highschool_svm

January 26, 2019

0.0.1  A naive and crude approximation of improper SVM using a high-school math

(c) Tomas Pevny, 2019

Start by importing some libraries and initiate plotting backend

In [1]: using Plots, StatsBase, Statistics

WARNING: using StatsBase.mode in module Main conflicts with an existing identifier.

Next, we generate some toy data, such as two gaussians and visualize them in scatter plot.

In [31]: x = randn(2, 100) .* [3, 1] .+ [5, 0]
   y = randn(2, 100) .* [1, 3] .+ [0, 5]
   X = hcat(x,y)
   Y = vcat(ones(Int,100),-ones(Int,100));
   d = size(x, 1);
   scatter(x[1,:), x[2,:], dpi = 500);
   scatter!(y[1,:], y[2,:])

Out[31]:
The theory behind Support Vector Machines says that if data are separable (in practice they are not, but for simplicity we will assume they are), than the optimal decision hyperplane is defined by \( d + 1 \) vectors, where \( d \) is the dimension of the data.

From there, we will take an inspiration and say that the decision hyperplane will be perpendicular to the vector defined by \( d \) vectors.

Therefore we randomly select \( d \) vectors.

```
In [32]: xx = X[:,rand(1:length(y), d)]
```

```
Out[32]: 2×2 Array{Float64,2}:
     7.71343  1.27999
    0.790019 -1.65198
```

Using analytic geometry, we find a normal vector of a hyperplane.

```
In [33]: w = xx' \ ones(d)
```

```
Out[33]: 2-element Array{Float64,1}:
     0.17755297325892452
    -0.4677639396875493
```

Assuming a square loss function (note that Support Vector Machines uses hinge loss), we use a “completion to square” to find an optimal threshold.

```
In [34]: b = mean(w' * X - Y')
```
Let's visualize the dependency of quadratic loss on the threshold.

```plaintext
In [35]: l(b) = mean((w' * X - b - Y').^2)
xr = -5:0.1:15
    plot(xr, l, dpi = 500)
```

Finally, we put everything together. The function `trial` below takes as an input training data \(X\), their corresponding labels \(Y\), and an array of indices defining the decision hyperplane. It calculates the normal vector of decision hyperplane, the threshold, and the error.

```plaintext
In [9]: function trial(X, Y, ind)
      xx = X[:,ind]
      w = xx' \ ones(size(X,1))
      b = mean(w' * X - Y')
      err = mean(((w' * X - b) .* Y') .< 0)
      (err, w, b)
end;
```

Finally, we create a simple coordinate ascent algorithm to find indexes of support vectors defining a normal vector of a decent hyperplane.
**In [13]:**

```
ind = [1,2]
best = trial(X,Y, ind)
@show best
for j in 1:3*size(X, 1)
    replaced_index = rand(1:size(X, 1))
    for i in setdiff(1:size(X, 2), ind)
        test_ind = deepcopy(ind)
        test_ind[replaced_index] = i
        try
            s = trial(X, Y, test_ind)
            if s[1] < best[1]
                best = s
                ind = test_ind
            end
        end catch
        end
    end
end
```

```
best = (0.77, [0.187039, 0.368005], 1.4638208442710092)
iteration: 4 error: 0.045 indices: [1, 4]
iteration: 15 error: 0.035 indices: [1, 15]
```

Visualize the normal vector of a decision hyperplane.

**In [30]:**

```
w, b = best[2], best[3]
scatter(x[1,:], x[2,:], dpi = 500);
scatter!(y[1,:], y[2,:]);
scatter!(X[1,ind], X[2,ind], color = :pink, marker = ([:circle :d], 6, 0.8))
xr = unique(sort(X[1,:]));
plot!(xr, x -> -x * w[1]/w[2] - b)
```

```
Out[30]:
```