Machine Learning Methods

# Exercise 3

Last updated: 4.2.24

## 1 General Description

In this exercise, you will solve linear classification problems analytically and using gradient descent methods. You will also learn how to work with the Pytorch Library [1], a very useful library for gradient based algorithms, specifically Neural Networks (Brace yourself were almost there!). We will work with a similar dataset as the previous exercise. As you saw, the US states are very suitable for tree based methods, due to their axis aligned division. This time we will work with some more complex border, in Europe.

You are expected to hand in a report, no longer than 8 pages, describing your experiments and answers to the questions in the exercise. The submission guidelines are described in Sec. 11, please read them carefully.

**Note 1: You must explain your results in each question! Unless specified otherwise, an answer without explanation will not be awarded marks.**
**Note 2:** When you are asked to plot something, we expect that plot to appear in the report.

## 2 Seeding

You should seed your code as in the previous exercises. Specifically, we will need to see both numpy and pytorch as follows:
(i) `np.random.seed(42)`
(ii) `torch.manual_seed(42)`
**NOTE:** It doesn't matter which seed you just that your results are reproducible!

## 3 Provided Code Helpers

Like the previous exercise, you are provided with helper functions. The helper functions are not mandatory, but they can greatly assist you. They are found in `helpers.py`.

We also provide you with a skeleton code for the Ridge and Logistic Regression parts. Unlike the previous helpers, these skeleton files are mandatory and you must fill-in the blanks for Sec. 6&9.

# 4 Resources

In this exercise you will experiment with training models using a GPU (Graphics Processing Unit). This requires a GPU which is configurable for working with pytorch. You probably don't have one on your personal computer, and even if you do it might take you several days to operate it for the first time. In other words extremely not recommended. Instead, implement your exercise in Google Colab, which is more than enough for exercise. Google Colab works similarly to a Jupyter Notebook, therefore, you may submit a Jupyter Notebook as your code. Remember to choose the GPU runtime under the "Runtime" $\rightarrow$ "Change runtime type" tab. **YOU DO NOT NEED ANYTHING MORE THAN THE FREE VERSION FOR THIS EXERCISE!**. Our experiments are very lightweight. If you are having trouble with slow runtimes (hours) your implementation is probably very inefficient.

# 5 Data

The dataset used for our exploration consists of six distinct files: *train.csv, validation.csv, test.csv*. Each file is structured as a table, with three columns: longitude (`long`), latitude (`lat`) and `country`. The samples in our dataset are represented as 2D coordinates of cities in Europe from different countries, with the corresponding labels indicating the country. For instance, if we consider a 2D city coordinate such as $(48.866667, 2.333333)$ – representative of Paris – its label would be *'France'*. To make it easier, we already encoded the `country` column into integers.

The samples in our dataset can be denoted as $\mathcal{X} \in \mathbb{R}^{N \times d}$, where $N$ represents the number of samples, and $d$ is the representation dimension. In our specific case, $d = 2$ as we are dealing with longitude and latitude coordinates.

We will use this spatial dataset for exploring linear methods, by classifying the cities into countries based on their geographical coordinates.

# 6 Ridge Regression - Analytical Solution

As you saw in class, ridge regression is a simple algorithm. It optimizes a linear regression by minimizing:

$$\underset{W}{\operatorname{argmin}} \left[ ||XW - Y||_2^2 + \lambda ||W||_2^2 \right] \tag{1}$$

where $\lambda$ is the regularization strength, and $y \in -1, 1$. Then, to predict a test sample's $(x)$ class we compute $Wx$ and classify by:

$$\hat{y} = \begin{cases} 1 & \text{if } W^T x \geq 0 \\ \text{-1} & \text{if } W^T x < 0 \end{cases} \tag{2}$$

where $\hat{y}$ is the class prediction.

In this exercise, you implement this analytical solution.

We remind you that the optimal solution ($W^*$) is:

$$W^* = (\frac{X^T X}{N_{train}} + \lambda I)^{-1} \frac{X^T Y}{N_{train}} \tag{3}$$

### 6.1 Task

Implement a ridge regression classifier according to the skeleton file. Train that classifier on the train data (*train.csv*), with the following choices of $\lambda$: $0., 2., 4., 6., 8., 10.$.

### 6.2 Questions

1. Plot the training, validation and test accuracies of the models vs. their $\lambda$ value (x axis - $\lambda$, y axis - accuracy). Report the test accuracy of the best model according to the validation set.

2. Using the visualization helper, plot the prediction space of the best and worst $\lambda$'s according using the validation set. Use the test points for the visualization. How does the $\lambda$ parameter affect the algorithm? Explain.

## 7 Gradient Descent in NumPy

This section should take you very little time. We simply want you to implement a gradient descent for a simple function in NumPy, and optimize an input according to it. Our function will be:

$$f(x, y) = (x - 3)^2 + (y - 5)^2 \tag{4}$$

Optimize a $(x, y)$ vector to reach the minimum of $f(x, y)$ using gradient descent. Use a learning rate of 0.1, for 1000 iterations. Initialize your $(x, y)$ vector to be $(0, 0)$.

1. Plot your optimized vector through the iterations (x axis - x, y axis - y). Color the points by the "time" (iterations). Which point did you reach? (No explanation needed here).

# 8 PyTorch: A Powerful Library for Gradient Based Methods

**What is PyTorch?** PyTorch (named `torch`) in code) is a library similar to `numpy`. Its arrays are called tensors, and they are operated in a similar way.
**Why `torch` over `numpy`?** Long story short: Pytorch saves the computational graph of tensors, and can automatically compute the gradient of each tesnor w.r.t any loss function).
Now in some more details: Pytorch is designed in consideration of tracking another feature of tensors, which is their gradient w.r.t some (unknown) function. To be able to do so, for any function, It keeps track of where tensors came from. Meaning, for each tensor it can automatically remember what are the mathematical operations that lead to its computation. This tracking graph is called the computational graph. Pytorch retains this computational graph for its tensors, at any time. Then, in a single command, pytorch can use this computational graph to automatically compute the gradient of any tensor with respect to any defined function (as long as all the operations done are derivable). This is the true power of pytorch. You can simply write your algorithm and "loss" function, and it automatically computes for you how to change the algorithm's parameters to fit this function. If this was not clear for you, we highly recommend attending the reception hour of Tal and Jonathan, where this will be explained in person if needed.

## 8.1 Task

Run the given tutorial in `pytorch_tutorial.py` file. The required inputs and outputs are given in the code. It should slowly guide you for how we use pytorch, and how easily we can use it to easily train gradient based models.

# 9 Logistic Regression - Stochastic Gradient Descent

You will now solve the logistic regression problem on the same data, using stochastic gradient descent and pytorch.

## 9.1 A Reminder on Logistic Regression

In Logistic Regression we also optimize a linear operator but it followed by a Softmax operation, which normalizes the scores to represent a distribution (all positive and sum up to 1). Formally the softmax operator is:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{5}$$

**NOTE**: In practice, pytorch's `nn.CrossEntropyLoss` module already performs a softmax operation for us, therefore we can simply use it as our penalty function, and return the output of our linear operator from the model at training time. At test time we will compensate by using pytorch's `nn.functional.softmax`. Don't forget to perform the softmax on the scores axis (usually the last one: $-1$).

## 9.2 Task

Implement a logistic regression classifier using pytorch, inside the given `model.py` file. Train it on *train.csv* using Stochastic Gradient Descent. Your optimized parameters are just the parameters of $W$ (the linear operator). The softmax operation has no parameters.

## 9.3 Questions - Binary Case

Train logistic regression classifiers on the train data (*train.csv*), with the following choices of learning rates: *learning_rate*: $0.1, 0.01, 0.001$. Use a batch size of 32 samples. Perform the training optimization for 10 epochs (1 epoch = looping over all the samples once).

During training compute the training accuracy and loss in every epoch (as done in part 7 of the pytorch tutorial). In addition, at the end of every epoch, iterate over the validation and test sets and keep their losses accuracies for every epoch as well.

1. Choose the logistic regression model with the best validation accuracy. Visualize its test predictions using the visualization helper.
2. Using the model you selected in Q1, plot the training, validation and test losses over the training epochs in the same figure. Did this model generalize well from the training data? Explain.
3. Compare the results from Q1 to the ones from Q1 of Sec. 6.2. Which method seems to work better? Why do you think so? Explain your answer.

## 9.4 Questions - Multi-Class Case

**NOTE**: Q5 and Q6 are bonus questions. Q5 is worth 5 additional points, and Q6 does not worth any additional points (read it, you will understand why).

We will now transfer to the multi-class case, using the given datasets in *train_multiclass.csv, validation_multiclass.csv and test_multiclass.csv*.
Train logistic regression classifiers on the train data (*train_multiclass.csv*), with the following choices of initial learning rates: *learning_rate*: $0.01, 0.001, 0.0003$).

Train your classifier for 30 epochs, with a batch size of 32. Decay the learning rate by 0.3 every 5 epochs (as seen in the tutorial).
Compute the training, validation and test set losses and accuracies, for every epoch as done in Sec. 9.3.

1. Plot the test and validation accuracies of the model vs. their *learning_rate* value (x axis - *learning_rate*, y axis - accuracy). Report the test accuracy of the best model according to the validation set.

2. Choose the model with highest validation accuracy. Plot its training, validation and test losses over the training epochs in the same figure (y axis - loss, x axis - training epochs). In addition, plot the training, validation & test accuracies in every epoch (y axis - accuracy, x axis - training epochs). Did this model generalize well from the training data? Explain.

3. Use the `sklearn` library to train a decision tree on the data (`sklearn.tree.DecisionTreeClassifier`). Use `max_depth` = 2. Report the tree accuracy and visualize its predictions as before. Compare this model to the model from Q2. Which one is more suitable for this task? Explain.

4. Now, train a Decision Tree Classifier, using `max_depth` = 10. Report the tree's accuracy and visualize its predictions as before. Compare this model to the model from Q2. Which one is more suitable for this task? Has your answer changed with respect to Q3? Explain.

5. **(Bonus 5 points) Adding Ridge Regularization.** Repeat the logisitic regression experiment, but add a ridge regularization. Use the same $\lambda$ values as in Sec. 6.1, and choose the best one according to the validation set. Start from a *learning_rate* of 0.01. Report the accuracy of this model, and plot its predictions using the visuzlization helper. Compare this model to the one from Q2. How are the models different? Which one is better? Exaplain.

6. **Bonus (0 pts)** (Just for the fun of it): Can you guess which countries are present in the data?

# 10 Ethics

We will not tolerate any form of cheating or code sharing. You may only use the `numpy`, `pandas`, `faiss`, `mathplotlib` and `sklearn` libraries.

# 11 Submission Guidelines

You should submit your report, code and README for the exercise as $ex3\_\{YOUR\_ID\}.zip$ file. **Other formats (e.g.** *.rar*) **will not be accepted!**.

The README file should include your name, cse username and ID.

Reports should be in PDF format and be no longer than 5 pages in length. They should include any analysis and questions that were raised during the exercise. Please answer the questions in Sec. 6, 7& 9 in sequential order. **You should submit your code (including your modified helpers file), Report PDF and README files alone without any additional files.**

### 11.1 ChatGPT / Github Copilot

For this exercise specifically, we advise you to avoid ChatGPT / Github Copilot, as one of the purposes is to get familiar with **numpy**. Saying that, we do not prohibit using them. If you do choose to use them, write at the end of your report a paragraph detailing for which parts /functionalities you used them.

### 11.2 Submission

Please submit a zip file named $ex3\_\{YOURID\}.zip$ that includes the following:

1. A README file with your name, cse username and ID.
2. The *.py* files with your filled code.
3. A PDF report.

## References

[1] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.