Open your mind. LUT.
Lappeenranta University of Technology

BM40A0700 Pattern Recognition

Lasse Lensu

Exercise 8: Common linear and nonlinear classifiers

1. Linear support vector machine (SVM) (2 points): Construct a Matlab function

   ```
   [w, w0] = svm(traindata, trainclass, C)
   ```

   that determines the linear SVM from the training data. `traindata` and `trainclass` are the training data set and its corresponding classes. `C` is the parameter that controls the penalty associated with an incorrect classification.

   The problem is solved by first optimizing

   $$Q(\boldsymbol{\lambda}) = \sum_{k=1}^{N} \lambda_k - \frac{1}{2} \sum_{k=1}^{N} \sum_{j=1}^{N} \lambda_k \lambda_j y_k y_j \mathbf{x}_k^T \mathbf{x}_j \tag{1}$$

   given constraints

   $$0 \leq \lambda_k \leq C, \quad \sum_{k=1}^{N} \lambda_k y_k = 0 \tag{2}$$

   After that, `w` and `w0` can be calculated using

   $$\begin{aligned} \mathbf{w} &= \sum_{k=1}^{N} \lambda_k y_k \mathbf{x}_k \\ w_0 &= 1 - \mathbf{x}_k^T \mathbf{w}, \quad \mathbf{x}_k \in \omega_1, \lambda_k > 0 \end{aligned} \tag{3}$$

   Note that in solving `w0`, any support vector belonging to class 1 can be used as x_k in Eq. 3.

   The optimization can be performed by Matlab Optimization toolbox function

   ```
   lambda = quadprog(H, f, A, b, Aeq, beq, LB, UB);
   ```

   which solves quadratic optimization problems with constraints. Before using `quadprog`, the optimized function must be stated in a matrix form. `quadprog` solves optimization problems of the form

   $$\arg\min_{\boldsymbol{\lambda}} \frac{1}{2} \boldsymbol{\lambda}^T \mathbf{H} \boldsymbol{\lambda} + \mathbf{f}^T \boldsymbol{\lambda}$$

   (see `help quadprog`). Fortunately this is easy, as $\mathbf{H}$ is an $N \times N$ matrix with elements $h_{ij} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ where $\mathbf{x}_i$ and y_i are training sample i and its class (1 for class 1, -1 for class 2). Also, $\mathbf{f}$ is now a vector of as many -1's as there are training samples, that is, $\mathbf{f} = (-1, -1, \ldots, -1)^T$. Note that the signs of $\mathbf{H}$ and $\mathbf{f}$ are different than in Eq. 1 because Matlab's `quadprog` minimizes the function, and we want to find the maximum.

   The constraints (Eq. 3) to `quadprog` can be given as follows: The equality constraint can be written as $\mathbf{y}^T \boldsymbol{\lambda} = 0$, and thus for Matlab:

PO BOX 20, FI-53851 LAPPEENRANTA, FINLAND
E-mail: lasse.lensu@lut.fi

`Aeq = y'; beq = 0;`
where `y` is the vector of y_i. Finally, the lower and upper bounds for the Lagrange multipliers λ can be presented in Matlab as:
`LB = zeros(N, 1); UB = C * ones(N, 1);`
where `N` is the number of training samples. There are no inequality constraints, so `A` and `b` are empty, i.e.,
`A = [ ]; b = [ ];`.

Your program should perform the following:

(a) Determine `y` from the class information.

(b) Determine `H, f, A, b, Aeq, beq, LB, UB`.

(c) Call `quadprog` to solve `lambda`.

(d) Solve `w` and `w0` according to Eq. 3.

You can test your program using the provided three data sets. You can use value $C = 5$ for the penalty term. After running your algorithm, you can draw the data and the decision boundary using the following code fragment:

```
xmax = max(data(1, :)) + 1;
xmin = min(data(1, :)) - 1;
ymax = max(data(2, :)) + 1;
ymin = min(data(2, :)) - 1;
hold off;
plot(data(1, class == 1), data(2, class == 1), 'bx', ...
  data(1, class == 2), data(2, class == 2), 'ro');
axis([xmin xmax ymin ymax]);
hold on;
line([xmin xmax], [-(w(1) * xmin + w0) / w(2), -(w(1) * xmax + w0) / w(2)]);
```

Additional files: `data1.mat, data2.mat, data3.mat`.

2. Nonlinear SVM classification (1 point): Contruct a Matlab function

```
testclass = svm2(traindata, trainclass, testdata, C)
```

that performs classification of `testdata` using a second-order polynomial SVM. Do this by improving a linear SVM implementation. A second-order polynomial kernel function is provided for you in Matlab.

The modifications you will need are as follows:

(a) In calculating the matrix `H`, you need to replace the dot products $\mathbf{x}_i^T\mathbf{x}_j$ with the kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$.

(b) At the end of the program, after determining the support vectors (lambdas corresponding to each training sample), classify each sample in the test data set and return the classes. Classification for each sample is performed by calculating the discriminant function

$$1 + \sum_{k=1}^{N} (\lambda_k y_k K(\mathbf{x}_k, \mathbf{x}) - \lambda_k y_k K(\mathbf{x}_k, \mathbf{x}_i))$$

for each test sample $\mathbf{x}$, and assigning the sample to class 1 (2) if the result is positive (negative). In the above formula, $\mathbf{x}_i$ is any one of the support vectors belonging to class 1. Note that the above formula comes from combining the decision rule with

$$\begin{aligned} \mathbf{w} &= \sum_{k=1}^{N} \lambda_k y_k \mathbf{x}_k \\ w_0 &= 1 - \mathbf{x}_k^T \mathbf{w}, \quad \mathbf{x}_k \in \omega_1, \lambda_k > 0 \end{aligned} \tag{4}$$

and replacing the dot products by the kernel function.

You can use the training data also as test data to see the difference between linear and nonlinear operation.

Additional files: `data1.mat, data2.mat, data3.mat, Kern.m`.

3. K-nearest-neighbour (kNN) classifier (1 point): Construct a Matlab function `C=knn(trainclass,traindata,data,k)` that performs kNN classification. Matrix `traindata` contains training examples so that each column is a single example. Row vector `trainclass` contains the classes of the examples, so that element i of `trainclass` is the class of the example in column i of `traindata`. Matrix `data` contains samples to be classified, one in each column, and `k` is a parameter which tells number of nearest neighbors used in classification. The return value `C` is a row vector of classes and it should include one value for each column in `data`. Use Euclidean distance as distance measure.

   The algorithm for kNN classification can described as follows:

   1. `Find k nearest neigbors from the training set for a sample to be classified.`
   2. `Classify the sample to the class which has most training samples among the k nearest neighbors.`

   The algorithm has one problem and it is the situation when two or more classes have the same, greatest amount of samples among the k nearest neigbors. Then unambiguous classification is not possible according to the algorithm above. There are several ways to handle this case. For example, one can decrease the value of k by one until unambiguous classification is possible.

   Use the provided "Iris" data set. Load this data into Matlab using command `load irisdata`. After this operation you will have four matrices in Matlab: `traindata`, `trainclass`, `data`, and `dataclass`. Try different k values and determine which value gives the best classification result.

   Additional files: `irisdata.mat`.