The goal of this project is to practice implementing your own machine learning optimization routines and play around with different approaches/parameters to see what works well in practice for certain datasets. For this reason, you should not use any functions from machine learning packages such as scikit-learn. For your implementations, please use Python with basic functions from packages such as NumPy and SciPy, or Julia with basic packages such as LinearAlgebra. Please use the same programming language for the entire project. If you prefer, you can use another open-source language, but please consult with the lecturer(s) first.

The project is designed for groups of three. Please collaborate on each question (as opposed to dividing the work). Please hand in your report as a PDF file. Include the code in easy-to-read listings (e.g., use the lstlisting environment in latex with syntax highlighting enabled and a small font to avoid too much line wrapping). Please also submit an additional text file containing a copy of all the code in your report. Use Brightspace to hand in the files. The hand-in date for this project is 17 November 2023.

Write an informal report where you discuss the questions below (please use a subsection for each subquestion). Of course, we know that the algorithms mentioned in this project can be found online or can be partially generated using large language models. Please write your report in such a way that it becomes clear to the reader that you wrote your own version of the code and that you experimented with it yourself. For instance, when answering a question you can give different versions of the same function, where you write things such as "We found that this version of the function was very slow due to the dense matrix operations; each iteration took 12 seconds. The following version of the function uses sparse linear algebra, where the iterations only take 0.8 seconds". Also, whenever there are hyperparameters, discuss your experiences with tuning these. How did this affect the performance on the training and test sets, or how did it affect the runtime? In your answers discuss whether or not the results you see agree with your expectations.

1. The IRIS dataset is one of the oldest and most (over)used example data sets for classification; see the iris.csv file on Brightspace, which contains a part of this dataset.
   (a) Write your own function to extract the data matrix X and the outcome vector y from the data file. Note that the last column corresponds to the outcome vector y, where you could assign the species "versicolor" to $+1$ and "virginica" to -1.
   (b) Write a function that splits the data into a training and test set according to some fraction $0 < f < 1$. Make sure to use randomization; that is, it should not be the case that the training set consists of the first data points and the test set of the remaining data points. Your function should return matrices X_{train} and X_{test} and vectors y_{train} and y_{test}.
   (c) Write a function that, given X, y, and a weight vector w defining a hyperplane, returns the number of correctly classified points. Verify that the output makes sense for random weight vectors.
   (d) Implement two functions for ordinary least squares regression with Tikhonov regularization: one using the QR factorization (for the QR factorization you may use a library function) and the other using gradient descent. Use this to train a weight vector on the training part of your dataset (you could start by splitting this 50/50). Should the answers obtained with QR factorization and gradient descent be the same? Verify whether this is the case. What is the impact of the regularization hyperparameter λ on the classification performance on the test set?

(e) Implement the perceptron. Since the full data set is not linearly separable, the algorithm will run indefinitely if we apply it to the full data set. If we use $f = 0.2$, then depending on which data points are selected, the training set will sometimes be strictly linearly separable. How good is the performance on the test set in such cases? Compare this to the ordinary least squares approach from (d).

2. The file url.mat[1] contains several data sets (a data set for each of 120 days of collecting data), where each row corresponds to one URL. The columns correspond to various (about 30 million) features of the URLs and the web hosts corresponding to the URLs. Properties of the webpages the URLs link to are not part of the feature set. The data points are classified according to whether the resulting webpage is malicious or not. The goal is to decide, based on these features, whether a new URL is malicious without visiting any of the webpage it links to.

   (a) Load the data set for the first day into a matrix X and a vector y. If you use Python or Julia, you can use the code from Listing 1 or 2.

   Listing 1: Python code

```python
import scipy.io

def parse_url(filename="url.mat", day=1):
    v = scipy.io.loadmat(filename)[f'Day{day}']
    X = v['data'][0][0]
    y = [(-1)**v['labels'][0][0][k][0] for k in range(len(v['labels'][0][0]))]
    return X, y
```

   Listing 2: Julia code

```julia
using MAT

function parse_url(filename="url.mat", day=1)
    v = matread(filename)["Day$day"]
    v["data"], (-1).^v["labels"]
end
```

   (b) Implement subgradient descent for the Hinge-loss SVM as discussed in the lecture. Your function should take as arguments the data matrix X and data vector y, the learning rate α (a good default value is $\alpha = 0.001$), an integer N indicating the number of gradient descent steps, and a regularization parameter λ. The function should return a weight vector w.

   Experiment with the hyperparameters, using dense and sparse linear algebra, on random splits of training and test data sets.

   Given for instance a 50/50 split between test and training data for the first day, what is the best classification performance you can obtain on the test set?

   (c) Write a mini-batch version of the previous function. Instead of the integer N, the function takes a batchsize argument indicating the size of the mini-batches, and an epochs argument indicating the number of times each data point should be considered in total. A good default value for the regularization parameter is $\lambda = 1/\text{epochs}$.

   (d) Try training on a few days of data. Does the classification performance improve? Discuss how long you think it may take to train on the full 120 days of data with your code and hardware.

[1] http://www.sysnet.ucsd.edu/projects/url/url.mat