Instructions: Try to solve all problems on your own. If you have difficulties, ask the instructor or TAs. Please use Python.

Binary Classification Problem

In the last lab, you might have recognized that decomposing a problem might help in coming up with optimization procedures to handle large data. In this lab, we will continue with this theme and try to develop procedures which are scalable and achieve reasonably accurate solutions.
Here, we will consider a different problem, namely the binary (or two-class) classification problem in machine learning. The problem is of the following form. For a data set $D = \{(x_i, y_i)\}_{i=1}^n$ where $x_i \in \mathscr{X} \subseteq \mathbb{R}^d$, $y_i \in \{+1, -1\}$, we solve:

$$\min_{w \in \mathbb{R}^d} f(w) = \frac{\lambda}{2}\|w\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} L(y_i, w^\top x_i). \tag{1}$$

Note that we intend to learn a classification rule $h : \mathscr{X} \to \{+1, -1\}$ by solving the problem (1). We will use the following prediction rule for a test sample $\hat{x}$:

$$h(\hat{x}) = \text{sign}(w^\top \hat{x}). \tag{2}$$

We will consider the following loss functions:

- $L_h(y_i, w^\top x_i) = \max\{0, 1 - y_i w^\top x_i\}$ (hinge)
- $L_\ell(y_i, w^\top x_i) = \log(1 + \exp(-y_i w^\top x_i))$ (logistic)
- $L_{sh}(y_i, w^\top x_i) = (\max\{0, 1 - y_i w^\top x_i\})^2$. (squared hinge)
- **Exercise 0:** Plot these loss functions where $z = y_i w^\top x_i$ takes values on the real line $[-\infty, \infty]$. Distinguish the loss functions using different colors.

Exercise 1: Data Preparation

1. Use the following code snippet. Load the iris dataset from `scikit-learn` package using the following code. We will load the features into the matrix A such that the i-th row of A will contain the features of i-th sample. The label vector will be loaded into y.
   (a) Check the number of classes C in iris data.
   (b) When loading the labels into y, convert classes $2, 3, \ldots, C$ to -1.
   (c) Note that a shuffled index array `indexarr` is used in the code. Use this index array to partition the data and labels into train and test splits. In particular, use the first 80% of the indices to create the training data and labels. Use the remaining 20% to create the test data and labels. Store them in the variables `train_data`, `train_label`, `test_data`, `test_label`.

```
import numpy as np
#for the following statement to compile successfully, you need the scikit-learn
    package.
#You can install it using pip install -U scikit-learn or conda install
    scikit-learn

from sklearn.datasets import load_iris
```

```
iris = load_iris()

#check the shape of iris data
print(iris.data.shape)

A = iris.data


#check the shape of iris target
print(iris.target.shape)

#How many labels does iris data have?
#C=num_of_classes
#print(C)


n = iris.data.shape[0] #Number of data points
d = iris.data.shape[1] #Dimension of data points


#In the following code, we create a nx1 vector of target labels
y = 1.0*np.ones([A.shape[0],1])
for i in range(iris.target.shape[0]):
   # y[i] = ???? # Convert the classes 2,3,...,C to -1

#Create an index array
indexarr = np.arange(n) #index array
np.random.shuffle(indexarr) #shuffle the indices

#Use the first 80% of indexarr to create the train data and the remaining 20% to
    create the test data

#train_data = ????
#train_label = ????

#test_data = ????
#test_label = ????
```

(d) Write a python function which implements the prediction rule in eqn. (2). Use the following code template.

```
def predict(w,x):
  #return ???
```

(e) Write a python function which takes as input the model parameter w, data features and labels and returns the accuracy on the data. (Use the predict function).

```
def compute_accuracy(data,labels,model_w):
  #Use predict function defined above
  #return ???
```

Exercise 2: An Optimization Algorithm

1. Note that problem (1) can be written as

$$\min_x f(w) = \min_w \sum_{i=1}^{n} f_i(w). \tag{3}$$

   Find an appropriate choice of $f_i(w)$.

2. Consider the loss function L_h. Write a python module to compute the loss function L_h.

```
def compute_loss_h(w,x,y):
  #return ???
```

3. Write a python routine to compute the objective function value. Use the `compute_loss` function.

```
def compute_objfnval(data,labels,model_w):
  #return ???
```

4. Write an expression to compute the gradient (or sub-gradient) of $f_i(w)$ for the loss function L_h. Denote the gradient by $g_i(w) = \nabla_w f_i(w)$. Define a python function to compute the gradient.

```
def compute_grad_loss_h(x,y,model_w):
  #return ???
```

5. Write an optimization algorithm where you pass through the training samples one by one and do the (sub-)gradient updates for each sample. Recall that this is similar to ALG-LAB7. Use the following template.

```
def OPT1(data,label,lambda, num_epochs):
  t = 1
  #initialize w
  #w = ???
  arr = np.arange(data.shape[0])
  for epoch in range(num_epochs):
    np.random.shuffle(arr) #shuffle every epoch
    for i in np.nditer(arr): #Pass through the data points
      # step = ???
      # Update w using w <- w - step * g_i (w)
      t = t+1
  return w
```

6. In OPT1, use `num_epochs`=500, `step`=$\frac{1}{t}$. For each $\lambda \in \{10^{-3}, 0.1, 1, 10\}$, perform the following tasks:

   (a) Plot the objective function value every 10 epochs. Use different colors for different λ values.

   (b) Plot the test set accuracy every 10 epochs. Use different colors for different λ values.

   (c) Plot the train set accuracy every 10 epochs. Use different colors for different λ values.

(d) Tabulate the final test set accuracy and train set accuracy for each λ value.

(e) Explain your observations.

7. Note that in OPT1, a fixed number of epochs is used. Can you think of some other suitable stopping criterion for terminating OPT1? Implement your stopping criterion and check how it differs from the one in OPT1. Use `step`$=\frac{1}{t}$ and λ which achieved the best test set accuracy in the previous experiment.

8. Repeat the experiments for different loss functions L_ℓ and L_{sh}. Explain your observations.

Exercise 3: Covertype data

1. Repeat Exercise 1 and 2 for covertype dataset. Use the following code to load covertype data. For covertype data, take the majority class as +1 and the remaining as -1.

```
import numpy as np
#for the following statement to compile successfully, you need the scikit-learn
    package.
#You can install it using pip install -U scikit-learn or conda install scikit-learn

from sklearn.datasets import fetch_covtype

covertype = fetch_covtype()
```