DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 1

Paradigma Funcional.

Paradigmas de Lenguajes

Grupo Two and a Half Blondes

Integrante	LU	Correo electrónico
De Sousa Bispo, Germán	359/12	germandesousa@gmail.com
Fernandez, Esteban	691/12	esteban.pmf@gmail.com
Wright, Carolina	876/12	wright.carolina@gmail.com

Reservado para la cátedra

Instancia	Fecha	Docente	Nota
Primera entrega			
Segunda entrega			

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Código

1.1. Ejercicio 1

```
split :: Eq a => a -> [a] -> [[a]]
split elementoSeparador xs = filter (\word -> length word > 0) (foldr f [[]] xs)
                             where f = (\x xss -> if x == elementoSeparador
                                                  then [] : xss
                                                  else (x : head(xss)) : tail(xss))
```

1.2. Ejercicio 2

```
longitudPromedioPalabras :: Extractor
longitudPromedioPalabras xs =  mean (map genericLength (listaDePalabras xs))
                               where listaDePalabras xs = split ' ' xs
```

1.3. Ejercicio 3

```
cuentas :: Eq a => [a] -> [(Int, a)]
cuentas xs = zip cantidadDeRepeticiones (nub xs)
        where cantidadDeRepeticiones = [length (filter (== y) xs)| y <- nub xs]
```

1.4. Ejercicio 4

```
repeticionesPromedio :: Extractor
repeticionesPromedio xs = mean ( map (\tupla -> fromIntegral (fst tupla))
                                     (cuentas (listaDePalabras xs)))
                          where listaDePalabras xs = split ' ' xs

tokens :: [Char]
tokens = "\_,)(*;-=>/.{}\"&:+#[]<|\%!\'@?^`$` abcdefghijklmnopqrstuvwxyz
         0123456789"
```

1.5. Ejercicio 5

```
fstDeLaUnicaTuplaEnLista :: [(Int, a)] -> Int
fstDeLaUnicaTuplaEnLista [] = 0
fstDeLaUnicaTuplaEnLista [x] = fst x

frecuenciaTokens :: [Extractor]
frecuenciaTokens =  map (\token -> (\texto -> let elemNQuantities =
                        cuentas texto in
                        (getTokenQuantityIn elemNQuantities token) /
                        (sumAllQuantitiesIn elemNQuantities))) tokens

sumAllQuantitiesIn :: [(Int, a)] -> Float
sumAllQuantitiesIn = (\elemNQuantities -> fromIntegral
                                $ sum
                                $ map (\elemAndQuantity -> fst elemAndQuantity)
                                $ elemNQuantities)

getTokenQuantityIn :: Eq a => [(Int, a)] -> a -> Float
getTokenQuantityIn = (\elemNQuantities token -> fromIntegral
                    $ fstDeLaUnicaTuplaEnLista
                    $ filter (\elemAndQuantity -> (snd elemAndQuantity)==token)
                    $ elemNQuantities)
```

1.6. Ejercicio 6

```
normalizarExtractor :: [Texto] -> Extractor -> Extractor
normalizarExtractor [] extractor = const 0
normalizarExtractor textos extractor =  let maximoFeature =
                        maximum (map abs [(extractor texto) | texto <- textos])
                        in (\text -> (extractor text) / maximoFeature)
```

1.7. Ejercicio 7

```
extraerFeatures :: [Extractor] -> [Texto] -> Datos
extraerFeatures extractores textos = let extractoresNorm =
                map (\extr -> normalizarExtractor textos extr) extractores
                in map (\text->
                    (map (\normExtr -> normExtr text) extractoresNorm)) textos
```

1.8. Ejercicio 8

```
distEuclideana :: Medida
distEuclideana p q = sqrt (sum (binomiosCuadrado p q) )
                 where binomiosCuadrado p q = map (\x -> x*x) (zipWith (-) p q)

distCoseno :: Medida
distCoseno p q = (sumatoriaProductos p q) / (productoVectorial p q)

sumatoriaProductos :: Medida
sumatoriaProductos p q = sum (productos p q)
                       where productos = zipWith (*)

productoVectorial :: Medida
productoVectorial p q = sqrt((sumatoriaProductos p p)*(sumatoriaProductos q q))
```

1.9. Ejercicio 9

```
knn :: Int -> Datos -> [Etiqueta] -> Medida -> Modelo
knn n matrizDatos etiquetas fDistancia = (\instancia -> moda n etiquetas
    $ zip (getDistanciasAInstancia matrizDatos instancia fDistancia) etiquetas)

getDistanciasAInstancia :: Datos -> Instancia -> Medida -> [Float]
getDistanciasAInstancia = (\matrizDatos instancia fDistancia ->
                                map (\dato ->
                                        fDistancia dato instancia) matrizDatos)

moda :: Int  -> [Etiqueta] -> [(Float, Etiqueta)] -> Etiqueta
moda = (\n etiquetas distsConEtiqs  -> snd $ maximumBy compare
                                    $ getNMejoresEtiqs n distsConEtiqs etiquetas)

getNMejoresEtiqs :: Int->[(Float, Etiqueta)]->[Etiqueta] ->[(Int, Etiqueta)]
getNMejoresEtiqs = (\n distanciasConEtiqs etiquetas ->
                        contarAparicionesEtiq (nub etiquetas)
                        $ take n
                        $ sortBy compare distanciasConEtiqs)

contarAparicionesEtiq :: [Etiqueta] -> [(Float, Etiqueta)] -> [(Int, Etiqueta)]
contarAparicionesEtiq = (\etiquetasSinRepe nMasCercanos ->
                [(aparicionesEtiqueta, etiqueta) | etiqueta<-etiquetasSinRepe,
                        let aparicionesEtiqueta =
                         length(filter(matcheaEtiqueta etiqueta) nMasCercanos)])

matcheaEtiqueta:: Etiqueta -> (Float, Etiqueta) -> Bool
matcheaEtiqueta etiqueta = (\label tupla -> label==(snd tupla)) etiqueta
```

1.10. Ejercicio 10

```
separarDatos :: Datos -> [Etiqueta] -> Int -> Int
                -> (Datos, Datos, [Etiqueta], [Etiqueta])
separarDatos datos etiquetas n p =
   let datosParticionado = (sacarInvalidos (foldl (\z elem ->
          if (length (last z)) < (div (length datos) n)
          then (init z) ++ [(last z) ++ [elem]]
          else (z ++ [[elem]])) [[]] datos) n)
   in let etiquetasParticionado = (sacarInvalidos (foldl (\z elem ->
          if (length (last z)) < (div (length etiquetas) n)
          then (init z) ++ [(last z) ++ [elem]]
          else (z ++ [[elem]])) [[]] etiquetas) n)
   in (getTrain datosParticionado p, getVal datosParticionado p,
       getTrain etiquetasParticionado p, getVal etiquetasParticionado p)

sacarInvalidos :: [[a]] -> Int -> [[a]]
sacarInvalidos datos n = if (length (last datos)) < n
                         then init datos else datos

getTrain:: [[a]] -> Int -> [a]
getTrain datos p = concat ((take (p-1) datos) ++ (drop p datos))

getVal::[a] -> Int -> a
getVal datos n = last (take n datos)
```

1.11. Ejercicio 11

```
accuracy :: [Etiqueta] -> [Etiqueta] -> Float
accuracy e1 e2 = sumaIguales (zip e1 e2) / fromIntegral (length (zip e1 e2))
   where sumaIguales = foldr (\t rec -> if fst t == snd t
                                        then 1+rec
                                        else rec) 0
```

1.12. Ejercicio 12

```
nFoldCrossValidation :: Int -> Datos -> [Etiqueta] -> Float
nFoldCrossValidation n datos etiquetas = mean $ accuracyN
        $ applyKnnToPartitions [separarDatos datos etiquetas n p | p <- [1..n]]

applyKnnToPartitions :: [(Datos, Datos, [Etiqueta], [Etiqueta])]
                        -> [([Etiqueta], [Etiqueta])]
applyKnnToPartitions = map (\(xTrain, xValid, yTrain, yValid) ->
                              (applyKnnToAllValid xTrain yTrain xValid, yValid))

applyKnnToAllValid :: Datos -> [Etiqueta] -> Datos -> [Etiqueta]
applyKnnToAllValid = (\xTrain yTrain xValid ->
  let trainedKnn = knn 15 xTrain yTrain distEuclideana
                   in map (\validInstancia -> trainedKnn validInstancia)xValid)

accuracyN :: [([Etiqueta], [Etiqueta])] -> [Float]
accuracyN = map (\(etiquetasSupuestas, etiquetasReales) ->
                      accuracy etiquetasSupuestas etiquetasReales)
```

2. Tests

2.1. Ejercicio 1

```
splitPorEspacioPresente = split ' ' "Habia una vez."
splitPorComaPresente = split ',' "Habia, una, vez."
splitPorEspacioNoPresente = split ' ' "Habiaunavez."
splitPorComaNoPresente = split ',' "Habia una vez."


splitTest1 = TestCase (assertEqual "Por espacio, presente"
                           ["Habia", "una", "vez."] splitPorEspacioPresente)
splitTest2 = TestCase (assertEqual "Por coma, presente"
                           ["Habia", " una", " vez."] splitPorComaPresente)
splitTest3 = TestCase (assertEqual "Por espacio, no presente "
                           ["Habiaunavez."] splitPorEspacioNoPresente)
splitTest4 = TestCase (assertEqual "Por coma, no presente"
                           ["Habia una vez."] splitPorComaNoPresente)
```

2.2. Ejercicio 2

```
longitudPromedioLetrasSueltas = longitudPromedioPalabras "a b c d e f g h i"
longitudPromedioDosLetrasXPalabra = longitudPromedioPalabras
                                    "aa bb cc dd ee ff gg hh ii"
longitudPromedioUnaPalabraLarga = longitudPromedioPalabras
                                    "aabbccddeeffgghhii"
longitudPromedioDiferentesTamanios2Palabras = longitudPromedioPalabras
                                    "aabbcc aabb"
longitudPromedioDiferentesTamanios3Palabras = longitudPromedioPalabras
                                    "aabbcc aabb ad"


longitudPromedioTest1 = TestCase (assertEqual "Letras sueltas"
                                  1 longitudPromedioLetrasSueltas)
longitudPromedioTest2 = TestCase (assertEqual "Dos letras por palabra"
                                  2 longitudPromedioDosLetrasXPalabra)
longitudPromedioTest3 = TestCase (assertEqual "Una palabra larga"
                                  18 longitudPromedioUnaPalabraLarga)
longitudPromedioTest4 = TestCase (assertEqual "Diferentes tamanios, 2 palabras"
                                  5 longitudPromedioDiferentesTamanios2Palabras)
longitudPromedioTest5 = TestCase (assertEqual "Diferentes tamanios, 3 palabras"
                                  4 longitudPromedioDiferentesTamanios3Palabras)
```

2.3. Ejercicio 3

```
cuentasDelVacio = cuentas [""]
cuentasDelEspacio = cuentas [" "]
cuentasUnaPalabraUnaRepeticion = cuentas ["Una"]
cuentasUnaPalabraVariasRepeticiones = cuentas ["Una", "Una", "Una"]
cuentasMuchasPalabrasUnaRepeticion = cuentas  ["Una", "Dos", "Tres", "Cuatro"]
cuentasMuchasPalabrasMuchasRepeticiones = cuentas  ["Una", "Dos", "Tres", "Cuatro",
                                                    "Una", "Dos", "Tres", "Cuatro",
                                                    "Una", "Dos", "Tres", "Cuatro"]

cuentasTest1 = TestCase (assertEqual "Vacio" [(1,"")] cuentasDelVacio)
cuentasTest2 = TestCase (assertEqual "Espacio" [(1," ")] cuentasDelEspacio)
cuentasTest3 = TestCase (assertEqual "Una palabra una repeticion"
                                  [(1,"Una")] cuentasUnaPalabraUnaRepeticion)
cuentasTest4 = TestCase (assertEqual "Una palabra varias repeticiones"
                                  [(3,"Una")] cuentasUnaPalabraVariasRepeticiones)
cuentasTest5 = TestCase (assertEqual "Muchas palabras, una repeticion"
                                  [(1,"Una"),(1,"Dos"),(1,"Tres"),(1,"Cuatro")]
                                  cuentasMuchasPalabrasUnaRepeticion)
```

```
cuentasTest6 = TestCase (assertEqual "Muchas palabras, muchas repeticiones"
                                [(3,"Una"),(3,"Dos"),(3,"Tres"),(3,"Cuatro")]
                                cuentasMuchasPalabrasMuchasRepeticiones)
```

2.4. Ejercicio 4

```
repeticionesPromedioVacio = repeticionesPromedio ""
repeticionesPromedioUnaPalabra = repeticionesPromedio "Una"
repeticionesPromedioUnaPalabra3Veces = repeticionesPromedio "Una Una Una"
repeticionesPromedioMuchasPalabras1Vez =
                                repeticionesPromedio  "Una Dos Tres Cuatro"
repeticionesPromedioMuchasPalabras3Veces =
                                repeticionesPromedio  "Una Dos Tres Cuatro
                                                       Una Dos Tres Cuatro
                                                       Una Dos Tres Cuatro"

repeticionesPromedioTest3 = TestCase (assertEqual
                                      "Una palabra una repeticion"
                                      1 repeticionesPromedioUnaPalabra)
repeticionesPromedioTest4 = TestCase (assertEqual
                                      "Una palabra varias repeticiones"
                                      3 repeticionesPromedioUnaPalabra3Veces)
repeticionesPromedioTest5 = TestCase (assertEqual
                                      "Muchas palabras, una repeticion"
                                      1 repeticionesPromedioMuchasPalabras1Vez)
repeticionesPromedioTest6 = TestCase (assertEqual
                                      "Muchas palabras, muchas repeticiones"
                                      3 repeticionesPromedioMuchasPalabras3Veces)
```

2.5. Ejercicio 5

```
frecuenciasTokensElToken = (head frecuenciaTokens) "_"
frecuenciasTokensUnaPalabraSinToken = (head frecuenciaTokens) "Una"
frecuenciasTokensUnaPalabraConToken = (head frecuenciaTokens) "Una_"
frecuenciasTokensMuchasPalabrasSinToken =
                                (head frecuenciaTokens)  "Una Dos Tres Cuatro"
frecuenciasTokensPalabrasConToken =
                                (head frecuenciaTokens) "Una_Dos_Tres"

frecuenciasTokensTest2 = TestCase (assertEqual "Solo el Token"
                                   1 frecuenciasTokensElToken)
frecuenciasTokensTest3 = TestCase (assertEqual "Una palabra sin Token"
                                   0 frecuenciasTokensUnaPalabraSinToken)
frecuenciasTokensTest4 = TestCase (assertEqual "Una palabra con Token"
                                   0.25 frecuenciasTokensUnaPalabraConToken)
frecuenciasTokensTest5 = TestCase (assertEqual "Muchas palabras, sin Token"
                                   0 frecuenciasTokensMuchasPalabrasSinToken)
frecuenciasTokensTest6 = TestCase (assertEqual "Muchas palabras, con Token"
                                   (formatFloatN 0.1666667 3)
                                   (formatFloatN
                                         frecuenciasTokensPalabrasConToken 3))
```

2.6. Ejercicio 6 y Ejercicio 7

```
checkNormalizado :: [Float] -> Bool
checkNormalizado xs = ((head xs)) >= 0 && ((head xs) <= 1)
                      && ((head (tail xs)) <= 1) &&  ((head (tail xs)) >= 0)

estaTodoNormalizado :: [[Float]] -> Bool
```

```
estaTodoNormalizado xss = foldr (\parDeFeatures ->
                          \rec ->
                              (checkNormalizado parDeFeatures) && rec) True xss

aCheckearNormalizado =
        extraerFeatures [longitudPromedioPalabras, repeticionesPromedio]
                        ["b=a", "a = 2; a = 4", "asd", "1233243453",
                        "assadasdasasd", "123 as"]

normalizarExtractorTest1 = TestCase (assertEqual "Esta normalizado"
                           True (estaTodoNormalizado aCheckearNormalizado))
```

2.7. Ejercicio 8

```
distanciaEuclideanaCero = distEuclideana [0,0] [0,0]
distanciaEuclideana2 = distEuclideana [1,0] [0,1]
distanciaEuclideana4 = distEuclideana [1,1] [1,1]

distanciaCosenoCero = distEuclideana [0,0] [0,0]
distanciaCoseno2 = distEuclideana [1,0] [0,1]
distanciaCoseno4 = distEuclideana [1,1] [1,1]

distanciaEuclideanaTest1 = TestCase (assertEqual "DistEuclideana Ceros"
                                    0 distanciaEuclideanaCero)
distanciaEuclideanaTest2 = TestCase (assertEqual "DistEuclideana 1 0, 0 1"
                                    (formatFloatN 1.414214 3)
                                    (formatFloatN distanciaEuclideana2 3))
distanciaEuclideanaTest3 = TestCase (assertEqual "DistEuclideana 1 1, 1 1"
                                    0 distanciaEuclideana4)

distanciaCosenoTest1 = TestCase (assertEqual "DistCoseno Ceros"
                                0 distanciaCosenoCero)
distanciaCosenoTest2 = TestCase (assertEqual "DistCoseno 1 0, 0 1"
                                (formatFloatN 1.414214 3)
                                (formatFloatN distanciaCoseno2 3))
distanciaCosenoTest3 = TestCase (assertEqual "DistCoseno 1 1, 1 1"
                                0 distanciaCoseno4)
```

2.8. Ejercicio 9

```
knnEnun = (knn 2 [[0,1],[0,2],[2,1],[1,1],[2,3]]
               ["i","i","f","f","i"] distEuclideana) [1,1]
knnTest = TestCase (assertEqual "Knn Enunciado" "f" knnEnun)
```

2.9. Ejercicio 10

```
xsTestEj10 = [[1,1],[2,2],[3,3],[4,4],[5,5],[6,6],[7,7]] :: Datos
yTestEj10 = ["1","2","3","4","5","6","7"]
(x_train, x_val, y_train, y_val) = separarDatos xsTestEj10 yTestEj10 3 2

separarDatosTest = TestCase (assertEqual "SepararDatos"
                            (x_train, y_train)
                            ([[1.0,1.0],[2.0,2.0],[5.0,5.0],[6.0,6.0]],
                             ["1","2","5","6"]))
```

2.10. Ejercicio 11

```
accuracy0 = accuracy ["i", "i", "i", "i", "i"] ["f", "f", "f", "f", "f"]
accuracy60 = accuracy ["f", "f", "i", "i", "f"] ["i", "f", "i", "f", "f"]
```

```
accuracy100 = accuracy ["f", "f", "f", "f", "f"] ["f", "f", "f", "f", "f"]

accuracyTest1 = TestCase (assertEqual "0\% accuracy" 0 accuracy0)
accuracyTest2 = TestCase (assertEqual "60\% accuracy" 0.6 accuracy60)
accuracyTest3 = TestCase (assertEqual "100\% accuracy" 1 accuracy100)
```

2.11. Ejercicio 12

```
twoFoldValidation = nFoldCrossValidation 2
                                        [[1,1],[2,2],[3,3],[4,4],[5,5],
                                         [6,6],[7,7],[8,8],[9,9],[10,10]]
                                        ["i","f","f","i","i","i","i","i","i","i"]
threeFoldValidation = nFoldCrossValidation 3
                                        [[1,1],[2,2],[3,3],[4,4],[5,5],
                                         [6,6],[7,7],[8,8],[9,9],[10,10]]
                                        ["i","f","f","i","i","i","i","i","i","i"]
fourFoldValidation = nFoldCrossValidation 4
                                        [[1,1],[2,2],[3,3],[4,4],[5,5],
                                         [6,6],[7,7],[8,8],[9,9],[10,10]]
                                        ["i","f","f","i","i","i","i","i","i","i"]

test2FoldValidation = TestCase (assertEqual "2FoldValidation"
                                0.8 twoFoldValidation)
test3FoldValidation = TestCase (assertEqual "3FoldValidation"
                                (formatFloatN 0.7777777 3)
                                (formatFloatN threeFoldValidation 3))
test4FoldValidation = TestCase (assertEqual "4FoldValidation"
                                0.75 fourFoldValidation)
```