Pattern recognition systems – Laboratory 10

Linear Classifiers and the Perceptron Algorithm

1. Objectives

This laboratory session presents the perceptron learning algorithm for the linear classifier. We will apply gradient descent and stochastic gradient descent procedure to obtain the weight vector for a two-class classification problem.

2. Theoretical Background

The goal of classification is to group items that have similar features values into a single class or group. A linear classifier achieves this goal via a discriminant function that is the linear combination of the features.

Definitions

Define a training set as the tuple *(X,Y)*, where $X \in M_{n \times m}(R)$ and Y is a vector $Y \in M_{n \times 1}(D)$, where D is the set of class labels. X represents the concatenation the feature vectors for each sample from the training set, where each row is an m dimensional vector representing a sample. Y is the vector the desired outputs for the classifier. A classifier is a map from the feature space to the class labels: $f: R^m \rightarrow D$.

Thus a classifier partitions the feature space into |*D*| **decision regions.** The surface separating the classes is called **decision boundary.** If we have only two dimensional feature vectors the decision boundaries are lines or curves. In the following we will discuss binary classifiers. In this case the set of class labels contains exactly two elements. We will denote the labels for classes as *D={-1,1}*.

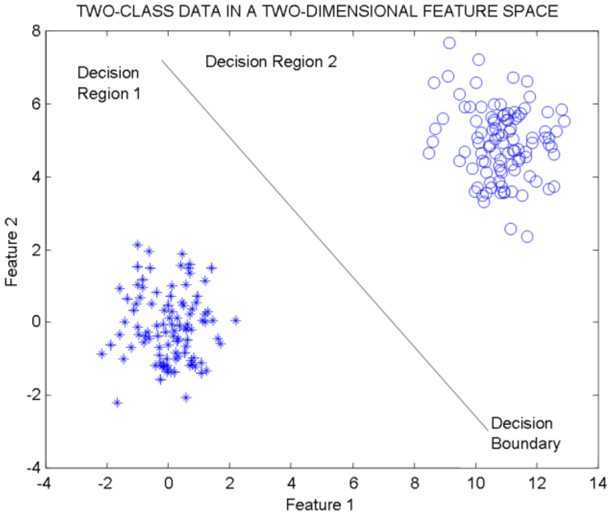


Figure 1. Example of linear classifier on a two-class classification problem. Each sample is characterized by two features.

2.1. General form of a linear classifier

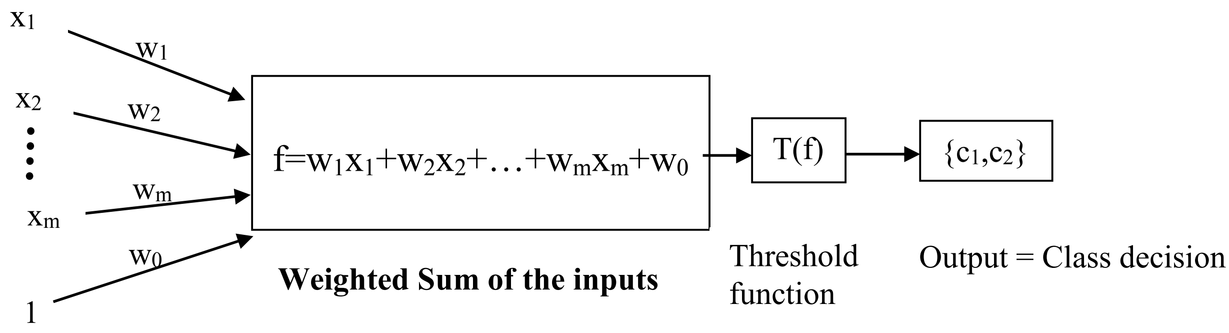
The simplest classifier is a linear classifier. A linear classifier outputs the class labels based on a linear combination of the input features. Considering $x \in M_{n\times 1}(R)$ as a feature vector we can write the linear decision function as:

$$\boxed{g(x) = wx^T + w_0 = \sum_{i=1}^{n} w_i x_i + w_0}$$

Where

- w is the **weight vector**
- w_0 is the **bias** or the **threshold weight**

A schematic view of the linear classifier is given in the next figure.



For convenience, we will absorb the intercept w_0 by augmenting the feature vector x with an additional constant dimension (let the bar over a variable denote the augmented version of the vector):

$$\boxed{wx^T + w_0 = \begin{bmatrix} w_0 & w \end{bmatrix} \begin{bmatrix} 1 \\ x^T \end{bmatrix} = \overline{w}\,\overline{x}^T}$$

A two-category linear classifier (or binary classifier) implements the following decision rule:

$$\boxed{\begin{cases} if\ \ g(x) > 0 \text{ decide that sample } x \text{ belongs to class } +1 \\ if\ \ g(x) < 0 \text{ decide that sample } x \text{ belongs to class } -1 \end{cases}}$$

or

$$\boxed{\begin{cases} if\ \ w^T x > -w_0 \text{ decide that sample } x \text{ belongs to class } +1 \\ if\ \ w^T x < -w_0 \text{ decide that sample } x \text{ belongs to class } -1 \end{cases}}$$

If $g(x) = 0$, x can ordinarily be assigned to either class.

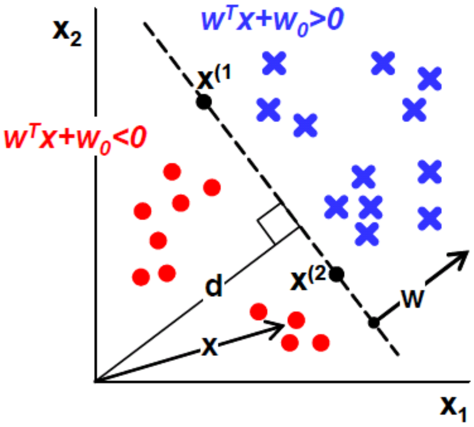


Figure 2. Image for 2D case depicting: linear decision regions (red and blue), decision boundary (dashed line), weight vector (w) and bias (w_0=d).

2.2. Learning algorithms for linear classifiers

We will present two main learning algorithms for linear classifiers. In order to perform learning we transform the task into an optimization problem. For this we define a loss functions L. The loss function applies a penalty for every instance that is classified into the wrong class. The perceptron algorithm adopts the following form for the loss function:

$$L(\overline{\boldsymbol{w}}) = \frac{1}{n}\sum_{i=1}^{n} max(0, -y_i\overline{\boldsymbol{w}} \cdot \overline{\boldsymbol{x}}_i^T) = \frac{1}{n}\sum_{i=1}^{n} L_i\,(\overline{\boldsymbol{w}})$$

If an instance is classified correctly, no penalty is applied because the second term is negative. In the case of a misclassification, the second (positive) term will be added to the function value. The objective now is to find the weights that minimize the loss function.

Gradient descent can be employed to find the global minimum of the loss function. This relies on the idea that a differentiable multivariate function decreases fastest in the opposite direction of the gradient. The update rule according to this observation is:

$$\overline{\boldsymbol{w}}_{k+1} \leftarrow \overline{\boldsymbol{w}}_k - \eta\boldsymbol{\nabla}\boldsymbol{L}(\overline{\boldsymbol{w}}_k)$$

where $\overline{\boldsymbol{w}}_k$ is the weight vector at time k, η is a parameter that controls the step size and is called the learning rate, and $\boldsymbol{\nabla}\boldsymbol{L}(\overline{\boldsymbol{w}})$ is the gradient vector of the loss function at point $\overline{\boldsymbol{w}}_k$. The gradient of the loss function is:

$$\boldsymbol{\nabla}\boldsymbol{L}(\overline{\boldsymbol{w}}) = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{\nabla}\boldsymbol{L}_{\boldsymbol{i}}(\overline{\boldsymbol{w}})$$

$$\boldsymbol{\nabla}\boldsymbol{L}_{\boldsymbol{i}}(\overline{\boldsymbol{w}}) = \begin{cases} 0, & if\ y_i\overline{\boldsymbol{w}} \cdot \overline{\boldsymbol{x}}_i^T > 0 \\ -y_i\overline{\boldsymbol{x}}_i, & otherwise \end{cases}$$

In the standard gradient descent approach we update the weights after visiting all the training examples. This is also called the batch-update learning algorithm. We can use stochastic gradient descent instead. This entails updating the weights after visiting each training example resulting in the classical online perceptron learning algorithm from [1]. In this case the update rule becomes:

$$\overline{\boldsymbol{w}}_{k+1} \leftarrow \overline{\boldsymbol{w}}_k - \eta\boldsymbol{\nabla}\boldsymbol{L}_{\boldsymbol{i}}(\overline{\boldsymbol{w}})$$

```
Algorithm:
Batch Perceptron

init w, η, E_limit, max_iter
for iter=1:max_iter
  E = 0, L = 0
  ∇L = [0,0,0]
  for i=1:n
    $z_i = \sum_{j=0}^{d} w_j X_{ij}$
    if $z_i \cdot y_i \leq 0$
      $\nabla \boldsymbol{L} \leftarrow \nabla \boldsymbol{L} - y_i \boldsymbol{X}_i$
      $E \leftarrow E + 1$
      $L \leftarrow L - y_i z_i$
    endif
  endfor
  $E \leftarrow E/n$
  $L \leftarrow L/n$
  $\nabla \boldsymbol{L} \leftarrow \nabla \boldsymbol{L}/n$
  if $E < E_{limit}$
    break
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla \boldsymbol{L}$
endfor
```

```
Algorithm:
Online Perceptron

init w, η, E_limit, max_iter
for iter=1:max_iter
  E = 0
  for i=1:n
    $z_i = \sum_{j=0}^{d} w_j X_{ij}$
    if $z_i \cdot y_i \leq 0$
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \boldsymbol{X}_i y_i$
      $E \leftarrow E + 1$
    endif
  endfor
  $E \leftarrow E/n$
  if $E < E_{limit}$
    break
endfor
```

3. Two-class two feature linear classifier

In this laboratory session we will find a linear classifier that discriminates between two sets of points. The points in class 1 are colored in red and the points in class 2 are colored in blue.

Each point is described by the color (that denotes the class label) and the two coordinates, x_1 and x_2.
The augmented weight vector will have the form $\overline{w} = [w_0\ w_1\ w_2]$.
The augmented feature vector will be $\bar{x} = [1\ \ x_1\ \ x_2\ ]$.

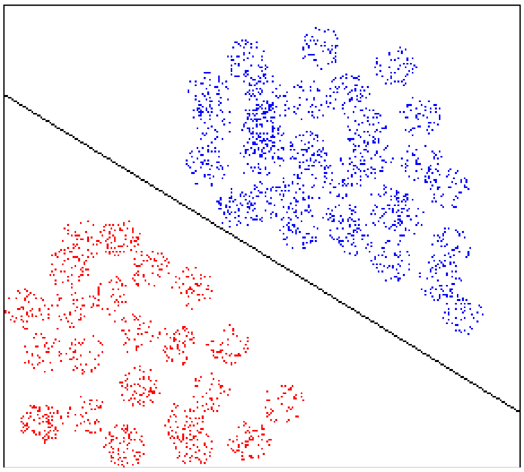
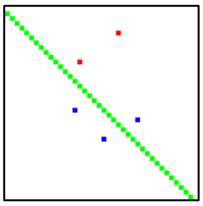


Figure 3. The decision boundary obtained from the perceptron algorithm

4. Numerical example

Consider the point from the file points00 as (x,y) pairs or (column, row):

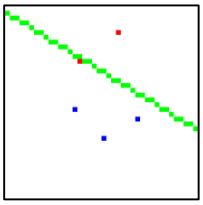
- Red points: (23, 5), (15,11) – class +1
- Blue points: (14, 21), (27,23), (20, 27) – class -1



Learning rate = 0.01

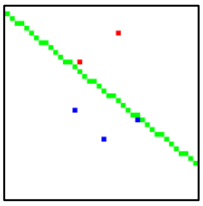
Iteration 0

```
i=0: w=[1.000000 1.000000 -1.000000] xi=[1 23 5] yi = 1 zi=19.000000
i=1: w=[1.000000 1.000000 -1.000000] xi=[1 15 11] yi = 1 zi=5.000000
i=2: w=[1.000000 1.000000 -1.000000] xi=[1 14 21] yi = -1 zi=-6.000000
i=3: w=[1.000000 1.000000 -1.000000] xi=[1 27 23] yi = -1 zi=5.000000 wrong
        update w0 = w0 - 0.01, w1 = w1 - 27*0.01, w2 = w2 – 23*0.01
i=4: w=[0.990000 0.730000 -1.230000] xi=[1 20 27] yi = -1 zi=-17.620000
```



Iteration 1

```
i=0: w= [0.990000 0.730000 -1.230000] xi= [1 23 5] yi = 1 zi=11.630000
i=1: w= [0.990000 0.730000 -1.230000] xi= [1 15 11] yi = 1 zi=-1.590000 wrong
        update w0 = w0 + 0.01, w1 = w1 + 15*0.01, w2 = w2 + 11*0.01
i=2: w= [1.000000 0.880000 -1.120000] xi= [1 14 21] yi = -1 zi=-10.200000
i=3: w= [1.000000 0.880000 -1.120000] xi= [1 27 23] yi = -1 zi=-1.000000
i=4: w= [1.000000 0.880000 -1.120000] xi= [1 20 27] yi = -1 zi=-11.640000
```



Iteration 2

```
i=0: w= [1.000000 0.880000 -1.120000] xi= [1 23 5] yi = 1 zi=15.640000
i=1: w= [1.000000 0.880000 -1.120000] xi= [1 15 11] yi = 1 zi=1.880000
i=2: w= [1.000000 0.880000 -1.120000] xi= [1 14 21] yi = -1 zi=-10.200000
i=3: w= [1.000000 0.880000 -1.120000] xi= [1 27 23] yi = -1 zi=-1.000000
i=4: w= [1.000000 0.880000 -1.120000] xi= [1 20 27] yi = -1 zi=-11.640000
All classified correctly
```

5. Practical work

1. Read the points from a single file *test0*.bmp* and construct the training set *(X,Y)*. Assign the class label +1 to red points and -1 to blue points.
2. Implement and apply the online perceptron algorithm to find the linear classifier that divides the points into two groups. Suggestion for parameters: $\eta=10^{-4}$, $w_0 = [0.1, 0.1, 0.1]$, $E_{limit}=10^{-5}$, $max\_iter = 10^5$.
   Note: for a faster convergence use a larger learning rate only for w_0
3. Draw the final decision boundary based on the weight vector **w**.
4. Implement the batch perceptron algorithm and find suitable parameters values. Show the loss function at each step. It must decrease slowly.
5. Visualize the decision boundary at intermediate steps, while the learning algorithm is running.
6. Change the starting values for the weight vector **w**, the learning rate and terminating conditions to observe what happens in each case. What does an oscillating cost function signal?

6. References

[1] Rosenblatt, Frank (1957), The Perceptron - a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

[2] Richard O. Duda, Peter E. Hart, David G. Stork: Pattern Classification 2nd ed.

[3] Xiaoli Z. Fern, Machine Learning Course, Oregon University - http://web.engr.oregonstate.edu/~xfern/classes/cs534/notes/perceptron-4-11.pdf

[4] Gradient Descent - http://en.wikipedia.org/wiki/Gradient_descent

[5] Avrim Blum, Machine Learning Theory, Carnegie Mellon University - https://www.cs.cmu.edu/~avrim/ML10/lect0125.pdf