Sequence 3 exercises: logistic regression

1. Predicting a breast cancer (Homework)

The goal of this exercise is to implement logistic regression (or binary classification here) and to test it on a dataset in order to understand how it works. We will play with medical data and build a model in order to predict if the data collected from patient nuclei is **malignant** or **benign**. You can get more information about the dataset
here: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic).

Download and complete the file *breast_cancer.py* according to the following questions.

1.1 Data understanding

X contains **30 columns**. It corresponds to the mean, the standard error, and the "worst" or largest (mean of the three largest values) of these features that were computed for each cell nucleus for each image:

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter^2 / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

This results in 30 features per image. For instance, column 3 is Mean Perimeter, field 13 is Perimeter SE, field 23 is Worst Perimeter.
y contains the target variable detailing if the tumor is benign (y=1) of malignant (y=0).
How many samples are there in the dataset? How many positive and negative samples in the dataset?

1.2 Logistic regression

Split the data into a training set with 70% of the available data and a test set with 30%. Fit a logistic regression on the training dataset (look at the *LogisticRegression()* function of the *scikit-learn* library) and compute the obtained training and testing accuracies with default hyper-parameters. In a first run, do not normalize the data features and normalize them in the second run (cf. *StandardScaler()* of the *sklearn.preprocessing* library). Compare the obtained results.

1.3 Tuning the regularization hyper-parameter

The *LogisticRegression()* function proposed in the scikit-learn library introduces a regularization term in order to prevent from overfitting. The default used value for the hyper-parameter, that controls the influence of the regularization term in the total loss function, is C=1. Be careful, in the proposed function, C can be considered as $1/\lambda$ with respect to the cost function formula defined in the lecture, meaning that if C is high, the regularization impact of the cost function is low and the opposite:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right]$$

Make this hyper-parameter vary from 0.1 to 4 with a 0.1 step. Plot the evolution of the training and testing accuracies on the same figure. Conclusion.

2. Multiclass Model Performances (classwork)

In this exercise, we are going to classify the 10 class digit dataset of the *scikit-learn.datasets* module using a logistic regression.
Download the *Multiclass_digits.py* file to be completed. The dataset has already been downloaded.

2.1 Dataset analysis
By looking at the documentation related to the digit dataset, explain the contain of the dataset. How many samples? With which features? And which labels? How many samples per class?

2.2 Logistic Regression model
Split your data into a training set of 80% of the data and a test set of 20%. Specify the random_state to 0 so that you get **reproducible** results. Fit a Logistic Regression on your data. Specify the use of the One versus All strategy in the *Linear_model.LogisticRegression()* function. Predict the class of the 200th sample.

2.3 Performances analysis
Report the training and testing accuracies.
In order to be more precise in the performance analysis, compute and report the confusion matrix (cf. *confusion_matrix()* function). What are the FP, TP, FN and TN values of class 8?
Report also the accuracy per class (look at the *classification_report()* function from the *sklearn.metrics* module). Which class is the worse predicted? Explain what the presented numbers represent.

3. Non Linear Data classification (classwork optional)

In this exercise, we will deal with data that is not **linearly separable**, which simply means that we cannot separate classes with a line. We will see how it becomes more difficult for models to fit it and we will explore a few options to tackle that.
Look at the *NL_data.py* file which has to be completed
First, we load and visualize the data for you using *make_circles* built-in function from *scikit-learn*

3.1 Logistic Regression

3.1.1 Fit a logistic regression to the data. In this exercise, use the *LogisticRegression()* function of the *sklearn.linear_model* module. What is the accuracy of your model on the whole dataset (no data splitting in this exercise)? Comment.

3.1.2 We want to visualize the decision boundaries of our model to better understand the previous performance. We will predict each pixel and assign it a color.

We create for you the grid of points.

- Store in the new variable y_grid the value of the predictions of your model.
- Run last block of code, it should successfully plot the data points and paint the decision boundaries

Comment the decision boundary. Is the logistic regression fitting properly the data?

3.2 Adding new features

3.2.1 Now, what if we create a new variable $z = \{x^2_1 + x^2_2\}$ and that we plot the points X in the dimension (x1, z)? Let's see what happens... A new variable *X_new* corresponding to X with an additional column $z = \{x^2_1 + x^2_2\}$ has been created and plotted in the dimensions (x_1, z). Comment the figure.

3.2.2 Fit a logistic regression on the new dataset and compute the accuracy. Comment.