Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Graphics Lab
ML Group

Prof. M. Gross / Prof. J. Buhmann

Remo Ziegler, Christian Voegeli, Daniel Cotting, Christian Sigg, Jens Keuchel

Solve before: July 04, 2006

Visual Computing
Perceptron, Support Vector Machines
Solution

General Remarks

It is not necessary to hand in your results. You will find an exemplary solution on the lecture's web page.

1) Classification applets

On the course homepage you can find a link to the Java applets used during the lecture. Try out the demos to explore the capabilities of the perceptron and SVM classifiers. Especially test different kernels and parameter settings for the SVM and note how the classification boundaries change.

2) Classification with SVM (from WRK exam 03/04)

Consider the following one-dimensional classification problem with six points given as vector-label pairs (x_i, y_i). There are two classes with labels -1 and $+1$.

i	x_i	y_i
1	-2	-1
2	-5	-1
3	-3	-1
4	3	+1
5	2	+1
6	5	+1

a) Based on the given data, derive a linear discriminant function for data points $x \in \mathbb{R}$ of the form

$$g(x) = w_0 + wx$$

with

$$g(x_i) < 0 \text{ for } y_i = -1$$
$$g(x_i) > 0 \text{ for } y_i = +1.$$

New data points x are then classified as $+1$ if $g(x) > 0$ and as -1 otherwise.
What is a feasible discriminant function? Is the solution unique?
Solution: Feasible solutions are e.g. all $g(x) = ax$ for $a > 0$. The solution is not unique.

b) One possibility to define a linear discriminant function $g(x)$ is to demand that the values of $g(x)$ for the training data should be close to the label values. This leads to a least squares problem of the following form:

$$\min_{w_0,w} \sum_i (g(x_i) - y_i)^2$$

Solve this problem analytically by computing the partial derivatives of the objective function $G(w_0, w) := \sum_i (g(x_i) - y_i)^2$ and setting them to zero.

Solution: One easily obtains:

$$\frac{\partial G(w_0, w)}{\partial w_0} = 2\sum_i (w_0 + wx_i - y_i) = 0$$

$$\Leftrightarrow w_0 = \frac{1}{n}(\overbrace{\sum_i y_i}^{=0} - w\overbrace{\sum_i x_i}^{=0}) = 0$$

and

$$\frac{\partial G(w_0, w)}{\partial w} = 2\sum_i (w_0 + wx_i - y_i)x_i = 0$$

$$\Leftrightarrow w = \frac{\sum_i y_i x_i}{\sum_i x_i^2} = \frac{5}{19}$$

Hence the optimal linear discriminant function becomes $g(x) = \frac{5}{19}x$.

c) Given the linear classifier from the previous question, how would you classify the new data points $x_1^* = 0.2$ and $x_2^* = -0.2$?

Solution: $g(x_1^* = 0.2) = \frac{1}{19} > 0 \rightarrow$ *class* $+1$, *and* $g(x_2^* = -0.2) = -\frac{1}{19} < 0 \rightarrow$ *class* -1.

d) Consider now an augmented training set with two additional data points $(x_7 = -1, y_7 = -1)$ and $(x_8 = 25, y_8 = +1)$. Describe the effect of both additional points on the resulting least-squares linear discriminant function. You don't need to explicitly compute the new decision function. How would you now classify the point $x^* = 0.2$?

Solution: x_8 *is a* **correctly labeled** *point* **far away** *from the decision boundary. The least-squares discriminant function tries to approximate the* y*-coordinates of* **all data points** *by minimizing the* **squared distances** *between* $g(x_i)$ *and* y_i*. The effect of* x_8 *on the decision boundary will be huge, since it disproportionately influences the squared error.*

The effect of x_7*, on the other hand, will be rather small, since its* x*-coordinate is of the same order of magnitude as those of the other samples in the class* -1*.*

As the "outlier" x_8 *pulls the decision boundary further away from* 0*, the point* $x^* = 0.2$ *now would be classified as* -1*. In fact, the new linear classifier becomes* $\tilde{g}(x) = 0.068x - 0.204$*, which even classifies the training point* $x_5 = 2$ *incorrectly.*

e) Let us return to the original training set $\{x_i, y_i\}_{i=1}^6$. Suppose you have successfully trained a (hard-margin) support vector machine (SVM) on this data set.

i) What would be the support vectors?

ii) What would be the corresponding classification rule?

Give reasons for your answers. You don't need to explicitly calculate the decision function.

Solution:

i) x_1 *and* x_5 *are the support vectors, since they are closest to the separating max-margin plane.*

ii) *The classification rule is the same as in the least-squares case: decide class* $+1$ *if* $x > 0$ *and class* -1 *else. This follows immediately from the symmetry of the support vectors around 0.*

f) Return now to the augmented training set with two additional data points $(x_7 = -1, y_7 = -1)$ and $(x_8 = 25, y_8 = +1)$.

i) Describe the effect of both additional data points on the resulting SVM classifier. You don't need to explicitly calculate the new decision function.

ii) What are now the support vectors?

iii) What is the new classification rule? What is the predicted class membership of two new data points $x_1^* = 0.2$ and $x_2^* = -0.2$?

iv) What variant of the SVM could reduce the influence of the additional data points (x_7, y_7) and (x_8, y_8)? Give reasons for your answer.

Solution:

i) x_8 *has no influence on the SVM, since it will not become a support vector. However,* x_7 *falls into the original margin, and will become a support vector. The decision boundary is shifted to positive* x*-values.*

ii) x_7 *and* x_5*.*

iii) *The decision boundary is in the middle of* x_7 *and* x_5*: decide class* $+1$ *if* $x > 0.5$ *and class* -1 *else. Both new data points are assigned to class* -1*.*

iv) *The soft margin variant may help in this case. It allows "violations" of the margin (i.e. some data points may be within the margin). So the sensitivity to* x_7 *is reduced.*

3) Perceptron

The goal of this exercise is to implement a perceptron in Matlab. Our implementation will use the homogeneous coordinate representation of vectors, i. e. vectors $\mathbf{x} \in \mathbb{R}^d$ are represented by $\mathbf{y} = (1, x_1, \ldots, x_d)^\top \in \mathbb{R}^{d+1}$. This representation turns affine hyperplanes $\{\mathbf{x} \in \mathbb{R}^d | \mathbf{w}^\top \mathbf{x} + w_0 = 0\}$ with normal vector $\mathbf{w} = (w_1, \ldots, w_d)^\top$ into hyperplanes through the origin of the form $\{\mathbf{y} \in \mathbb{R}^{d+1} | \mathbf{a}^\top \mathbf{y} = 0\}$, where $\mathbf{a} = (w_0, w_1, \ldots, w_d)^\top$ is the vector characterizing the hyperplane.

To work with a perceptron, we need a two-class set of linearly separable data. For this purpose, you can use the Matlab function `[S,C] = fakedata(a,n)`, which is available on the course homepage. It creates an artificial sample set of n data values in d-dimensional space that are separated by the provided hyperplane $\mathbf{a} \in \mathbb{R}^{d+1}$. The data is represented by the $n \times (d+1)$-matrix `S`, so each row is the (homogeneous coordinate) representation of a single data point. This matrix contains the data of both classes, so we additionally represent the corresponding class labels by a vector `C` of length n, with entries in $\{-1, +1\}$.

In order to visualize your results in 2D (for $d = 2$ or $d = 3$), you can use the Matlab function `visualize(S,C,a)`, which is also available on the course homepage.

a) To evaluate a perceptron solution (a hyperplane classifier trained by a perceptron algorithm), write a function `C=classify(S,a)`. `S` is a sample data set in homogeneous coordinates, and `a` is the perceptron weight vector (which will be returned by the perceptron training algorithm). `C` is a class label vector as described above.

Solution:

```
function C = classify(S, a)

% calc distance to hyperplane
C = S*a;

% and normalize labels
C = -1*(C<0)+(C>=0);
```

b) Implement two versions of the batch perceptron training algorithm (cf. slide 191):

- The fixed step size version (”fixed-increment single sample perceptron”).
- The variable step size version with $\eta(k) = \frac{1}{k}$, where k is the number of the current iteration.

Both functions should be of the form

```
[a,a_history] = perceptrain(S,C),
```

where `a` is the normal vector of the hyperplane. `a_history` is a matrix containing the history of the normal vector throughout the training run, i. e. a $(\text{number of iterations}) \times (d+1)$ matrix, the rows of which are the interim results for `a`.

A recommended sanity check is to train a perceptron on a `fakedata` sample, and to make sure that the returned hyperplane correctly classifies its own training data:

```
a_orig = [0 1 -0.5];
[S,C] = fakedata(a_orig, 30);
visualize(S,C,a_orig);
[a, a_history] = perceptrain(S, C);
for i=1:size(a_history,1)
  visualize(S,C,a_history(i,:));
end;
```

Hint: Don’t forget to normalize the data points first according to their labels!

Solution:

- *The fixed step size version:*

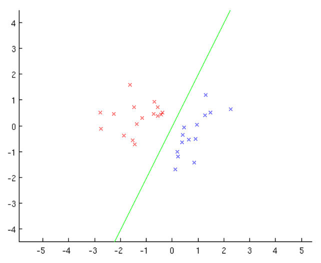
```
function [a, a_history] = perceptrain(S, C)

  % normalization of S
  [n,d] = size(S);
  S = S.*repmat(C,1,d);

  % initialization
  a = [0 ones(1,d-1)];
  a_history = [a];
  k = 0;

  % training
  E = ((S*a')<0);
  while (sum(E) > 0)
    k = mod(k,n)+1;
```

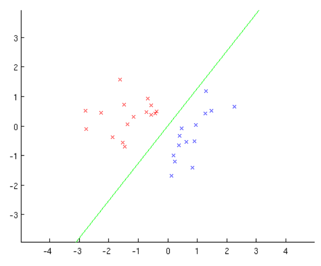
```
        if (S(k,:)*a' < 0)
          a = a + S(k,:);
          a_history = [a_history; a];
        end
        E = ((S*a')<0);
      end
```

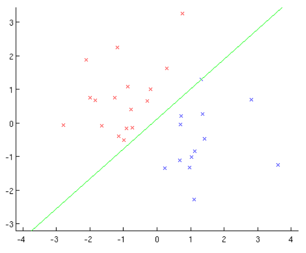
- *The variable step size version:*

```
    function [a, a_history] = perceptrain(S, C)

      % normalization of S
      [n,d] = size(S);
      S = S.*repmat(C,1,d);

      % initialization
      a = [0 ones(1,d-1)];
      a_history = [a];
      theta = 1e-5;
      k = 1;

      % training
      E = S(find((S*a')<0), :);  % collect misclassified data points
      while (norm(sum(E)/k)>theta)
        a = a + sum(E)/k;
        a_history = [a_history; a];
        k = k+1;
        E = S(find((S*a')<0), :);
      end
```

The example gives:


Original hyperplane


Classifier obtained with fixed step size training


Classifier obtained with variable step size training

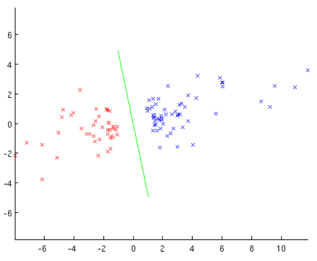
c) Generate a 2D random vector a, do a `[S,C]=fakedata(a,100)`, and train both your perceptrons on this set. Rerun `fakedata` to produce a test set, and check whether it is classified correctly.

Solution:

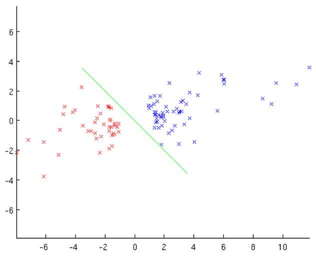
```
  a_orig = [0 rand rand];
  [S,C] = fakedata(a_orig, 100);
  visualize(S,C,a_orig);
  [a, a_history] = perceptrain(S, C);
```

```
visualize(S,C,a);
[S2,C2] = fakedata(a_orig, 100);
C3 = classify(S2, a);
visualize(S2,C3,a);
sum(C2-C3)     % should be zero!
```

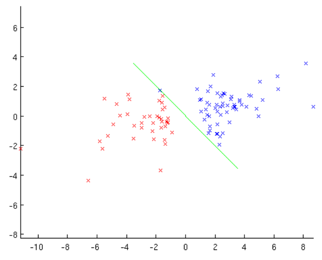
Possible solution, where 1 data point of the test set is misclassified:



Original hyperplane



Classifier obtained with fixed step size training



Classification of test data: 1 point is misclassified!