Homework 09

Brown University

DATA 1010

Fall 2020

Problem 1

If we fit a kernelized SVM model using LIBSVM, then it produces a collection of values that can be used to compute the decision function (essential the prediction function, but just before we apply the signum function):

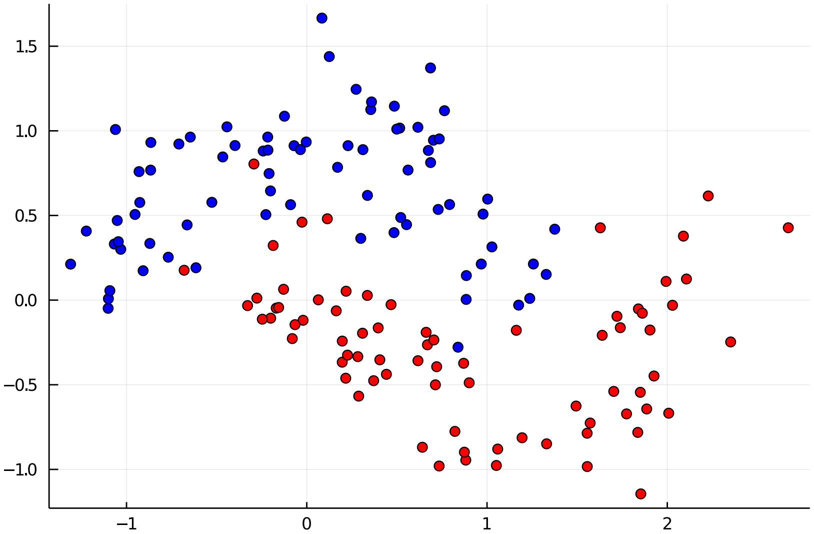
In [1]:
```
using Pkg; Pkg.activate("MLJ", shared=true)
Pkg.add("MLJ")
Pkg.add("MLJScikitLearnInterface")
```

```
 Activating environment at `~/.julia/environments/MLJ/Project.toml`
   Updating registry at `~/.julia/registries/General`
######################################################################## 100.0%
  Resolving package versions...
No Changes to `~/.julia/environments/MLJ/Project.toml`
No Changes to `~/.julia/environments/MLJ/Manifest.toml`
  Resolving package versions...
No Changes to `~/.julia/environments/MLJ/Project.toml`
No Changes to `~/.julia/environments/MLJ/Manifest.toml`
```

In [2]:
```
using MLJ, Plots
```

In [3]:
```
using Random; Random.seed!(123)
X, y = make_moons(noise = 1/4)
scatter(X.x1, X.x2, group = y, color = [:blue :red], label = "")
```

Out[3]:



In [4]:
```
model = @load SVMClassifier
model.kernel = "rbf"
svm = machine(model, X, y)
fit!(svm)
```

```
┌ Info: Precompiling MLJScikitLearnInterface [5ae90465-5518-4432-b9d2-8a1def2f0cab]
└ @ Base loading.jl:1278
┌ Info: Training Machine{SVMClassifier} @953.
└ @ MLJBase /home/enminz/.julia/packages/MLJBase/xg2Ti/src/machines.jl:319
```

Out[4]:
```
Machine{SVMClassifier} @953 trained 1 time.
  args:
    1:  Source @845 ⏎ `Table{AbstractArray{Continuous,1}}`
    2:  Source @574 ⏎ `AbstractArray{Multiclass{2},1}`
```

In [5]:
```
fitted_params(svm)
```

Out[5]:
```
(support = Int32[27, 33, 41, 43, 50, 52, 55, 58, 65, 79  …  108, 109, 110, 125, 132, 135, 137, 140, 142, 148],
 support_vectors = [0.4839195070201942 0.3987849338861288; -1.10041255494672 0.008576762121766862; … ; 0.06405257812100115 0.0017672298035346923; -
0.6801894138042015 0.17639266603252177],
 n_support = Int32[24, 25],
 dual_coef = [-1.0 -0.693755862643101 … 1.0 1.0],
 coef = nothing,
 intercept = [0.08985912448735472],
 fit_status = 0,
 classes = UInt32[0x00000001, 0x00000002],)
```

The `dual_coef` values correspond to what we're calling η in the math notation. However, note that only the nonzero η values are stored! To see which ones are nonzero, look at `support`. The `support_vectors` are also included, for your convenience. What we've been calling α corresponds to `intercept`.

Write some code which uses these values to compute the decision function (that is, from scratch, not using `_decision_function` as we did in class), and use it to plot a heatmap of the decision function. You might have to include the radial basis function manually, since that's actually hard-coded in LIBSVM.

Notes:
(1) the course cheatsheet describes the prediction function exactly.
(2) the knowledge you gain in this question is not Julia-specific; LIBSVM is the main library underlying SVM packages in various dynamic languages.

In [6]:
```
support, support_vectors, n_support, dual_coef, coef, intercept, fit_status, classes = fitted_params(svm)
```

Out[6]:
```
(support = Int32[27, 33, 41, 43, 50, 52, 55, 58, 65, 79  …  108, 109, 110, 125, 132, 135, 137, 140, 142, 148],
 support_vectors = [0.4839195070201942 0.3987849338861288; -1.10041255494672 0.008576762121766862; … ; 0.06405257812100115 0.0017672298035346923; -0.6801894138042015 0.17639266603252177],
 n_support = Int32[24, 25],
 dual_coef = [-1.0 -0.693755862643101 … 1.0 1.0],
 coef = nothing,
 intercept = [0.08985912448735472],
 fit_status = 0,
 classes = UInt32[0x00000001, 0x00000002],)
```

In [20]:
```
using LinearAlgebra
rbf_kernel(a,b;γ=1) = exp(-γ*sum((a.-b).^2));
y_nonzero = [y[i] for i in support]
# n = zeros(length(y))
# index = 1
# for i in support
#     n[i] = dual_coef[index]
#     index += 1
# end
int_y = [if i==1 1 else -1 end for i in y]
# new_X = [[X[1][i], X[2][i]] for i = 1:150]
# beta = new_X' * (n .* int_y)
function predict(x)
    k = sum([rbf_kernel(x, [support_vectors[i,1], support_vectors[i,2]])*dual_coef[i] for i in 1:length(y_nonzero)])
    res = k .+ intercept[1]
    return res[1]
end
```

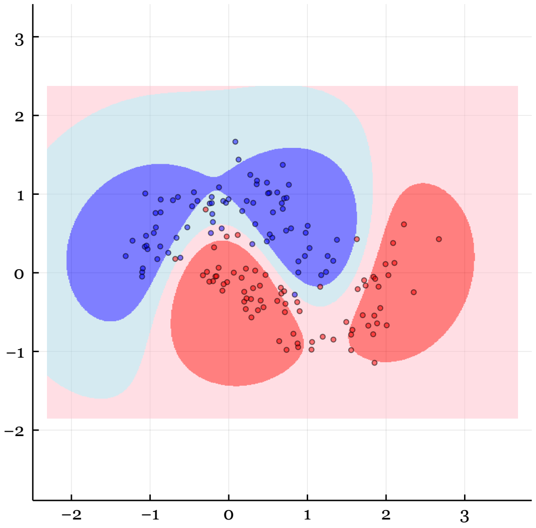
Out[20]:
```
predict (generic function with 2 methods)
```

In [21]:
```
using Statistics, LinearAlgebra, Distributions, Plots, JuMP, Ipopt, Contour
# include("contourplot.jl");
function visualize(signum = true)
    zone(x) = x < -1 ? 1 : (x < 0 ? 2 : (x < 1 ? 3 : 4))
    xmin, xmax = extrema([i[1] for i in new_X])
    ymin, ymax = extrema([i[2] for i in new_X])
    xrange = xmax - xmin
    xmax += 0.25*xrange
    xmin -= 0.25*xrange
    yrange = ymax - ymin
    ymax += 0.25*yrange
    ymin -= 0.25*yrange
    xs = range(xmin, stop = xmax, length=512)
    ys = range(ymin, stop = ymax, length=512)
    Plots.heatmap(xs, ys, (x,y) -> zone(predict([x,y])),
            fillopacity = 0.5, colorbar = false, fontfamily = "Palatino",
            fillcolor = cgrad([:blue, :lightblue, :pink, :red]), aspect_ratio = 1, size = (400, 400))
    reds = new_X[int_y .== 1]
    blues = new_X[int_y .== -1]
#     contourplot!(xs, ys, (x,y) -> predict([x,y]), [-1,0,1], color = :black, linewidth = 0.5)
    scatter!([i[1] for i in reds], [i[2] for i in reds], color = :red, label = "", ms = 2, opacity = 0.5)
    scatter!([i[1] for i in blues], [i[2] for i in blues], color = :blue, label = "", ms = 2, opacity = 0.5)
end
```

Out[21]: visualize (generic function with 2 methods)

In [22]:
```
visualize()
```

Out[22]:



Problem 2

Apply the Lagrange duality method we apply in Data Gymnasia to obtain the dual SVM to the problem of minimizing the objective function $f(x, y) = x^2 + y^2$ subject to the constraint $x + y \geq 5$. Show that strong duality holds (that is, we get the same value for the objective function if we swap min and max).

Note: this problem is asking you to walk through the same steps: introducing the function H, swapping min/max, etc. The only difference is that you're dealing with a much simpler optimization problem.

The constraint is $x + y \geq 5$, which is $5 - x - y \leq 0$. We define a function H(x) so that when $x \leq 0$, H(x) = 0 and when H(x) = $\inf$ otherwise.

So our optimization problem is equivalent to $x^2 + y^2 + H(5 - x - y)$, since when the constraint is not satified, H term will return infinity and the problem will not be minimized.

Let H(x) be $u * x$, the Lagrangian is L(x,u) = $x^2 + y^2 + u(5 - x - y)$, and the lagrangian dual function is $\theta(u)$ = min{$x^2 + y^2 + u(5 - x - y)$} = 5u + min{$x^2 + y^2 - ux - uy$} = 5u + min{$x^2 - ux$}+min{$y^2 - uy$}

For a fixed value $u \geq 0$, the minimum of L(x,u) is attained at x = y = $\frac{u}{2}$ and L(x,u) = $5u - \frac{u^2}{2}$.

The dual function is concave and differentiable, so the strong duality holds and the optimal objective function values are equal. We want to maximize the value of the dual function: $\frac{dL}{du} = 5 - u = 0$

This implies u = *5 and $\theta(u*) = 20 - 12.5 = 7.5$

Therefore, x(u*) = 2.5 and y(u*) = 2.5 and the minimized f(x,y) = 12.5

Problem 3

Let's take a look at the zeros and ones in the MNIST dataset:

In [221]:
```
Pkg.add("MLDatasets")
using MLDatasets
```

```
  Resolving package versions...
No Changes to `~/.julia/environments/MLJ/Project.toml`
No Changes to `~/.julia/environments/MLJ/Manifest.toml`
```

In [222]:
```
function load_MNIST_zeros_and_ones()
    features, labels = MNIST.traindata()
    features = reshape(features[:, :, labels .∈ Ref((0, 1))], (28^2, :))
    labels = labels[labels .∈ Ref((0,1))];
    float(features), labels
end
```

Out[222]:
```
load_MNIST_zeros_and_ones (generic function with 1 method)
```

In [223]:
```
features, labels = load_MNIST_zeros_and_ones()
```

```
This program has requested access to the data dependency MNIST.
which is not currently installed. It can be installed automatically, and you will not see this message again.

Dataset: THE MNIST DATABASE of handwritten digits
Authors: Yann LeCun, Corinna Cortes, Christopher J.C. Burges
Website: http://yann.lecun.com/exdb/mnist/

[LeCun et al., 1998a]
    Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner.
    "Gradient-based learning applied to document recognition."
    Proceedings of the IEEE, 86(11):2278-2324, November 1998

The files are available for download at the offical
website linked above. Note that using the data
responsibly and respecting copyright remains your
responsibility. The authors of MNIST aren't really
explicit about any terms of use, so please read the
website to make sure you want to download the
dataset.



Do you want to download the dataset from ["http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz", "http://yann.lecun.com/exdb/mnist/train-lab
els-idx1-ubyte.gz", "http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz", "http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz"] to
"/home/enminz/.julia/datadeps/MNIST"?
[y/n]
```

Out[223]:
```
(Float32[0.0 0.0 … 0.0 0.0; 0.0 0.0 … 0.0 0.0; … ; 0.0 0.0 … 0.0 0.0; 0.0 0.0 … 0.0 0.0], [0, 1, 1, 1, 1, 0, 1, 1, 0, 0  …  0, 0, 1, 1, 1, 0, 1, 1,
0, 1])
```

Each column of `features` stores the pixel intensities for a 28 by 28 image:

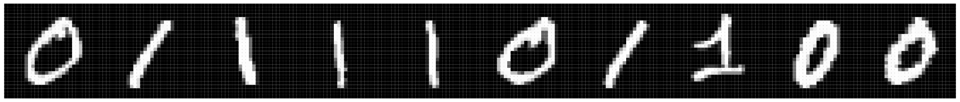
In [224]:
```
size(features)
```

Out[224]:
```
(784, 12665)
```

We can see these images by reshaping each column back into a 28 by 28 image and converting the intensity to an actual color value:

In [239]:
```
hcat([Gray.(reshape(features[:, k], 28, 28)') for k in 1:10]...)
```

Out[239]:



Would you guess that there exists a hyperplane in the 784-dimensional space in which the digits live which perfectly separates all the zeros from the ones?

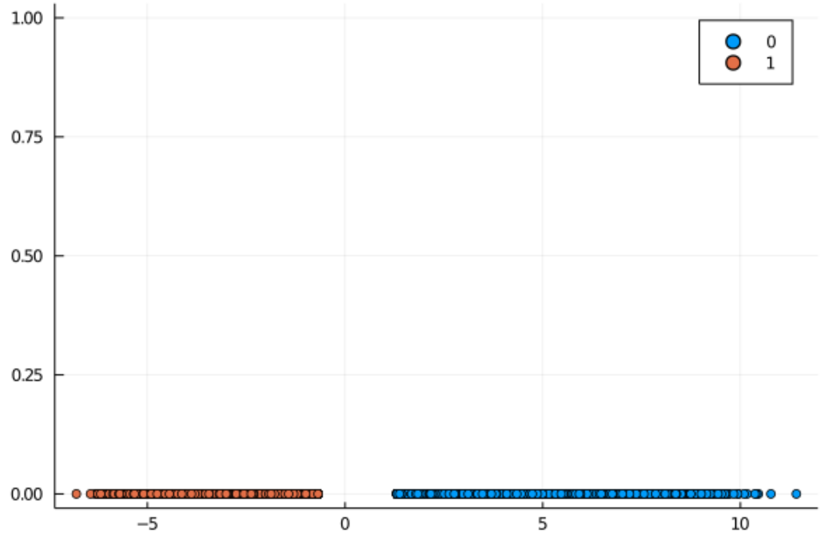
Make a conjecture first and then figure out the truth computationally.

(Sol) My conjecture is that it is separable because the dimensional space is high and there is a clear difference between the pattern of '1's and pattern of '0's in certain dimensions.

In [242]:
```
import LIBSVM
```

In [255]:
```
model = LIBSVM.svmtrain(float(features), float(labels), kernel = LIBSVM.Kernel.Linear, cost = 100.0);
β = model.SVs.X * model.coefs
gr(fmt=:png)
scatter(features' * β .- [1], zeros(size(features,2)), group = labels, legend = true)
```

Out[255]:



In [258]:
```
class = [if i < 0 1 else 0 end for i in sign.(features' * β .- [1])]
sum(class .== labels)/length(labels)
```

Out[258]: 1.0

The output shows that the 784-dimension vector is mapped to a certain value that are clearly separable by a hyperplane and we have a prediction accuracy of 1.0.

In [ ]: