1[c], marker=m2[c], label='class %d' % (c))

x1lim = [Xte[:,0].min(), Xte[:,0].max()]
x2lim = [Xte[:,1].min(), Xte[:,1].max()]

npts = 100
x1s = np.linspace(x1lim[0], x1lim[1], npts)
x2s = np.linspace(x2lim[0], x2lim[1], npts)

m = np.zeros((npts, npts))
for k1, x1 in enumerate(x1s):
  for k2, x2 in enumerate(x2s):
    x = np.array([x1, x2])
    m[k1,k2] = model.predict([x])

plt.figure()
plt.imshow(m.T,cmap='RdYlGn',origin='lower',extent=(x1lim[0],x1lim[1],x2lim[0],x2lim[1]))
for c in range(M):
  plot_class(c, Xte, yte)
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.legend()
plt.show()
```

Listing 1: Source code for generating the classification map.

(b) Load file ds*gg*_samples.csv (*gg* is the group number) and classify the 4 samples x contained therein, indicating also the probability *a posteriori* for each class $p(\omega_i|x)$. Notice that, in this way, we can say, for the involved track, the probability of becoming a *highly popular* song (according to the training data available). HINT: The `predict_proba` method of *scikit-learn* classification models of tasks T4 – T7 provide **unnormalized** probabilities per class, i.e. the probabilities do not sum to 1, and so you have to normalize them. Notice that for the cases of tasks T1 – T3 you already get unnormalized probabilities.

[5] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html
[6] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[7] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[8] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

(*1) In all cases, you have to evaluate the model performance:

(a) For tasks T1 – T3, use a train set/test set strategy (*holdout cross validation*[9]), with 70% of the dataset for training and 30% for testing.

(b) For tasks T4 – T7, use *3-repetitions, 5-fold cross validation*[10]. Report on the average values and the standard deviations.

(*2) Use the following table to report on the performance for all models, including the results for the samples contained in file `ds`*gg*`_samples.csv`:

Task/Model	**All features (case A)**				**Two features (case B)**			
	A	P	R	F_1	A	P	R	F_1
T1. Quadratic BC	—/—	—/—	—/—	—/—	pval/—	pval/—	pval/—	pval/—
T2. Linear BC	—/—	—/—	—/—	—/—				
T3. Generic BC	pval/—	pval/—	pval/—	pval/—	⋮	⋮	⋮	⋮
T4. NB classifier	avg/std	avg/std	avg/std	avg/std	avg/std	avg/std	avg/std	avg/std
T5. LR classifier								
T6. Stacked classifier	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
T7. Tuned RF classifier								

Best model: *(according to F_1)*	*<name of the model and case A or B>*		
	$\mathbf{p}(\omega_1\|x)$	$\mathbf{p}(\omega_2\|x)$	**class**
Sample #1			
Sample #2	⋮	⋮	⋮
Sample #3			
Sample #4			

In the table, A = accuracy, P = precision, R = recall, F_1 = F_1-score, avg = average, std = standard deviation, pval = performance value. This table can be found as file `results.xlsx`, which you can fill automatically adapting the following source code:

```python
from openpyxl import load_workbook

# Load the excel file
wb = load_workbook(filename = "results.xlsx")

# Grab the active worksheet
ws = wb.active

# Store data directly in cells
ws["B6"] = 0.983 # average accuracy for case A, NB classifier
ws["C6"] = 0.023 # standard deviation of accuracy for case A, NB classifier

# Save the file
wb.save("results.xlsx")
```

Listing 2: How to fill an excel file.

[9] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`

[10] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RepeatedStratifiedKFold.html#sklearn.model_selection.RepeatedStratifiedKFold`

DELIVERY INSTRUCTIONS:

- To implement the solutions to tasks T0 – T8, 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 December 15, 2023 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`), the `results.xlsx` file, the classification map (with name `classmap.png`) 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.