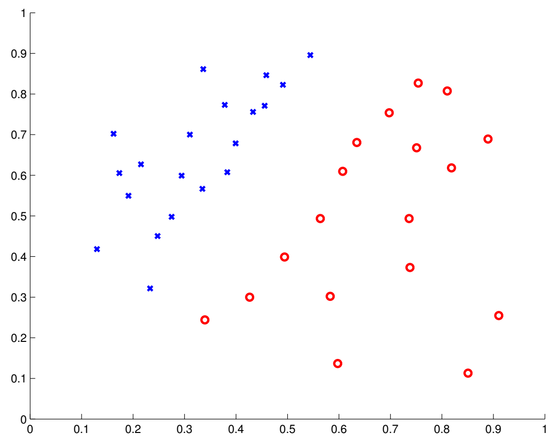
Paul Voigtlaender <voigtlaender@vision.rwth-aachen.de>
Sabarinath Mahadevan <mahadevan@vision.rwth-aachen.de>

Exercise 2: Least Square Linear Classifiers, SVM

due **before** 2018-11-22

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 L^{2}P 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 L^{2}P **in groups of 3 to 4 students**. You can use the **Discussion Forum** on L^{2}P 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

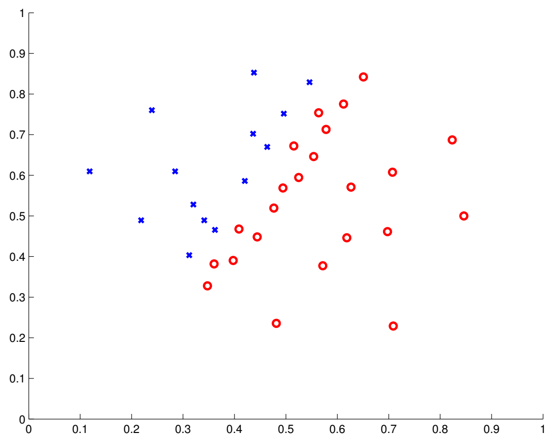

(b) Test data

Figure 1: Datasets for linear classifier

Question 1: Least Square Linear Classifier . . . . ($\Sigma = 5$)

Train a least-squares linear classifier on the 2D training data (c.f. Figure 1a) and test it on the training and test set (c.f. Figure 1b). The data for this exercise 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 write two functions:

(a) The function **(2 pts)**

```
def leastSquares(data, label):
    # Sum of squared error shoud be minimized
    #
    # INPUT:
    # data        : Training inputs  (num_samples x dim)
    # label       : Training targets (num_samples x 1)
    #
    # OUTPUT:
    # weights     : weights   (dim x 1)
    # bias        : bias term (scalar)
    return weight, bias
```

that trains a least-squares classifier based on a data matrix `data` and its class label vector `label`. It provides as output the linear classifier weight vector `weight` and bias `bias`.

(b) The function **(2 pts)**

```
def linclass(weight, bias, data):
    # Linear Classifier
    #
    # INPUT:
    # weight      : weights                 (dim x 1)
    # bias        : bias term               (scalar)
    # data        : Input to be classified (num_samples x dim)
    #
    # OUTPUT:
    # class_pred  : Predicted class (+-1) values  (num_samples x
       1)
    return class_pred
```

that classifies a data matrix `data` based on a trained linear classifier `weight, bias`.

(c) Run the script `apply`. This function loads the train and the test datasets. It first trains the linear classifier on the training data and then applies it on both the training and the test datasets. Analyze the classification plots for both the datasets. Are the sets optimally classified? Explain! **(1 pt)**

Question 2: Support Vector Machine . . . . $\left(\boldsymbol{\Sigma = 4 + 2}\right)$

Classify the same data as for the linear classifier using a linear SVM with a soft margin. To this end, implement the SVM learning algorithm in the function

```
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
```

(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

$$0 \le a_n \le C \qquad n = 1, 2, \dots, N, \tag{1}$$

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

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:

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

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

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)**

```python
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)**

```python
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 3 [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.