TP 3 – AOS1

Regularized logistic regression

1 Introduction

The purpose of the practical session is to program and use (binary) logistic regression, and then use two kinds of Bayesian regularization in order to improve its results.

2 Binary logistic regression

2.1 Implementation

We will begin with implementing binary logistic regression. The algorithm used for training will be the Newton-Raphson algorithm presented in the course.

You can compare the results with those obtained via the `scikit-learn` implementation, using the following instructions (mind the `penalty` argument):

```
from sklearn.linear_model import LogisticRegression as SklearnLogisticRegression
sk_cls = SklearnLogisticRegression(penalty="none")
sk_cls.fit(X, y)
sk_cls.coef_
sk_cls.intercept_
```

1 Fill-in the missing parts in the `src/logistic_regression.py` file provided.

2 Train the logistic regression model on the data contained in the `SynthPara_n1000_p2.csv` dataset and compare the model obtained to the one obtained via the `scikit-learn` function. The decision boundary can be plotted using the `add_decision_boundary` function.

2.2 Polynomial logistic regression

The logistic regression model inherently provides linear decision boundaries. Its extension to nonlinear classification problems is however straightforward. The principle is to extend the original dataset onto a nonlinear space, where the decision boundary is (supposedly) linear. This generalization of logistic regression comes however at a price: the number of parameters to be estimated is indeed higher.

Here, we will explore a strategy based on the polynomial expansion of the input variables in order to introduce more flexibility in logistic regression. For instance, assume that $\boldsymbol{X} = (X_1, X_2, X_3)$, then mapping the instances into a second-order polynomial space feature would lead to define a new feature vector

$$\tilde{\boldsymbol{X}} = \left(X^1, X^2, X^3, X^1X^2, X^1X^3, X^2X^3, (X^1)^2, (X^2)^2, (X^3)^2\right).$$

Polynomial (and thus quadratic) expansions are already implemented in `scikit-learn` through the model `PolynomialFeatures`:

```
from sklearn.preprocessing import PolynomialFeatures
```

This class admits the polynomial degree of the expansion as input: for instance, the third-degree polynomial expansion of the feature vector can be obtained using the following code.

```
poly = PolynomialFeatures(degree=3)
poly.fit_transform(X)
```

The polynomial transform can be composed with other processings by creating a pipeline.

```
from sklearn.pipeline import make_pipeline
poly = PolynomialFeatures(degree=2, include_bias=False)
cls = LogisticRegression()
pipe = make_pipeline(poly, cls)
```

3 Using both your implementation and the `scikit-learn` version, make pipelines so as to compute the d-order polynomial expansion of the data in the `SynthNLin_n1000.csv` and learn a (non-penalized) logistic regression model on the expanded data, for increasing degrees $d = 1, 2, \dots, 10$. Plot the decision boundaries and compare. What do you notice as the degree increases ?

4 Using the `scikit-learn` logistic regression implementation only, make pipelines so as to compute the d-order polynomial expansion of the same data (for increasing degrees $d = 1, 2, \dots, 8$) and learn three logistic regression models on the expanded data:

- a non-penalized one,
- a ℓ_2-penalized one,
- a ℓ_1-penalized one (you will require to change the solver to `liblinear`).

Again, plot the decision boundaries and compare. Also analyze the coefficients of the model (which can be accessed, e.g. for the non-penalized model learnt previously, via `pipe_skl[1].coef_`). What do you notice ?