Objectives

- Continuation of Prior Lecture
- More about kNN
- Code in Julia
- XPL's

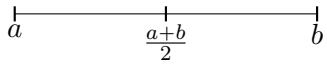
---

1 Continuation of Prior Lecture

Consider the following.

1. Given $[a, b] \to [0, 1]$ our objective is to map $a \to 0$ and $b \to 1$

$$\begin{aligned} x &= \frac{1}{b-a}(x-a) \\ &= \frac{1}{b-a}x - \frac{a}{b-a} \\ &= \frac{1}{b-a}\left(\frac{a+b}{2}\right) - \frac{2a}{2(b-a)} \\ &= \frac{a+b-2a}{2(b-a)} \\ &= \frac{b-a}{2(b-a)} \\ &= \frac{1}{2} \end{aligned}$$



Check the formula by testing with $0, \frac{1}{2}, 1$, which make up the bounds and the midpoint.
It is given that $(a, 0) \wedge (b, 1)$ where $m = \frac{1-0}{b-a} = \frac{1}{b-a}$.
We can then derive $y - 0 = \frac{1}{b-a}(x-a) = \frac{1}{b-a}x - \frac{a}{b-a}$

2. $[a,b] \to [c,d]$

$$\frac{x-a}{b-a}d + \frac{b-x}{b-a}c$$
$$(a,c) \wedge (b,d) \to m = \frac{d-c}{b-a}$$

2 More about kNN

argmax$\sum_{i \in n_k} \text{I}c_j(y_i)$ $j = 1, 2, ...m$
suppose $j = 2$ and $k = 8$, it is binary classification, N_k = set of indicies of the KNN

Example problem:
$y = [1, -1, -1, 1, 1, 1, -1, 1]$ $c_1 = 1, c_2 = -1$ $x_t \to 1$
$argmax_{j=[1,2]}[I(1 = 1) + I(1 = -1) + I(1 = -1) + I(1 = 1) + I(1 = 1) + I(1 = 1) + I(1 = -1))$
+ I(1=1),
$I(-1 = 1)+I(-1 = -1)+I(-1 = -1)+I(-1 = 1)+I(-1 = 1)+I(-1 = 1)+I(-1 = -1)+I(-1 = 1)$
$argmax_{j=[1,2]}(1 + 0 + 0 + 1 + 1 + 1 + 0 + 1, 0 + 1 + 1 + 0 + 0 + 0 + 1 + 0)$
$argmax_{j=[1,2]}\ [5, 3] = 1$

3 Code in Julia

The following is code written in Julia 1.10.4. It is represeting a kNN algorithm on the Iris dataset.

```
using Pkg
using Distances
using RDataSets
using MLJBase
Pkg.add("StatsBase")
using StatsBase

iris=dataset("datasets","Iris")
x = Matrix(iris[:,1:4])
y=@.ifelse(iris.species=="setosa",1,-1)
c = unique(y)
c[argmax(map(i->sum(y.==c[i]),1:lastindex(c)))]
mode(y)

#This can also be done using the NearestNeighbors package
function kNN(X,x,y,k,d = Euclidean())
    n-size(X,2)
    distances = map(i-> d(x,X[:,i]), 1:n)
    indices = partialsortperm(distances,1:k)
    if typeof(y) == Vector{union{Float32,Float64}}
        yhat = mean(y[indices])
    else
```

```
        yhat = mode
    end
    return yhat
end

#Metrics
accuracy = sum(yhat.==ytest/length(ytest))
precsion = sum ((yhat.==1.&(ytest.==1)))/sum(yhat.==1)
recall = sum((yhat.==1.&(ytest.==1)))/sum(ytest.==1)
specircity = sum((yhat.==-1).&(ytest.==-1))/sum(ytest.==-1)
f1 = 2*[precsion * recall / (precision + recall)
Distances.Hamming()(ytest,yhat)
train, test = partition(1:size(X,2),0.7,shuffle=true)

    Xtrain = X[:,train]
    Xtest = X[:test]
    ytrain = y[train]
    ytest = y[test]

yhat = map(i->kNN(Xtrain,Xtest[:,i],ytrain,1)1:size(Xtest,2))
```

The following is some code about setting a random seed

```
Pkg.add("random")
using Random
Random.seed!(123) #Where 123 is the seed
```

4 XPL's

1.) Code up a partition function