TP 3 – AOS1

Bayesian logistic regression, Gaussian process classification

1 Introduction

This practical session aims first at applying logistic regression with Bayesian regularization in a first step, and to briefly study Gaussian processes in a second step. You will need several libraries for this purpose.

```
import numpy as np
import pandas as pd
import scipy as sp
import scipy.stats as spst
import matplotlib.pyplot as plt
import seaborn as sns
```

2 Logistic regression

2.1 LR without regularization

Logistic regression is implemented in `scikit-learn` via the `LogisticRegression` class; it is available with the following command:

```
from sklearn.linear_model import LogisticRegression
```

The `penalty` argument is particularly important. By default, a ℓ_2 penalization term is used. You will also consider the ℓ_1 term, and no penalization (`penalty="none"`).
After training, several attributes are available, among which `coef_` contains the coefficients of the model, and `intercept_` the intercept coefficient.

1 Consider the synthetic datasets provided [1]. Train a logistic regression model *without regularization*, and visualize the decision boundary using the appropriate function from `utils.py`.

2 Using the `coef_` and `intercept_` attributes, get the expression of the decision boundary.

3 Use the `levels` argument so as to display the level curves of the posterior probability distribution. You may for instance use the levels 0.1, 0.3, 0.5, 0.7 and 0.9.

Remark 1. If the `add_decision_boundary` does not work properly, you can use the following code to display the level curves for the model outputs:

[1] contained in the `SynthCross_n1000_p2.csv`, `SynthPara_n1000_p2.csv` and `SynthPlus_n1000_p2.csv` files

```
Gx, Gy = np.mgrid[-15:16:200j, -15:16:200j]
grid = np.stack([Gx, Gy], axis=-1)

fG = LR_model.predict_proba(grid.reshape(-1, 2))[:, 1].reshape(*Gx.shape)

fig, ax = plt.subplots()
ax.scatter(X.X1, X.X2, c=(y=="A").astype(int))
CS = ax.contour(Gx, Gy, np.reshape(fG, Gx.shape))
ax.clabel(CS, inline=1, fontsize=10)
ax.set_title('Contour plot, $p(y^*=1|x^*,X,y)$')
```

2.2 LR with regularization

4 Sub-sample the dataset so as to keep only a (small) fraction of the original data (e.g., 1%). Train a logistic regression classifier *without regularization* on the subsampled data and display the results. Compare with the model trained *without regularization* on the whole dataset.

5 Do the same *with regularization*.

3 Gaussian process classification

This part aims at testing Gaussian process classification on synthetic data. You can import the `scikit-learn` Gaussian process classification implementation using

```
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import ConstantKernel as CK, RBF
```

6 Generate a 1D synthetic dataset with $n_1 = 20$ instances from class c_1 and $n_2 = 20$ instances from class c_2, with

$$X \underset{c_1}{\sim} \frac{1}{2}\mathcal{N}(-2,1) + \frac{1}{2}\mathcal{N}(2,1), \quad X \underset{c_2}{\sim} \mathcal{N}(0,1).$$

7 Train a Gaussian process classifier on these 1D data with the following code. Display the model outputs and interpret.

```
rbf = CK(1.0) * RBF(length_scale=1.0)
gpc = GaussianProcessClassifier(kernel=rbf)
```

8 Generate a 2D dataset using the `make_moons` function from the `datasets` component of the `scikit-learn` library. Display the data obtained.

9 Fit a Gaussian process classification model to the data. Display the results.