Solution to Assignment 1

MATH7502 2019, Semester 2

Assignment 1 questions (https://courses.smp.uq.edu.au/MATH7502/2019/ass1.pdf)

Solution to Question 1

(a) $1'w$ is the number of words in the document

(b) $w_{282} = 0$ implies that the 282'nd word in the dictionary has 0 occurences in the document.

(c) $h = \frac{1}{1'w}w$.

(d)

In [1]:
```
using HTTP, JSON

data = HTTP.request("GET",
"https://ocw.mit.edu/ans7870/6/6.006/s08/lecturenotes/files/t8.shakespea
shakespeare = String(data.body)
shakespeareWords = split(shakespeare)

jsonWords = HTTP.request("GET",
"https://raw.githubusercontent.com/"*
"h-Klok/StatsWithJuliaBook/master/1_chapter/jsonCode.json")
parsedJsonDict = JSON.parse( String(jsonWords.body))

keywords = Array{String}(parsedJsonDict["words"])
numberToShow = parsedJsonDict["numToShow"]
wordCount = Dict([(x,count(w -> lowercase(w) == lowercase(x), shakespeai
                   for x in keywords])

sortedWordCount = sort(collect(wordCount),by=last,rev=true)
sortedWordCount[1:numberToShow]
```

Out[1]:
```
5-element Array{Pair{String,Int64},1}:
 "king" => 1698
 "love" => 1279
  "man" => 1033
  "sir" => 721
  "god" => 555
```

(e) Clearly there are multiple answers. One possible answer is to use the countmap() function from StatsBase

In [7]:
```
using HTTP, JSON, StatsBase

data = HTTP.request("GET", "https://ocw.mit.edu/ans7870/6/6.006/s08/lect
shakespeare = String(data.body)
shakespeareWords = lowercase.(split(shakespeare))

numberToShow = 20
wordCount = countmap(shakespeareWords)
sortedWordCount = sort(collect(wordCount),by=last,rev=true)
sortedWordCount[1:numberToShow]
```

Out[7]:
```
20-element Array{Pair{String,Int64},1}:
  "the" => 27549
  "and" => 26037
    "i" => 19540
   "to" => 18700
   "of" => 18010
    "a" => 14383
   "my" => 12455
   "in" => 10671
  "you" => 10630
 "that" => 10487
   "is" => 9145
  "for" => 7982
 "with" => 7931
  "not" => 7643
 "your" => 6871
  "his" => 6749
   "be" => 6700
  "but" => 5886
   "he" => 5884
   "as" => 5882
```

Solution to Question 2

Just write a as $a + b - b$. Then:

$$||a|| = ||a + b + (-b)|| \leq ||a + b|| + ||-b|| = ||a + b|| + ||b||$$

Hence,

$$||a|| - ||b|| \leq ||a + b||$$

Solution to Question 3

(a)

(i) $\beta = e_1$: Tommorow's sales prediction is having sales the same as today.

(ii) $\beta = 2e_1 - e_2$: Tommorow's sales predition is twice the sales of today take away the sales of yesterday.

(iii) $\beta = e_6$: Tommorow's sales prediction is having the same sales as 6 days ago.

(iv) $\beta = \frac{1}{2}e_1 + \frac{1}{2}e_2$: Tomorrow's sales prediction is the average of today and yesterday.

(b)

In [25]:
```
using Distributions, Random, LinearAlgebra, Statistics
Random.seed!(0)

T, M,  N = 100, 10, 1000

ee(i) = begin v = zeros(M); v[i] = 1; v end       #just a function for e
predict(beta,data,t) = data[t-1:-1:t-M]'*beta     #notice reverse order
makeData() = [cos(t*2pi/7) + rand(Normal(0,0.2)) for t in 1:T]
loss(data,pred) = norm(data[M+1:T] - pred)  #L_2 loss

betas = [ee(1),
         2ee(1)-ee(2),
         ee(6),
         (ee(1) + ee(2))/2,
         ee(7)]                  #notice this additional beta not specif

for beta in betas
    losses = []
    for _ in 1:N
        data = makeData()
        pred = [predict(beta,data,t) for t in M+1:T]
        push!(losses,loss(data,pred))
    end
    println("Loss is: ", mean(losses))
end
```

```
Loss is: 6.397265063244324
Loss is: 6.8827023607315905
Loss is: 6.4291797868524965
Loss is: 8.27291621054637
Loss is: 2.673542339803326
```

It can thus be seen that from the four specified estimators the first one does the best (slightly better than the third). However if we specify another estimator with $\beta = e_7$ we do much better.

Solution to Question 4

$||x||_1 = |x_1| + \ldots + |x_n|$

(i) $||x||_1 = |x_1| + \ldots + |x_n| \geq 0$ trivially.

(ii) $||x + y||_1 \leq ||x||_1 + ||y||_1$:

$$||x + y||_1 = \sum_{i=1}^{n} |x_i + y_i| \leq \sum_{i=1}^{n} |x_i| + |y_i| = ||x||_1 + ||y||_1$$

This follows from the triangle inequality for real numbers $|a + b| \leq |a| + |b|$

(iii) $||x||_1 = 0$ if and only if $x = 0$. Follows trivially.

(iv) $||\beta x||_1 = |\beta| ||x||_1$:

$$||\beta x||_1 = \sum_{i=1}^{n} |\beta x_i| = |\beta| \sum_{i=1}^{n} |x_i| = |\beta| ||x||_1 .$$

$||x||_\infty = \max\{|x_1|, \dots, |x_n|\}$

(i) $||x||_\infty \geq |x_i| \geq 0$ for all i. Hence $||x||_\infty \geq 0$.

(ii) $||x + y||_\infty \leq ||x||_\infty + ||y||_\infty$:

$||x + y||_\infty = \max\{|x_1 + y_1|, \dots, |x_n + y_n|\} = |x_i + y_i| \leq |x_i| + |y_i|$ for some i.

And this is $\leq |x_j| + |y_k|$ for the j and k that determine $||x||_\infty$ and $||y||_\infty$ respectivly. Hence,

$$||x + y||_\infty \leq ||x||_\infty + ||y||_\infty .$$

(iii) $||x||_\infty = 0$ if and only if $x = 0$. Follows trivially.

(iv) $||\beta x||_\infty = |\beta| ||x||_\infty$.

This is by pulling out the constant $|\beta|$ out of the maximum: $\max\{|a|z, |a|u, |a|v\} = |a| \max\{z, u, v\}$.

solution

Solution to Question 5

(a)

$$\int_\alpha^\beta p(x)\,dx = \left[c_1 x + c_2 \frac{1}{2}x^2 + c_3 \frac{1}{3}x^3 + \dots + c_n \frac{1}{n}x^n\right|_\alpha^\beta$$

$$= c_1\beta + c_2 \frac{1}{2}\beta^2 + c_3 \frac{1}{3}\beta^3 + \dots + c_n \frac{1}{n}\beta^n - (c_1\alpha + c_2 \frac{1}{2}\alpha^2 + c_3 \frac{1}{3}x^3 + \dots + c_n \frac{1}{n}\alpha^n)$$

$$= c_1(\beta - \alpha) + c_2 \frac{1}{2}(\beta^2 - \alpha^2) + \dots + c_n \frac{1}{n}(\beta^n - \alpha^n) = c^T a$$

where $c = (c_1, \dots, c^n)^T$ and

$$a = \left(\beta - \alpha, \frac{\beta^2 - \alpha^2}{2}, \dots, \frac{\beta^n - \alpha^n}{n}\right)^T .$$

(b) Since $p(\alpha) = c_1 + c_2\alpha + c_3\alpha^2 + \dots + c_n x^{n-1}$ we have:

$$p'(\alpha) = c_2 + 2c_3\alpha + 3c_4\alpha^2 + \dots + (n-1)c_n\alpha^{n-2} = b^T(c_1, \dots, c_n)^T .$$

Hence,

$$b = \left(0, 1, 2\alpha, 3\alpha^2, \dots, (n-1)\alpha^{n-2}\right)^T .$$

Solution to Question 6

(a) Testing for homogeniety: $f(\alpha x) = \alpha f(x)$:

$\phi(-1(-1,1,1)) = (-1)\phi(-1,1,1)$ implies $\phi(1,-1,-1) = -1 \times 1$ but $\phi(1,-1,1) = 1$. So the test fails. Hence (ii) is the correct answer. ϕ cannot be linear.

(b) One modifcation can be $\phi(1,1,1) = 5$, $\phi(2,2,2) = 10$ and $\phi(3,3,3) = 15$. In this case denote $a = (1,1,1)$. We now have $\phi(a + 2a) = \phi(3a)$ and $\phi(2a) = 2\phi(a)$. This function can be linear however it also can be modified as to be non-linear based on other points. So the answer is (ii).

Solution to Question 7

(a)

$$L(a) = \|x - a\mathbf{1}\| = \sqrt{\sum_{i=1}^{n}(x_i - a)^2}$$

The minimizer of $L(a)$ is also the minimizer of $\tilde{L}(a) = L(a)^2$.

$$\tilde{L}(a) = \sum_{i=1}^{n}(x_i - a)^2 = \sum_{i=1}^{n}(x_i^2 - 2x_i a + a^2) = na^2 - 2(\sum x_i)a + \sum x_i^2.$$

This can be done via calculus however also observed that $\tilde{L}(a)$ is an upward facing parabula and is hence minimized at,

$$a^* = \frac{-(-2\sum x_i)}{2(n)} = \bar{x}$$

where $\bar{x}$ is the mean of $x_1, \ldots, x_n$.

(b) For gradient descent observe that $\frac{d}{a}\tilde{L}(a) = 2(na - \sum x_i)$. Denote $S = \sum x_i$ and hence the gradient is $2(na - S)$.

Gradient descent then starts at some a_0 and implements,

$$a_{k+1} = a_k - 2\eta(na_k - S) = (1 - 2\eta n)a_k + 2\eta S$$

In [67]:

```
using Statistics, Random
Random.seed!(1)

n = 100;
data = 9 .+ 2*rand(n);
xbar = mean(data) #sample mean
println("The sample mean is: ",xbar)
S = sum(data);

function gd(a0,eta)
    a = a0
    aPrev = -a
    numSteps = 0
    while abs(a-aPrev) > 10^-3 && numSteps < 1000
        aPrev = a
        a = (1-2*eta*n)*a + 2*eta*S
        numSteps += 1
    end
    println("Executed ",numSteps, " steps")
    a
end
aStar = gd(50,0.001)
println("The result of gradient descent is ", aStar)
```

```
The sample mean is: 9.973653925527081
Executed 42 steps
The result of gradient descent is 9.977058990472438
```

(c) We have a recursion of the form $a_{k+1} = \kappa a_k + \gamma$ where $\kappa = 1 - 2\eta n$ and $\gamma = 2\eta S$

Notice:

$$a_1 = \kappa a_0 + \gamma$$

$$a_2 = \kappa^2 a_0 + \kappa\gamma + \gamma$$

$$a_3 = \kappa^3 a_0 + \kappa^2\gamma + \kappa\gamma + \gamma$$

and in general,

$$a_k = \kappa^k + \gamma \sum_{i=0}^{k-1} \kappa^i = \kappa^k + \gamma\frac{\kappa^k - 1}{\kappa - 1} = \kappa^k(1 + \frac{\gamma}{\kappa - 1}) + \frac{\gamma}{1-\kappa} = \kappa^k(1 + \frac{\gamma}{\kappa - 1}) + \bar{x}.$$

Hence we have that $\lim_{k\to\infty} a_k$ diverges if $|\kappa| \geq 1$ and otherwise converges to $\bar{x}$. So we need $|1 - 2\eta n| < 1$. With $\eta > 0$ we always have $1 - 2\eta n < 1$ so the requirment is $-1 < 1 - 2\eta n$. Hence we need $\eta < \frac{1}{n}$

In [70]:

```
badEta = 1/n + 0.001
gd(50,badEta)
```

```
Executed 1000 steps
```

Out[70]:

```
6.075639453913123e80
```

In [71]:
```
goodEta = 1/n - 0.001
gd(50,goodEta)
```

```
Executed 52 steps
```

Out[71]:
```
9.974019541591606
```

Solution to Question 8

(assume for simplicity that n is even)

(a) Define the $n \times n$ matrices A_1, A_2 (these are composed of collumns of either e_i or 0):

$$A_1 = [e_1\ 0\ e_3\ 0\ \dots\ e_{n-1}\ 0], \qquad A_2 = [0\ e_2\ 0\ e_4\ \dots\ 0\ e_n],$$

Then $f(x) = \Big(f_1(x),\ f_2(x)\Big)$ has components,

$$f_1(x) = \|A_1 x\|^2 + \mathbf{1}^T(A_2 x), \qquad f_2(x) = \|A_2 x\|^2 + \mathbf{1}^T(A_1 x).$$

(b) The $2 \times n$ Jacobian matrix $D(x)$ can be worked out directly (e.g. assume n is even):

$$D(x) = \begin{bmatrix} 2x_1 & 1 & 2x_3 & 1 & \dots & 2x_{n-1} & 1 \\ 1 & 2x_2 & 1 & 2x_3 & \dots & 1 & 2x_n \end{bmatrix}.$$

It can also be worked out via vectors. The first row as $\nabla f_1(x)^T$ and second row as $\nabla f_2(x)^T$. Observe that $\|Bx\|^2 = x^T B^T B x$ and that $\nabla \|Bx\|^2 = 2B^T B x$. Further observe that $A_1^T A_1 = A_1$ and $A_2^T A_2 = A_2$. Also A_1 and A_2 are symmetric. Hence,

$$\nabla f_1(x) = 2A_1 x + \mathbf{1}^T A_2, \qquad \nabla f_2(x) = 2A_2 x + \mathbf{1}^T A_1.$$

Or

$$D(x) = \begin{bmatrix} 2x^T A_1 + \mathbf{1}^T A_2 \\ 2x^T A_2 + \mathbf{1}^T A_1 \end{bmatrix}.$$

(c) and (d) Now take $z = \mathbf{1}$ hence

$$D(z) = \begin{bmatrix} 2\mathbf{1}^T A_1 + \mathbf{1}^T A_2 \\ 2\mathbf{1}^T A_2 + \mathbf{1}^T A_1 \end{bmatrix} = \begin{bmatrix} \mathbf{1}^T(2A_1 + A_2) \\ \mathbf{1}^T(2A_2 + A_1) \end{bmatrix} = \begin{bmatrix} 2 & 1 & 2 & 1 & \dots & 2 & 1 \\ 1 & 2 & 1 & 2 & \dots & 1 & 2 \end{bmatrix}$$

In [129]:
```
n = 6;
A1 = [isodd(i) && i == j ? 1 : 0 for i in 1:n, j in 1:n ];
A2 = [iseven(i) && i == j ? 1 : 0 for i in 1:n, j in 1:n ];
D = [ones(n)'*(2A1+A2)
     ones(n)'*(2A2+A1)]
```

Out[129]:
```
2×6 Array{Float64,2}:
 2.0  1.0  2.0  1.0  2.0  1.0
 1.0  2.0  1.0  2.0  1.0  2.0
```

In [130]:
```
using LinearAlgebra
```

In [131]:
```
f(x) = [norm(A1*x)^2 + ones(n)'*A2*x, norm(A2*x)^2 + ones(n)'*A1*x]
f1 = f(ones(n))
```

Out[131]:
```
2-element Array{Float64,1}:
 6.0
 6.0
```

In [132]:
```
fApprox(x) = f1 + D*(x-ones(n))
```

Out[132]:
```
fApprox (generic function with 1 method)
```

In [145]:
```
#This function takes random vectors of radius r around z=ones(n) and lo
function randomWorst(r)
    N = 10^5
    worst = -Inf
    for _ in 1:N
        v = rand(n)
        v = r*v/norm(v)
        x = ones(n) + v
        err = norm(f(x) - fApprox(x))
        if err > worst
            worst = err
        end
    end
    println("After trying $(N) random vectors within raidus $(r), the wo
end
randomWorst(0.5)
randomWorst(0.25)
randomWorst(1.0)
```

```
After trying 100000 random vectors within raidus 0.5, the worst has is 0.
2499577140111187
After trying 100000 random vectors within raidus 0.25, the worst has is
0.0624535565900629
After trying 100000 random vectors within raidus 1.0, the worst has is 0.
99943513971916
```

Note: There is also an analytical solution

Solution to Question 9

Listing 9.9 from SWJ (https://github.com/h-Klok/StatsWithJuliaBook/blob/master/9_chapter/kMeansManual.jl)

In [38]:

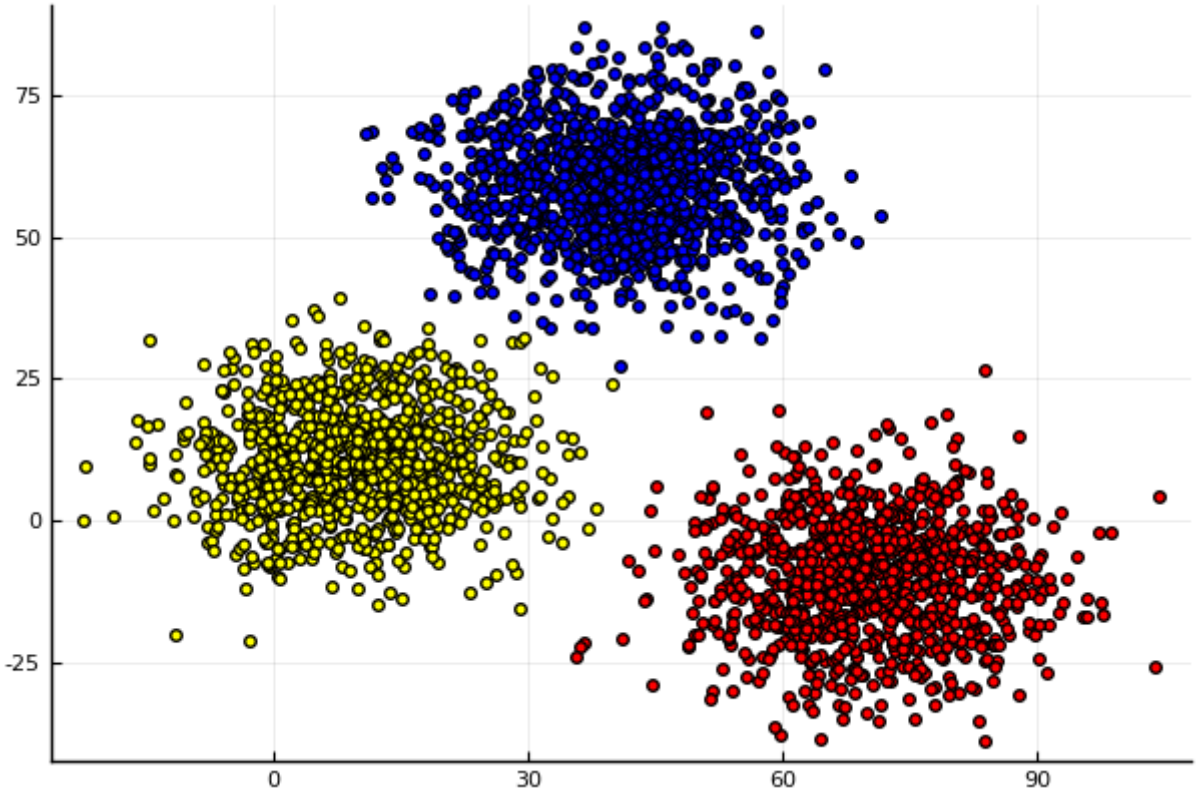
```
using RDatasets, Distributions, Random
Random.seed!(1)

k = 3

xclara = dataset("cluster", "xclara")
n,_ = size(xclara)
dataPoints = [convert(Array{Float64,1},xclara[i,:]) for i in 1:n]
shuffle!(dataPoints)

xMin,xMax = minimum(first.(dataPoints)),maximum(first.(dataPoints))
yMin,yMax = minimum(last.(dataPoints)),maximum(last.(dataPoints))

means = [[rand(Uniform(xMin,xMax)),rand(Uniform(yMin,yMax))]  for _ in 1
labels = rand([1,k],n)
prevMeans = -means

while norm(prevMeans - means) > 0.001
    prevMeans = means
    labels = [findmin([norm(means[i]-x) for i in 1:k])[2]  for x in data
    means = [sum(dataPoints[labels .== i])/sum(labels .==i) for i in 1:k
end

cnts = [sum(labels .== i) for i in 1:k]

println("Counts of clusters (manual implementation): ", cnts)
```

```
Counts of clusters (manual implementation): [899, 952, 1149]
```

In [39]:

```
1  using Plots; pyplot()
2  scatter(first.(dataPoints[labels .== 1]),last.(dataPoints[labels .== 1])
3  scatter!(first.(dataPoints[labels .== 2]),last.(dataPoints[labels .== 2]
4  scatter!(first.(dataPoints[labels .== 3]),last.(dataPoints[labels .== 3]
```

Out[39]:



Now fix some labels... See lines 12-14 for fixing lables and lines 26-28 for enforcing it in the algorithm

In [40]:

```
using RDatasets, Plots, Distributions, Random, LinearAlgebra
Random.seed!(1)

k = 3

xclara = dataset("cluster", "xclara")
n,_ = size(xclara)
dataPoints = [convert(Array{Float64,1},xclara[i,:]) for i in 1:n]
shuffle!(dataPoints)

#Say these are the fixed labels - they are based on the location
fixedLabels = [ filter((k)->norm(dataPoints[k] - [70,70]) < 20,1:n),
                filter((k)->norm(dataPoints[k] - [50,50]) < 5,1:n),
                filter((k)->norm(dataPoints[k] - [90,-5]) < 5,1:n)]

xMin,xMax = minimum(first.(dataPoints)),maximum(first.(dataPoints))
yMin,yMax = minimum(last.(dataPoints)),maximum(last.(dataPoints))

means = [[rand(Uniform(xMin,xMax)),rand(Uniform(yMin,yMax))]  for _ in 1
labels = rand([1,k],n)
prevMeans = -means

while norm(prevMeans - means) > 0.001
    prevMeans = means
    labels = [findmin([norm(means[i]-x) for i in 1:k])[2]  for x in data
    for k in 1:3
        labels[fixedLabels[k]] .= k
    end
    means = [sum(dataPoints[labels .== i])/sum(labels .==i) for i in 1:k
end

cnts = [sum(labels .== i) for i in 1:k]

println("Counts of clusters (manual implementation): ", cnts)
fixedLabels
```

```
Counts of clusters (manual implementation): [1024, 1004, 972]
```

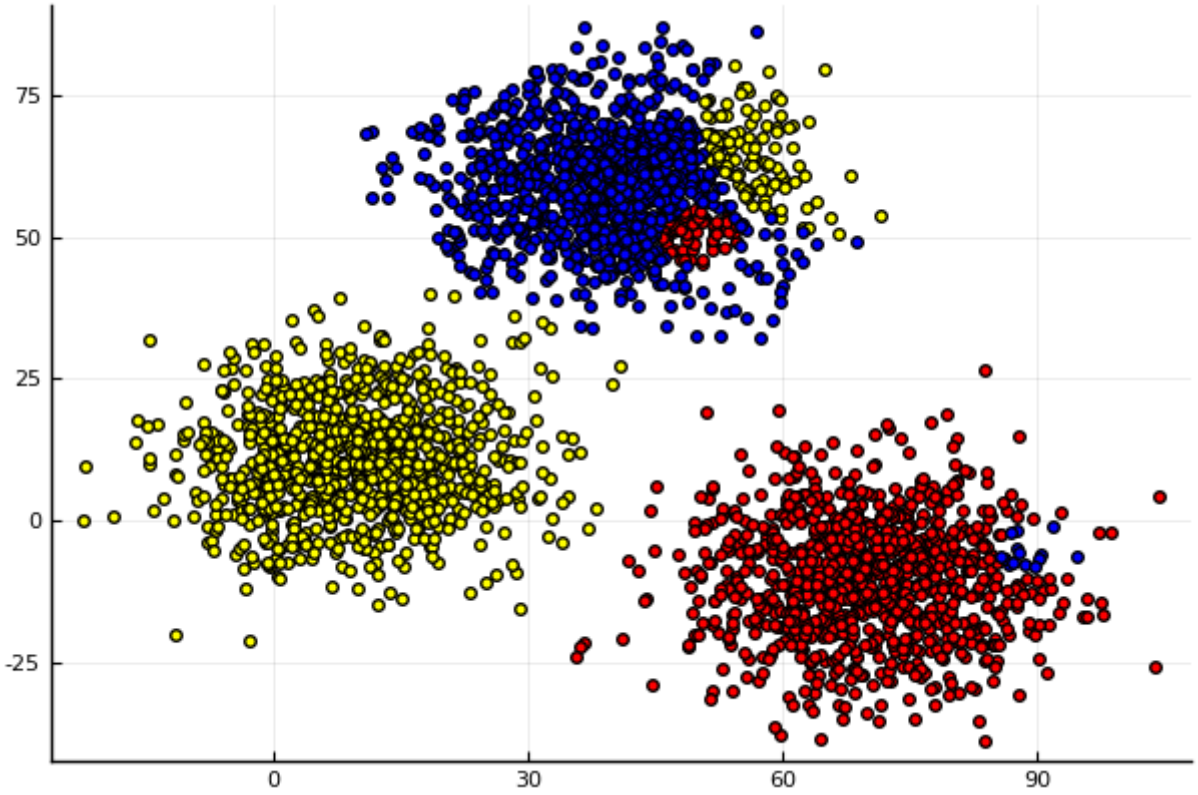
Out[40]:

```
3-element Array{Array{Int64,1},1}:
 [15, 21, 65, 73, 87, 110, 116, 119, 136, 144  …  2678, 2733, 2814, 2839,
2883, 2900, 2915, 2955, 2995, 3000]
 [29, 57, 95, 160, 198, 228, 237, 298, 330, 338  …  2863, 2892, 2896, 292
5, 2929, 2930, 2945, 2948, 2958, 2990]
 [279, 890, 991, 1044, 1105, 1121, 1272, 1377, 1945, 1961, 2170, 2394, 25
25, 2855]
```

In [41]:

```
using Plots; pyplot()
scatter(first.(dataPoints[labels .== 1]),last.(dataPoints[labels .== 1])
scatter!(first.(dataPoints[labels .== 2]),last.(dataPoints[labels .== 2]
scatter!(first.(dataPoints[labels .== 3]),last.(dataPoints[labels .== 3]
```

Out[41]:



Solution to Question 10

We have that for $i = 1, \dots, k$: $b_i = \sum_{j=1}^{m} \alpha_j a_j$. Further, $c = \sum_{i=1}^{k} \beta_i b_i$.

Hence

$$c = \sum_{i=1}^{k} \beta_i \sum_{j=1}^{m} \alpha_j a_j = \sum_{i=1}^{k} \sum_{j=1}^{m} \beta_i a_j = \sum_{j=1}^{m} \sum_{i=1}^{k} \beta_i \alpha_j a_j = \sum_{j=1}^{m} a_j \sum_{i=1}^{k} \beta_i \alpha_j = \sum_{j=1}^{m} a_j \gamma_j$$

where $\gamma_j = \sum_{i=1}^{k} \beta_i \alpha_j$. Then c is a linear combination of $a_1, \dots, a_m$.