Optimization Algorithms for Machine Learning

Stefano Gualandi
Università di Pavia, Dipartimento di Matematica

email: stefano.gualandi@unipv.it
twitter: @famo2spaghi
blog: http://stegua.github.com
web: http://matematica.unipv.it/gualandi/opt4ml

Supervised Learning vs. Unsupervised Learning

Definition 1 (Supervised Learning)

Supervised Learning is the task of learning (inferring) a function f that maps input vectors to their corresponding target vectors, by using a dataset containing a given set of pairs of $(input, output)$ samples. Examples:

- Regression: the output vectors take one or more continuous values.
- Classification: the output vectors take one value of a finite number of discrete categories. Special case: binary classification.

Supervised Learning vs. Unsupervised Learning

Definition 2 (Unsupervised Learning)

Unsupervised Learning is the task of learning (inferring) a function f that maps input vectors to their corresponding target vectors, but without any a priori knowledge about the correct mapping. Examples:

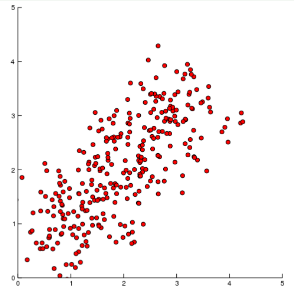
- CLUSTERING: The goal of clustering is to group or partition the input vectors (if possible) into k groups or clusters, with the vectors in each group close to each other. In this case, the input vectors represents usually features of objects.
- DENSITY ESTIMATION: The goal is to project the data from a high dimensional space down to two or three dimensions, usually for the purpose of *visualization*.

Unsupervised Learning: Clustering

Suppose we have n vectors: $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, where each $\boldsymbol{x}_i \in \mathbb{R}^d$.

The goal of clustering is to group or partition the vectors (if possible) into k groups or clusters (with $k << n$), with the vectors in each group close to each other.

Example 3 (Clustering $n = 300$ points in $\mathbb{R}^2$, into $k = 3$ clusters)

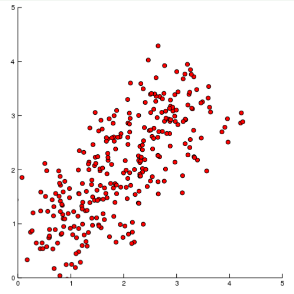


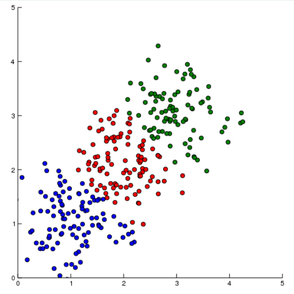
Unsupervised Learning: Clustering

Suppose we have n vectors: $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, where each $\boldsymbol{x}_i \in \mathbb{R}^d$.

The goal of clustering is to group or partition the vectors (if possible) into k groups or clusters (with $k << n$), with the vectors in each group close to each other.

Example 3 (Clustering $n = 300$ points in $\mathbb{R}^2$, into $k = 3$ clusters)





Specifying the Cluster Assignments

To specify a clustering or assignment of the n vectors, we used the labels $1, \ldots, k$ and a vector c of n elements, with the convention that $c_i = j$ means that the i-th vector belong to the j-th cluster.

Specifying the Cluster Assignments

To specify a clustering or assignment of the n vectors, we used the labels $1, \ldots, k$ and a vector c of n elements, with the convention that $c_i = j$ means that the i-th vector belong to the j-th cluster.

Example 4 (Cluster Assignment)

Suppose we have $n = 5$ vectors and $k = 3$ groups. If we are given the assignment vector $c = [3, 1, 1, 1, 2]$, this means that we have the following 3 groups:

$$G_1 = \{2, 3, 4\}, \quad G_2 = \{5\}, \quad G_3 = \{1\}$$

More compactly, we can write the grouping

$$G_j = \{i \mid c_i = j\}.$$

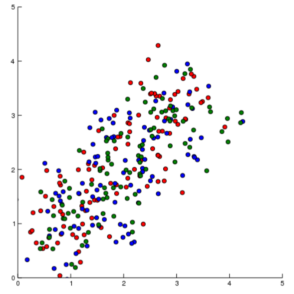
A cluster objective

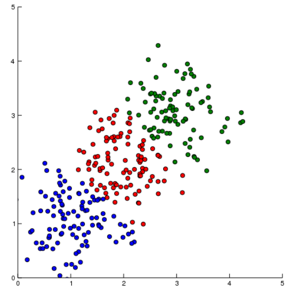
How can we evaluate a given choice of clustering?

A cluster objective

How can we evaluate a given choice of clustering?

Example 5 (Clustering $n = 300$ points in $\mathbb{R}^2$, into $k = 3$ clusters)





Group Representatives

How can we evaluate a given choice of clustering?

Group Representatives

How can we evaluate a given choice of clustering?

Within each cluster we select a

group representative n-vector denoted by: $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$

The representative can be any vector of $\mathbb{R}^d$.

DESIDERATA: each representative is as close as possible to the vector in its associated group. We want to keep as small as possible the quantities:

$$||\boldsymbol{x}_i - \boldsymbol{z}_{c_i}||$$

A Cluster Objective

Using the clustering representative, we can write a single function that measure the quality of a given clustering $\boldsymbol{c}$ with given representative $\boldsymbol{z}_i$:

$$L(\boldsymbol{c}, \boldsymbol{z}) = \frac{1}{n} \sum_{i=1}^{n} ||\boldsymbol{x_i} - \boldsymbol{z}_{c_i}||^2$$

which is the mean square distance from the vectors to their associated representatives. **NOTE:** other objective functions could be used.

OPTIMAL CLUSTERING: a choice of group assignment $c_1, \ldots, c_n$ and group representatives $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$ that minimize the objective $L(\boldsymbol{c}, \boldsymbol{z})$.

Optimal clustering with fixed representatives

In general, exact clustering is NP-Hard, already for $k = 2$.

Optimal clustering with fixed representatives

In general, exact clustering is NP-Hard, already for $k = 2$.
(... and hence, no hope for efficient scalable exact algorithms!)

Optimal clustering with fixed representatives

In general, exact clustering is NP-Hard, already for $k = 2$.
(... and hence, no hope for efficient scalable exact algorithms!)

However, given the representatives $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$, we can find the assignment vector $\boldsymbol{c}$ that achieve the smallest possible value of $L(\boldsymbol{x}, \boldsymbol{c}, \boldsymbol{z})$.

The choice of c_i only affects the term

$$\frac{1}{n} \left|\left|\boldsymbol{x_i} - \boldsymbol{z}_{c_i}\right|\right|^2$$

Optimal clustering with fixed representatives

In general, exact clustering is NP-Hard, already for $k = 2$.
(... and hence, no hope for efficient scalable exact algorithms!)

However, given the representatives $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$, we can find the assignment vector $\boldsymbol{c}$ that achieve the smallest possible value of $L(\boldsymbol{x}, \boldsymbol{c}, \boldsymbol{z})$.

The choice of c_i only affects the term

$$\frac{1}{n} ||\boldsymbol{x_i} - \boldsymbol{z}_{c_i}||^2$$

and, hence, we can select

$$c_i = j^* = \text{argmin}_{j=1,\ldots,k} ||\boldsymbol{x_i} - \boldsymbol{z}_j|| \tag{1}$$

Optimal clustering with fixed assignments

If we fix the assignment of points to the clusters, then it is possible to find the representatives that minimize the objective $L(\boldsymbol{c}, \boldsymbol{z})$. If we re-arrange the objective using the groups we obtain:

$$L(\boldsymbol{z}) = \frac{1}{n} \sum_{j=1}^{k} \sum_{i \in G_j} ||\boldsymbol{x_i} - \boldsymbol{z}_j||^2$$

Note that we want to find the representative $\boldsymbol{z}_j$ that minimized the j-th term. Thus we should choose the vector $\boldsymbol{z}_j \in \mathbb{R}^d$ that minimize the mean square distance to the vectors in groups j, that is, the average (or *mean*, or *centroid*):

$$z_j = \frac{1}{|G_j|} \sum_{i \in G_j} \boldsymbol{x}_i \tag{2}$$

Main Idea

Since we know:

1. How to compute $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$ given $\boldsymbol{c}$:

$$z_j = \frac{1}{|G_j|} \sum_{i \in G_j} \boldsymbol{x}_i$$

2. How to compute $\boldsymbol{c}$ given $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$:

$$c_i = j^* = \text{argmin}_{j=1,\ldots,k} ||\boldsymbol{x_i} - \boldsymbol{z}_j||$$

We start with any assignment of points to k clusters, and then we iterate until convergence.

Main Idea

Since we know:

1. How to compute $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$ given $\boldsymbol{c}$:

$$z_j = \frac{1}{|G_j|} \sum_{i \in G_j} \boldsymbol{x}_i$$

2. How to compute $\boldsymbol{c}$ given $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$:

$$c_i = j^* = \text{argmin}_{j=1,\ldots,k} ||\boldsymbol{x_i} - \boldsymbol{z}_j||$$

We start with any assignment of points to k clusters, and then we iterate until convergence.

QUESTION 1: How can we select an initial assignment vector $\boldsymbol{c}$?

Main Idea

Since we know:

1. How to compute $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$ given $\boldsymbol{c}$:

$$z_j = \frac{1}{|G_j|} \sum_{i \in G_j} \boldsymbol{x}_i$$

2. How to compute $\boldsymbol{c}$ given $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$:

$$c_i = j^* = \text{argmin}_{j=1,\ldots,k} ||\boldsymbol{x_i} - \boldsymbol{z}_j||$$

We start with any assignment of points to k clusters, and then we iterate until convergence.

QUESTION 1: How can we select an initial assignment vector $\boldsymbol{c}$?

QUESTION 2: When can we stop?

Main Idea

Since we know:

1. How to compute $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$ given $\boldsymbol{c}$:

$$z_j = \frac{1}{|G_j|} \sum_{i \in G_j} \boldsymbol{x}_i$$

2. How to compute $\boldsymbol{c}$ given $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$:

$$c_i = j^* = \text{argmin}_{j=1,\ldots,k} \, ||\boldsymbol{x_i} - \boldsymbol{z}_j||$$

We start with any assignment of points to k clusters, and then we iterate until convergence.

QUESTION 1: How can we select an initial assignment vector $\boldsymbol{c}$?

QUESTION 2: When can we stop?

QUESTION 3: How to choose k?

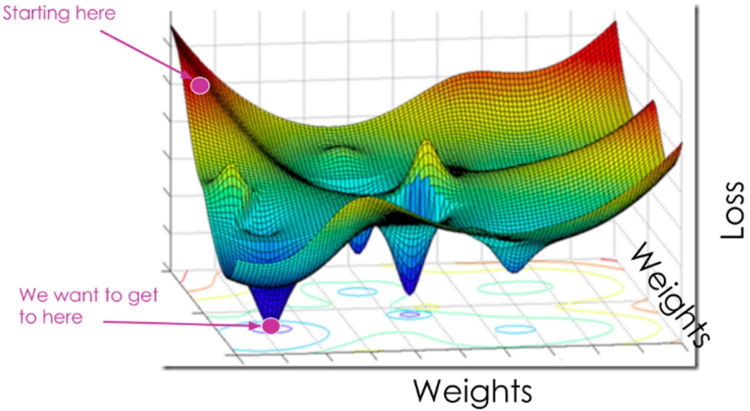
The Lloyd Algorithm, aka k-means algorithm

Algorithm 1: k-means algorithm

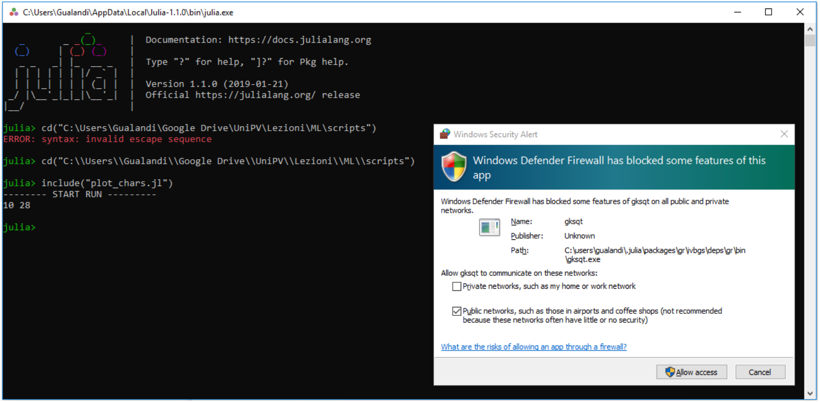
Data: $\boldsymbol{X} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ input vectors
Data: k number of clusters
Result: $\boldsymbol{c}$ clustering assignment
Result: $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$ clustering representatives

$\boldsymbol{c} \leftarrow \textsc{RandomAssignment}(\boldsymbol{X}, k)$;
for $i \leftarrow 1$ **to** *maxiter* **do**
    $\boldsymbol{c}_0 \leftarrow \boldsymbol{c}$;
    $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k \leftarrow \textsc{BestRepresentatives}(\boldsymbol{X}, \boldsymbol{c})$;
    $\boldsymbol{c} \leftarrow \textsc{BestAssignment}(\boldsymbol{X}, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_k)$;
    if $\boldsymbol{c}_0 = \boldsymbol{c}$ **then**
        break;
return $\boldsymbol{c}, (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k)$

Loss Function Landascape (micro-example)



Julia: Editor + Shell



Motivating Julia: Summing Numbers

Multple Dispatch: Run the right code at the right time!

Motivating Julia: Summing Numbers

Multple Dispatch: Run the right code at the right time!

Multiple Dispatch is the selection of a function implementation based on the types of each argument of the function.

Motivating Julia: Summing Numbers

Multple Dispatch: Run the right code at the right time!

Multiple Dispatch is the selection of a function implementation based on the types of each argument of the function.

Example 6 (Summing Numbers)

In Julia, we have several different **types** for representing numbers:
`Int8, Int16, Int32, Int64, Float8, Float16, Float32, Float64, ...`
With a high level programming language, with **dynamic types**, we can simply write:

```
julia> plus(x, y) = x + y
```

and then run:

```
julia> plus(1, 1), plus(1.0, 1.0), plus(1, 1.0)
```

However, at assembly level, different operations are performed!

Motivating Julia: Summing Numbers

Example 7 (Summing Numbers (con't))

We can simulate what happens at assembly level with the following code:

Motivating Julia: Summing Numbers

Example 7 (Summing Numbers (con't))

We can simulate what happens at assembly level with the following code:

```
julia> add(x::Int64, y::Int64) = x + y
```

Motivating Julia: Summing Numbers

Example 7 (Summing Numbers (con't))

We can simulate what happens at assembly level with the following code:

```
julia> add(x::Int64, y::Int64) = x + y

julia> vaddsd(x::Float64, y::Float64) = x + y
```

Motivating Julia: Summing Numbers

Example 7 (Summing Numbers (con't))

We can simulate what happens at assembly level with the following code:

```
julia> add(x::Int64, y::Int64) = x + y

julia> vaddsd(x::Float64, y::Float64) = x + y

julia> vcvtsi2sd(x::Int64) = float(x)
```

Motivating Julia: Summing Numbers

Example 7 (Summing Numbers (con't))

We can simulate what happens at assembly level with the following code:

```
julia> add(x::Int64, y::Int64) = x + y

julia> vaddsd(x::Float64, y::Float64) = x + y

julia> vcvtsi2sd(x::Int64) = float(x)
```

Using these functions, we can define:

```
julia> plus(x::Int64, y::Int64) = add(x, y)

julia> plus(x::Float64, y::Float64) = vaddsd(x, y)

julia> plus(x::Int64, y::Float64) = vaddsd(vcvtsi2sd(x), y)

julia> plus(x::Float64, y::Int64) = plus(y, x)
```

VADDSD: Vector ADD Scalar Double-precision,
VCVTSI2SD: Vector Convert Doubleword (Scalar) Integer to Scalar Double Precision Floating-Point value

Julia Just-in-time compilation 1/4

```
julia> @code_native plus(1,1)
        .text
;  @ summing.jl:9 within ‘plus’
        pushq   %rbp
        movq    %rsp, %rbp
;   @ summing.jl:4 within ‘add’
;    @ int.jl:53 within ‘+’
        leaq    (%rcx,%rdx), %rax
;
        popq    %rbp
        retq
        nopw    (%rax,%rax)
;
```

Julia Just-in-time compilation 2/4

```
julia> @code_native plus(1.0,1.0)
        .text
;    @ summing.jl:10 within `plus'
        pushq   %rbp
        movq    %rsp, %rbp
;     @ summing.jl:5 within `vaddsd'
;      @ float.jl:395 within `+'
        vaddsd  %xmm1, %xmm0, %xmm0
;
        popq    %rbp
        retq
        nopw    (%rax,%rax)
;
```

Julia Just-in-time compilation 3/4

```
julia> @code_native plus(1.0,1)
        .text
;   @ summing.jl:12 within ‘plus’
        pushq   %rbp
        movq    %rsp, %rbp
;   @ summing.jl:12 within ‘plus’ @ summing.jl:11
;    @ summing.jl:6 within ‘vcvtsi2sd’
;     @ float.jl:271 within ‘float’
;      @ float.jl:256 within ‘Type’ @ float.jl:60
        vcvtsi2sdq      %rdx, %xmm1, %xmm1
;
;   @ summing.jl:12 within ‘plus’ @ float.jl:395
         vaddsd  %xmm0, %xmm1, %xmm0
;   @ summing.jl:12 within ‘plus’
        popq    %rbp
        retq
        nop
;
```

Julia Just-in-time compilation 4/4

```
julia> @code_native g(10)
        .text
;  @ summing.jl:20 within ‘g’
        pushq   %rbp
        movq    %rsp, %rbp
;  @ summing.jl:21 within ‘g’
;   @ summing.jl:17 within ‘f’
;    @ int.jl:54 within ‘*’
        imulq   $9765625, %rcx, %rax    # imm = 0x9502F9
;
;   @ int.jl:52 within ‘f’
        addq    $-2441406, %rax         # imm = 0xFFDABF42
;
;  @ summing.jl:23 within ‘g’
        popq    %rbp
        retq
        nopw    %cs:(%rax,%rax)
```

Motivating Julia: Unrolling loops

Example 8 (Unrolling loops)

Consider what happens if we make a fixed number of iterations on an integer arguments:

```
# 10 iterations of F8k) ? 5*k - 1 on integers
f(x) = 5*x - 1

function g(k)
  for i = 1:10
    k = f(k)
  end
  return k
end
```

Julia Just-in-time compilation 4/4

```
julia> @code_native g(10)
        pushq   %rbp
        movq    %rsp, %rbp
        imulq   $9765625, %rcx, %rax     # imm = 0x9502F9
        addq    $-2441406, %rax          # imm = 0xFFDABF42
        popq    %rbp
        retq
        nopw    %cs:(%rax,%rax)
```

Because the compiler knows that integer addition and multiplication are associative and that multiplication distributes over addition, it can optimize the entire loop down to just a multiply and an add.

Indeed, if $f(k) = 5k - 1$, it is true that the tenfold iterate

$$f^{(10)}(k) = -2441406 + 9765625k$$

.

Exercise 1: k-means in $\mathbb{R}^2$

```
using Plots, Random, Distributions, LinearAlgebra

function MakeData(N, K)
  M = N/K
  Ls = Array{Float64,1}[]
  for k = 1:K
    for i = 1:M
      push!(Ls, [k,k]+0.5*randn(2) )
    end
  end
  return Ls
end
function RandomAssign(X, k)
  unif = DiscreteUniform(1, k)
  return [rand(unif) for x in X]
end
k = 3
X = MakeData(300, k)
C = RandomAssign(X, k)
```

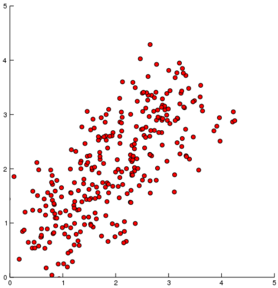
Exercise 1: k-means in $\mathbb{R}^2$

```
Partition(X,C)=[[X[i] for i=1:length(X) if C[i] == j]
                  for j=1:maximum(C)]

function PlotSingleCluster(X, C, n)
  clusters = Partition(X, C)
  CL = [:red, :blue, :green]
  plot(legend=false, grid=false, size=(500,500),
        xlims=(0,5), ylims=(0,5))
  for j = 1:maximum(C)
    scatter!([c[1] for c in clusters[j]],
             [c[2] for c in clusters[j]], c=CL[j])
  end
  p = plot!(legend=false, grid=false, size=(500,500),
            xlims=(0,5), ylims=(0,5))
  #png(p, "single$(n)")
end
k = 3
X = MakeData(300, k)
C = RandomAssign(X, k)
```

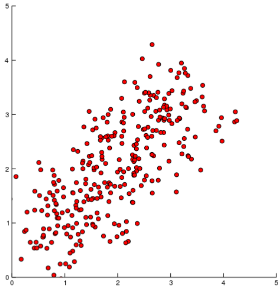
Exercise 1: Useful function

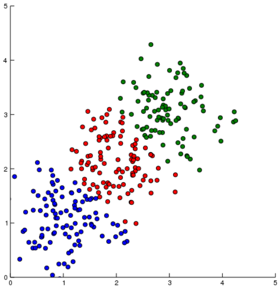
```
julia> length([4,6,7,8,9,3])
6

julia> sum([4,6,7,8,9,3])
37

julia> maximum([4,6,7,8,9,3])
9

julia> findmin([4,6,7,8,9,3])
(3, 6)

julia> norm([1,1,1,1])
2.0

julia> norm([2,1]-[1,0])
1.4142135623730951
```

Exercise 1: Possible results

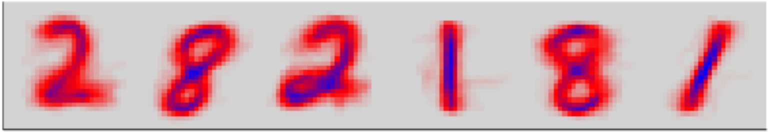


Exercise 1: Possible results





Exercise 2: *k*-means using MNSIT digits



Exercise 3: using the whole dataset...