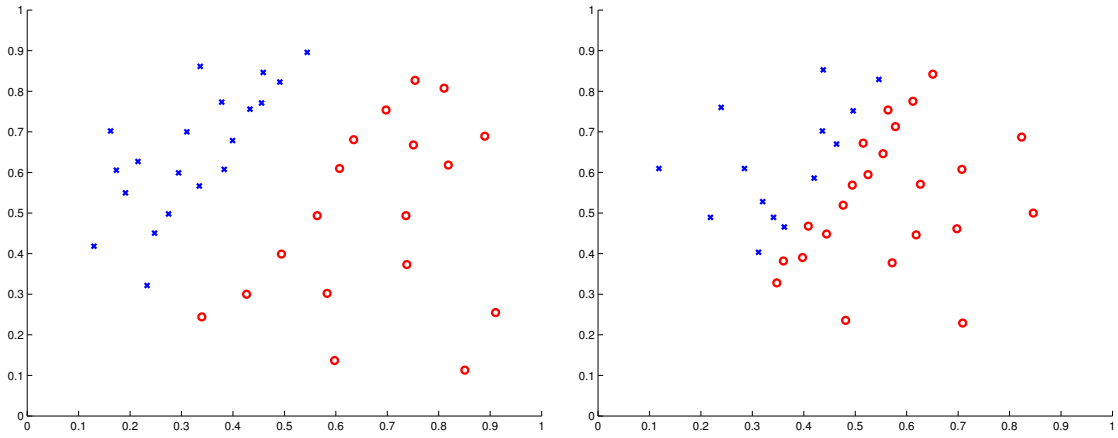
George Lydakis <lydakis@vision.rwth-aachen.de>
Idil Esen Zulfikar <zulfikar@vision.rwth-aachen.de>

Exercise 3: SVM, AdaBoost

due **before** 2022-12-08

Important information regarding the exercises:

- The exercises are not mandatory. Still, we strongly encourage you to solve them! All submissions will be corrected. If you submit your solution, please read on:
- Use the Moodle system to submit your solution. You will also find your corrections there.
- Due to the large number of participants, we require you to submit your solution to Moodle **in groups of 3 to 4 students**. You can use the **Discussion Forum** on Moodle to organize groups.
- If applicable submit your code solution as a zip/tar.gz file named `mn1_mn2_mn3.{zip/tar.gz}` with your **matriculation numbers** (`mn`).
- Please do **not** include the data files in your submission!
- Please upload your pen & paper problems as PDF. Alternatively, you can also take pictures (.png or .jpeg) of your hand written solutions. Please make sure your handwriting is legible, the pictures are not blurred and taken under appropriate lighting conditions. All non-readable submissions will be discarded immediately.



(a) Training data (b) Test data

Figure 1: Datasets for linear classifier

Question 1: Support Vector Machine ..... $(\Sigma = 4 + 2)$

Classify the 2D training data (c.f. Figure 1a) with a linear classifier using a linear SVM with a soft margin. The data for this question are stored in the files `lc_train_data.dat`, `lc_train_label.dat`, `lc_test_data.dat` and `lc_test_label.dat`.
To this end, implement the SVM learning algorithm in the function

```
1 def svmlin(X, t, C):
2     # Linear SVM Classifier
3     #
```

```
    # INPUT:
    # X     : the dataset           (num_samples x dim)
    # t     : labeling              (num_samples x 1)
    # C     : penalty factor for slack variables (scalar)
    #
    # OUTPUT:
    # alpha     : output of quadprog function  (num_samples x 1)
    # sv        : support vectors (boolean)    (1 x num_samples)
    # w         : parameters of the classifier (1 x dim)
    # b         : bias of the classifier       (scalar)
    # result    : result of classification     (1 x num_samples)
    # slack     : points inside the margin (boolean)  (1 x num_samples)
    return alpha, sv, w, b, result, slack
```

(a) To solve the dual task, use the function `a = cvxopt.solvers(P, q, G, h, A, b)` from the `cvxopt` package. This function requires formulating the quadratic criterion using matrices. The dual task **(2 pts)**

$$\mathbf{a} = \underset{\mathbf{a}}{\operatorname{argmax}} \left(\sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m \langle x_n, x_m \rangle \right)$$

under conditions

$$0 \leq a_n \leq C \qquad n = 1, 2, \ldots, N, \tag{1}$$

$$\sum_{n=1}^{N} a_n t_n = 0 \tag{2}$$

can be stated using matrices as:

$$\mathbf{a} = \underset{\mathbf{a}}{\operatorname{argmax}} \left(\mathbf{1}^\top \mathbf{a} - \frac{1}{2} \mathbf{a}^\top \mathbf{H} \mathbf{a} \right)$$

under conditions:

$$\mathbf{t}^T \mathbf{a} = 0, \tag{3}$$

$$\mathbf{0} \leq \mathbf{a} \leq \mathbf{C}, \tag{4}$$

where

- $\mathbf{a}$ is a vector $[N \times 1]$ of sought numbers (this is what we want to compute),
- $\mathbf{H}$ is a matrix $[N \times N]$ with elements $H(n, m) = t_n t_m \langle x_n, x_m \rangle$,
- $\mathbf{C}$ is a vector $[N \times 1]$ containing C,
- $\mathbf{t}$ is a vector $[1 \times N]$ containing class identifiers t_n,
- $\mathbf{1}$ is a vector $[N \times 1]$ of ones,
- $\mathbf{0}$ is a vector $[N \times 1]$ of zeros.

Now you can solve the above dual problem given the equality constraints using the function `a = cvxopt.solvers(P, q, G, h, A, b)` as follows:

- `q = (-1)* numpy.ones(N)`
- There are no inequality constraints, so `G = np.vstack([-np.eye(n), np.eye(n)])`, for `n = H.shape[1]`.
- For equality constraint as in Equations 2 and 3, `A = t` and `b = 0`.
- Finally, the lower `LB` and upper `UB` bounds for the Lagrange multipliers as in Equations 1 and 4, can be presented as: `h = np.hstack([-LB, UB])`.

Note that you need to provide the inputs as dense matrices in the `cvxopt` package, i.e. by using the command `cvxopt.matrix(...)`.

(b) Create the function **(1 pt)**

```
def svmlin(X, t, C):
    # Linear SVM Classifier
    #
    # INPUT:
    # X    : the dataset          (num_samples x dim)
    # t    : labeling             (num_samples x 1)
    # C    : penalty factor for slack variables (scalar)
    #
    # OUTPUT:
    # alpha    : output of quadprog function  (num_samples x 1)
    # sv       : support vectors (boolean)    (1 x num_samples)
    # w        : parameters of the classifier (1 x dim)
    # b        : bias of the classifier       (scalar)
    # result   : result of classification     (1 x num_samples)
    # slack    : points inside the margin (boolean)  (1 x
       num_samples)
    return alpha, sv, w, b, result, slack
```

to train and validate an SVM classifier on the digits 1 and 3 of the USPS dataset provided in the data. Each dataset consists of a training set and a test set. Each row of the matrices is an image of size 16×16. Provide a table showing the training and test errors of your SVM classifier. Additionally give the number of support vectors and the margin that you obtain for a few values of C.

(c) Do the same as in Part b, but for the digits 3 and 8. Are the test errors obtained in Part c similar? If not, give a short explanation why this is not the case? **(1 pt)**

(d) Implement a function **(2 bonus)**

```
def svmkern(X, t, C, p):
    # Non-Linear SVM Classifier
    #
    # INPUT:
    # X         : the dataset            (num_samples x dim)
    # t         : labeling               (num_samples x 1)
    # C         : penalty factor the slack variables (scalar)
    # p         : order of the polynom           (scalar)
    #
    # OUTPUT:
    # sv        : support vectors (boolean)      (1 x num_samples)
    # b         : bias of the classifier         (scalar)
    # slack     : points inside the margin (boolean) (1 x
       num_samples)
    return alpha, sv, b, result, slack
```

that performs classification of the test data using a second-order polynomial SVM. Do this by modifying the program in the previous Part a. A polynomial kernel function $\text{kern}(x_n, x_m)$ is provided for you in Python. You will need to modify the calculation of the matrix $\mathbf{H}$ by replacing the dot product $x_n^\top x_m$ by the kernel function $\text{kern}(x_n, x_m)$. Use the training data also as test data to see the difference between linear and nonlinear operation.

Question 2 [optional]: SVMlight and LibSVM ..... **(Bonus = 4)**

In practice, you will not have to solve the quadratic programming problem yourselves when using SVM's. There exist a number of freely available packages which employ highly optimized algorithms for this tasks, e.g. `SVMlight` (`http://svmlight.joachims.org/`) or `libSVM` (`http://www.csie.ntu.edu.tw/~cjlin/libsvm/`). You can try to use `SVMlight` or `libSVM` in order to solve the recognition task from the SVM question. Describe the steps of your solution and provide results and possible scripts/programs you have implemented.

Question 3: AdaBoost ..... **($\Sigma = 9 + 3$)**

In this question you will implement AdaBoost and apply it to a synthetic dataset as well as a real dataset. The goal is to first implement the algorithm around a very simple decision stub classifier. In a later step this will be replaced by a least squares classifier. To test your implementation, use the provided `apply` script.

A quick recap of the AdaBoost algorithm: Given a weak classifier and training samples (x_i, y_i), $i = 1 \dots N$ (where $x_i \in \mathbb{R}^d$ are data points in d dimensional space and $y_i \in \{-1, +1\}$ are class labels), the AdaBoost algorithm for the two-class problem is:

1. Initialize the weights $w_i^1 = \frac{1}{N}$
2. For $k = 1 \dots K$
   (a) Train the weak classifier c_k on the weighted data i.e. select the optimal parameter combination
   (b) Compute the error: $e_k = \frac{\sum_{i=1}^N w_i^k \mathbf{I}\{y_i \neq c_k(x_i)\}}{\sum_{i=1}^N w_i^k}$
   (c) Compute the voting weight for the weak classifier: $\alpha_k = \frac{1}{2} ln\left(\frac{1-e_k}{e_k}\right)$
   (d) Recalculate the weights: $w_i^{k+1} = w_i^k \exp\left(-\alpha_k y_i c_k(x_i)\right)$ and normalize w such that it is a probability distribution
3. Resulting classifier: $c_K(\mathbf{x}) = sgn\left(\sum_{k=1}^K \alpha_k c_k(\mathbf{x})\right)$;

Note: In the code archive you will find apply routines to check your implementation and visualize the classification result. This is meant as an assistance to you.

(a) When AdaBoost is applied to a weak learner with a linear decision boundary, what is the form of the resulting decision boundary of the boosted classifier? **(1 pt)**

(b) As you know from the lecture, the foundation of the Adaboost algorithm are the so-called weak classifiers. For a start we are going to use extremely simple decision stub classifiers as weak classifiers. **(2 pts)**

$$c(\mathbf{x}|j, \theta, p) := \begin{cases} 1 & \text{if } px_j > p\theta \\ -1 & \text{otherwise,} \end{cases}$$

where $p \in \{-1, 1\}$ is a parity indicating the direction of the inequality sign. This classifier ignores all entries of $\mathbf{x}$ those in dimension j, x_j. To train this weak classifier on weighted data, we use the following learning rule:

$$(\hat{j}, \hat{\theta}) := \text{argmin}_{j,\theta} \frac{\sum_{i=1}^n w_i \mathbf{I}\{y_i \neq c(\mathbf{x}|j, \theta, p)\}}{\sum_{i=1}^n w_i}$$

Write the function

```
1 def simpleClassifier(X, Y):
2     # Select a simple classifier
```

```
    #
    # INPUT:
    # X         : training examples (numSamples x numDim)
    # Y         : training lables (numSamples x 1)
    #
    # OUTPUT:
    # theta     : threshold value for the decision (scalar)
    # j         : the dimension to "look at" (scalar)
    return [j, theta]
```

implementing the above selection routine.

Note: You do not need the w_i term at this stage, because we simply assume that the provided training data points are sampled according to their weights.

This file loads the provided 2D synthetic data. You can generate your own synthetic 2D datasets using `createDataSet`.

(c) As a next step, implement the AdaBoost algorithm in **(4 pts)**

```
def adaboostSimple(X, Y, K, nSamples):
    # Adaboost with decision stump classifier as weak classifier
    #
    # INPUT:
    # X         : training examples (numSamples x numDim)
    # Y         : training lables (numSamples x 1)
    # K         : number of weak classifiers to select (scalar)
    #             (the _maximal_ iteration count - possibly abort
    #             earlier when error is zero)
    # nSamples  : number of training examples which are selected
    #             in each round (scalar)
    #             The sampling needs to be weighted!
    #             Hint - look at the function 'choice' in package
    #             numpy.random
    #
    # OUTPUT:
    # alphaK : voting weights (K x 1) - for each round
    # para   : parameters of simple classifier (K x 2) - for each
    #          round
    #        : dimension 1 is j
    #        : dimension 2 is theta
    return [alphaK, para]
```

In this first implementation of AdaBoost you should use the simple weak classifiers of the previous sub-question. Evaluate the current boosting classifier in the function `eval_adaBoost_simpleClassifier`.

```
def eval_adaBoost_simpleClassifier(X, alphaK, para):
    # INPUT:
    # para      : parameters of simple classifier (K x 2) - for
       each
    #             round
    #           : dimension 1 is j
    #           : dimension 2 is theta
    # alphaK    : classifier voting weights (K x 1)
    # X         : test data points (numSamples x numDim)
    #
    # OUTPUT:
```

```
    # classLabels : labels for data points (numSamples x 1)
    # result      : weighted sum of all the K classifier
    #               (numSamples x 1)
    return [classLabels, result]
```

(d) Next, we are going to add a cross-validation step to the training procedure. This means that only a part of the available training data is used for actual training. The remaining part is used to estimate the generalization performance of the learned classifier. To this end, split the training data according to the given percentage (percent = 0.4 means 40% is used for **validation**). After each iteration k of the algorithm, estimate the classification error of the current boosting classifier (not the base classifier) by cross-validation. Write your code in **(2 pts)**

```
def adaboostCross(X, Y, K, nSamples, percent):
    # Adaboost with an additional cross validation routine
    #
    # INPUT:
    # X          : training examples (numSamples x numDims )
    # Y          : training lables (numSamples x 1)
    # K          : number of weak classifiers to select (scalar)
    #              (the _maximal_ iteration count - possibly abort
    #              earlier)
    # nSamples   : number of training examples which are selected
    #              in each round. (scalar)
    #              The sampling needs to be weighted!
    # percent    : percentage of the data set that is used as test
    #              data set (scalar)
    #
    # OUTPUT:
    # alphaK     : voting weights (K x 1)
    # para       : parameters of simple classifier (K x 2)
    # testX      : test dataset (numTestSamples x numDim)
    # testY      : test labels  (numTestSamples x 1)
    # error      : error rate on validation set after each of the
    #              K iterations (K x 1)
    return [alphaK, para, testX, testY, error]
```

Plot the cross validation error estimates vs. the number k of iteration. What do you observe?

(e) Now, we are going to use a more complex weak classifier in the AdaBoost framework, namely the least squares classifier (already implemented in `leastSquare`). Implement the AdaBoost algorithm in **(2 bonus)**

```
def adaboostLSLC(X, Y, K, nSamples):
    # Adaboost with least squares linear classifier as weak
    # classifier
    # for a D-dim dataset
    #
    # INPUT:
    # X          : the dataset (numSamples x numDim)
    # Y          : labeling    (numSamples x 1)
    # K          : number of weak classifiers (iteration number
    #              of Adaboost) (scalar)
    # nSamples   : number of data which are weighted sampled
    #              (scalar)
```

```
    #
    # OUTPUT:
    # alphaK    : voting weights (K x 1)
    # para      : parameters of least square classifier (K x 3)
    # For a D-dim dataset each least square classifier has D+1
    # parameters w0, w1, w2........wD
    return [alphaK, para]
```

using said least squares classifier. Write the code to evaluate the current boosting classifier in the function `eval_adaBoost_leastSquare`.

```
def eval_adaBoost_leastSquare(X, alphaK, para):
    # INPUT:
    # para      : parameters of simple classifier (K x (D +1))
    #           : dimension 1 is w0
    #           : dimension 2 is w1
    #           : dimension 3 is w2
    #             and so on
    # alphaK    : classifier voting weights (K x 1)
    # X         : test data points (numSamples x numDim)
    #
    # OUTPUT:
    # classLabels: labels for data points (numSamples x 1)
    # result     : weighted sum of all the K classifier (scalar)
    return [classLabels, result]
```

Compare the classification performance of this classifier learned by AdaBoost with the one in c. State in one or two lines which classifier is better and why.

(f) Now, you are going to use the least-squares based AdaBoost on real data, i.e. the USPS data (provided in `usps.mat`). The dataset consists of a matrix `X` and a label vector `Y`. Each row of the matrix `X` is an image of size 20×14 and can be viewed with `matplotlib.pyplot.imshow(X[0,:].reshape(20,14,order='F').copy())`. The first 5000 rows of X contain the images of the digit 2, and the rest contains the images of the digit 9. Perform a random split of the 10000 data points into two equally sized subsets, one for training and one for validation. Run this at least three times and plot the cross validation error estimates vs. the number k of iterations. Implement your code in **(1 bonus)**

```
def adaboostUSPS(X, Y, K, nSamples, percent):
    # Adaboost with least squares linear classifier as weak
    # classifier on USPS data
    # for a high dimensional dataset
    #
    # INPUT:
    # X         : the dataset (numSamples x numDim)
    # Y         : labeling    (numSamples x 1)
    # K         : number of weak classifiers (scalar)
    # nSamples  : number of data points obtained by weighted
    #             sampling (scalar)
    #
    # OUTPUT:
    # alphaK    : voting weights (1 x k)
    # para      : parameters of simple classifier (K x (D+1))
    #             For a D-dim dataset each simple classifier has
```

```
17      #              D+1 parameters
18      # error      : training error (1 x k)
19      return [alphaK, para, error]
```