Práctica minería de flujos de datos

Carlos Manuel Sequí Sánchez

25 de abril de 2019

alumno: Carlos Manuel Sequí Sánchez asignatura: Series temporales y minería de flujos de datos Master: ciencias de datos e ingenieria de computadores Trabajo autónomo I: Series Temporales Fecha entrega: 25-4-2019

TEORÍA

Respuestas tipo test

1.El aprendizaje incremental es útil cuando se quiere ganar eficiencia
2.La minería de flujo de datos se considera cuando el problema genera datos continuamente
3.La cota de Hoeffding sirve para saber cuándo hay suficientes datos para una estimación fiable
4.¿Qué características de clusters mantiene el algoritmo BIRCH? Suma lineal, suma cuadrática y número de objetos
5.¿El algoritmo Stream maneja concept drift? NO
6.¿Qué es el concept drift? Cambios en la dinámica del problema
7.¿Cómo gestiona CVFDT el concept drift? Mantiene árboles alternativos
8.¿Por qué es útil el ensemble learning en concept drift? Porque aprovecha la diversidad que se genera en los cambios
9.¿Cuál es más eficiente entre DDM y ADWIN? DDM es más eficiente
10.¿Por qué es controvertida la clasificación en flujo de datos? Porque se requiere al oráculo por siempre
11.¿Cómo gestiona ClueStream el concept drif? Mantiene información dobre el tiempo
12.¿Por qué es complejo generar reglas de asociación en flujos de datos?

PRÁCTICAS

2.1. Entrenamiento offline (estacionario) y evaluación posterior.

Entrenar un clasificador HoeffdingTree offline (estacionario, aprender modelo únicamente), sobre un total de 1.000.000 de instancias procedentes de un flujo obtenido por el generador WaveFormGenerator con semilla aleatoria igual a 2. Evaluar posteriormente (sólo evaluación) con 1.000.000 de instancias generadas por el mismo tipo de generador, con semilla aleatoria igual a 4. Repita el proceso varias veces con la misma semilla en evaluación y diferentes semillas en entrenamiento, para crear una población de resultados. Anotar como resultados los valores de porcentajes de aciertos en la clasificación y estadístico Kappa.

El comando que vamos a utilizar para entrenar el el siguiente:
for /L %e in (1,1,20) do java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateModel -m (LearnModel -l trees.HoeffdingTree -s (generators.WaveformGenerator -i 2) -m 1000000) -s (generators.WaveformGenerator -i 4) -i 1000000"

A continuación generamos la población de 20 (tanto de Hoeffding Trees como de Adaptative Hoeffding Trees para el siguiente ejercicio) a partir de dicho comando:

```
for /L %e in (1,1,2) do
    java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask
    "EvaluateModel -m (LearnModel -l trees.HoeffdingTree
    -s (generators.WaveformGenerator -i %e) -m 1000000)
    -s (generators.WaveformGenerator -i 4) -i 1000000"
    > ejHF%e.csv

for /L %e in (1,1,2) do
    java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask
    "EvaluateModel -m (LearnModel -l trees.HoeffdingAdaptiveTree
    -s (generators.WaveformGenerator -i %e) -m 1000000)
    -s (generators.WaveformGenerator -i 4) -i 1000000"
    > ejHFA%e.csv
```

Leemos los datos de Hoeffding Trees y creamos su población a partir de los ficheros generados con el script

```
datosHF1 = as.data.frame(matrix(0,nrow = 30,ncol = 2))
names(datosHF1) = c("accuracy","kappa")

for(i in 1:30)
{
  nombreFichero = paste(c(paste(c("./datosEj1/ejHF",i),collapse=""),".txt"),collapse="")
  file = readLines(nombreFichero)

  # almacenamos el porcentaje de clasificaciones correctas
  a = sub('.*= ', '', file[2])
  a = as.double(sub(',','.',a))
  datosHF1[i,"accuracy"] = a

  # almacenamos el porcentaje de clasificaciones correctas
  b = sub('.*= ', '', file[3])
  b = as.double(sub(',','.',b))
  datosHF1[i,"kappa"] = b
}

datosHF1
```

```
##    accuracy  kappa
## 1    84.509 76.765
## 2    84.512 76.770
## 3    84.590 76.887
## 4    84.666 77.001
## 5    84.481 76.723
## 6    84.342 76.514
## 7    84.799 77.200
## 8    84.153 76.231
## 9    84.641 76.963
## 10   84.578 76.869
## 11   84.539 76.810
## 12   84.457 76.688
## 13   84.369 76.555
## 14   84.547 76.822
## 15   84.648 76.974
## 16   84.626 76.940
## 17   84.513 76.772
```

```
## 18   84.434 76.653
## 19   84.605 76.910
## 20   84.568 76.853
## 21   84.518 76.779
## 22   84.657 76.988
## 23   84.543 76.815
## 24   84.754 77.132
## 25   84.646 76.971
## 26   84.586 76.880
## 27   84.488 76.734
## 28   83.529 75.294
## 29   84.591 76.888
## 30   84.505 76.759
```

Repetir el paso anterior, sustituyendo el clasificador por HoeffdingTree adaptativo.

El comando que vamos a utilizar para entrenar el el siguiente:
for /L %e in (1,1,20) do java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateModel -m (LearnModel -l trees.HoeffdingAdaptiveTree -s (generators.WaveformGenerator -i %e) -m 1000000) -s (generators.WaveformGenerator -i 4) -i 1000000" > ejHF%e.txt}

Como ya habíamos generado los ficheros de datos del Hoeffding adaptativo, leemos los datos de Adaptative Hoeffding Trees y creamos su población a partir de los ficheros generados con el script

```
datosHFA1 = as.data.frame(matrix(0,nrow = 20,ncol = 2))
names(datosHFA1) = c("accuracy","kappa")

for(i in 1:20)
{
  nombreFichero = paste(c(paste(c("./datosEj1/ejHFA",i),collapse=""),".txt"),collapse="")
  file = readLines(nombreFichero)

  # almacenamos el porcentaje de clasificaciones correctas
  a = sub('.*= ', '', file[2])
  a = as.double(sub(',','.',a))
  datosHFA1[i,"accuracy"] = a

  # almacenamos el porcentaje de clasificaciones correctas
  b = sub('.*= ', '', file[3])
  b = as.double(sub(',','.',b))
  datosHFA1[i,"kappa"] = b
}

datosHFA1
```

```
##    accuracy  kappa
## 1    84.521 76.783
## 2    84.474 76.712
## 3    84.416 76.625
## 4    84.465 76.699
## 5    84.262 76.395
## 6    84.368 76.554
## 7    84.271 76.408
## 8    84.243 76.367
```

```
## 9    84.478 76.719
## 10   84.326 76.491
## 11   84.371 76.558
## 12   84.416 76.627
## 13   84.498 76.749
## 14   84.415 76.624
## 15   84.229 76.345
## 16   84.328 76.494
## 17   84.358 76.539
## 18   84.456 76.685
## 19   84.451 76.679
## 20   84.459 76.690
```

Responda a la pregunta: ¿Cree que algún clasificador es significativamente mejor que el otro en este tipo de problemas? Razone su respuesta.

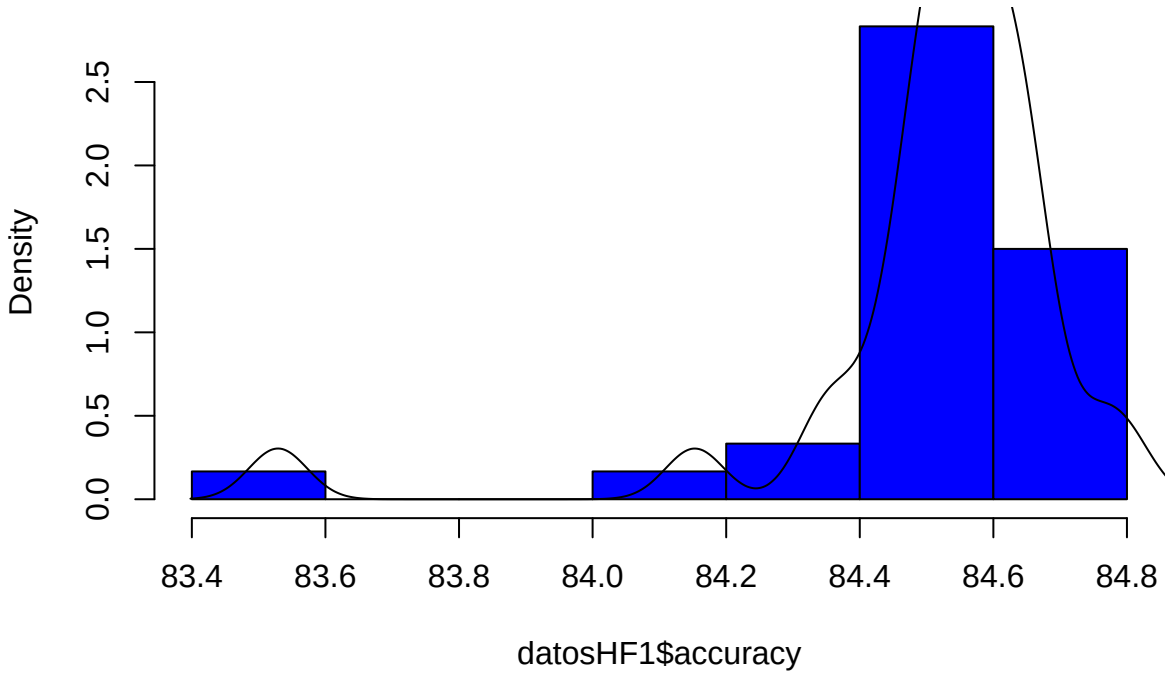
Para averiguar si alguno de los clasificadores es mejor que el otro procedemos a utilizar tests estadísticos. Para ser que test utilizar, testeamos la normalidad de ambas poblaciones con sus histogramas y tests de Saphiro Wilk.

Para Hoeffding Trees:

```
shapiro.test(datosHF1$accuracy)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  datosHF1$accuracy
## W = 0.68496, p-value = 9.496e-07
```

```
hist(datosHF1$accuracy, col="blue", prob = TRUE)
lines(density(datosHF1$accuracy))
```



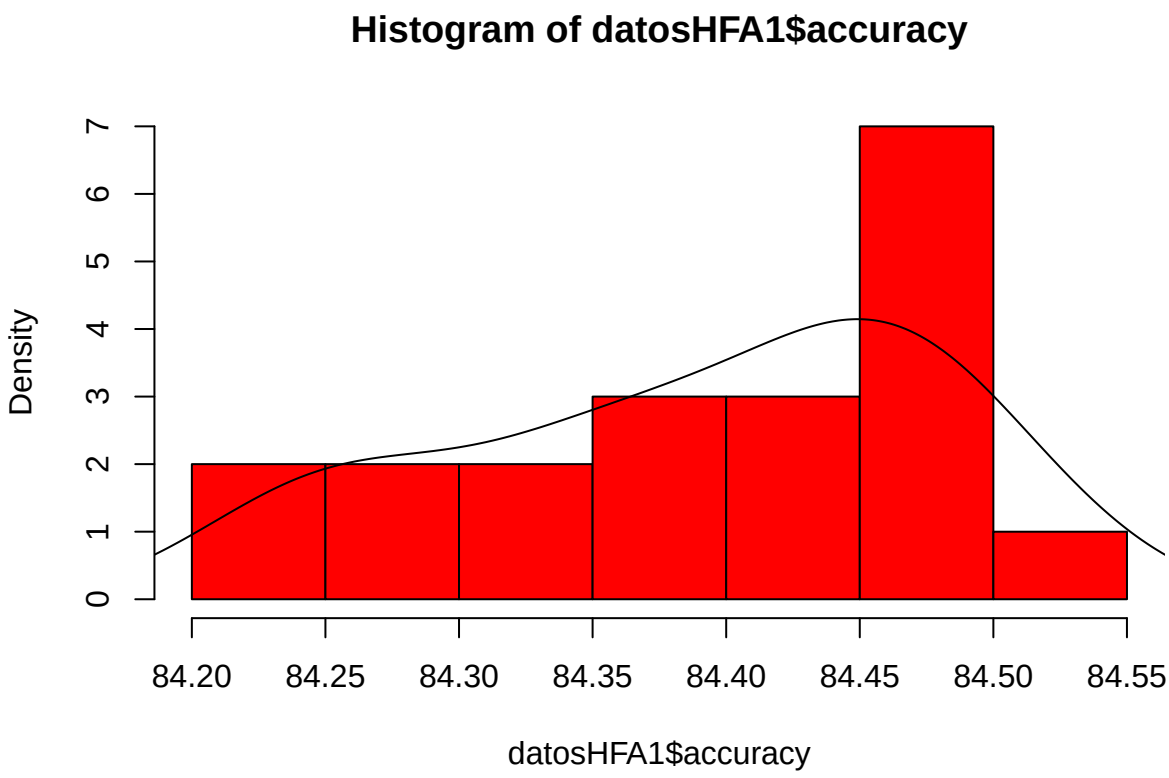
El test de Saphiro Wilk, debido al bajo valor del p-value, nos indica que los datos no siguen una distribución normal, sin embargo, tal como podemos observar en el histograma, los datos no siguen una distribución normal debido a la oblicuidad que poseen (skewness), aunque realmente poseen una gran cantidad de datos centrados en un punto, lo que puede ser indicativo de ser buen modelo.

Para Adaptive Hoeffding Trees:

```
shapiro.test(datosHFA1$accuracy)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  datosHFA1$accuracy
## W = 0.93288, p-value = 0.1754
```

```
hist(datosHFA1$accuracy, col="red", prob = TRUE)
lines(density(datosHFA1$accuracy))
```



Al contrario que el modelo anterior, según Saphiro Wilk, estos datos sí que siguen una distribución normal, ya que el p-value es superior a 0.05. Podemos corroborarlo con el histograma que se muestra.

Al tener tener dos poblaciones siguiendo una distribución normal y otra no, nos decidiremos por utilizar el de Wilcoxon para comprobar si existen diferencias significativas en las soluciones generadas por ambos:

```
wilcox.test(datosHF1$accuracy,datosHFA1$accuracy, exact = F)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  datosHF1$accuracy and datosHFA1$accuracy
## W = 511, p-value = 3.064e-05
## alternative hypothesis: true location shift is not equal to 0
```

Al obtener un p-value inferior a 0.05 concretamos que ambos modelos sí poseen diferencias significativas en sus accuracy, por lo que nos decantamos por el modelo que tiene más datos concentrados por encima de 84.4% que el otro, es decir, el Hoeffding Trees estacionario.

2.2. Entrenamiento online.

Entrenar un clasificador HoeffdingTree online, mediante el método Interleaved Test-Then-Train, sobre un total de 1.000.000 de instancias procedentes de un flujo obtenido por el generador WaveFormGenerator con semilla aleatoria igual a 2, con una frecuencia de muestreo igual a 10.000. Pruebe con otras semillas aleatorias para crear una población de resultados. Anotar los valores de porcentajes de aciertos en la clasificación y estadístico Kappa.

El comando que vamos a utilizar para entrenar el el siguiente:
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l trees.HoeffdingTree -s (generators.WaveformGenerator -i 2) -i 1000000 -f 10000"

Creamos la población de resultados con distintas semillas igual que en el ejercicio anterior.

Leemos los datos de Hoeffding Trees y creamos su población a partir de los ficheros generados con el script

```
accuracy = array(dim = 30)
kapp = array(dim = 30)
for (i in 1:30)
{
  nombreFichero = paste(c(paste(c("./datosEj2/ej2HF",i),collapse=""),".csv"),collapse="")
  nuevoFichero = read.csv(nombreFichero)
  nuevoDatoAcc = nuevoFichero$classifications.correct..percent.[length(nuevoFichero$classifications.cor
  nuevoDatoKap = nuevoFichero$Kappa.Statistic..percent.[length(nuevoFichero$Kappa.Statistic..percent.)]
  accuracy[i] = nuevoDatoAcc
  kapp[i] = nuevoDatoKap
}
datosHF1 = as.data.frame(cbind(accuracy,kapp))
names(datosHF1) = c("accuracy","kappa")
datosHF1
```

```
##    accuracy    kappa
## 1   83.8903 75.83624
## 2   83.7851 75.67750
## 3   83.8876 75.82954
## 4   84.0451 76.06946
## 5   83.8402 75.75999
## 6   83.9062 75.85906
## 7   83.8867 75.82928
## 8   83.8687 75.80284
## 9   83.7875 75.68172
## 10  83.8479 75.77149
## 11  83.7456 75.61783
## 12  83.8392 75.75913
## 13  83.9761 75.96426
## 14  83.8801 75.81921
## 15  83.9843 75.97620
## 16  83.8343 75.75183
## 17  83.8963 75.84450
## 18  83.8406 75.76171
## 19  83.7880 75.68285
## 20  83.8152 75.72332
## 21  83.8623 75.79297
## 22  83.9778 75.96662
```

```
## 23  83.8462 75.76955
## 24  83.8860 75.82860
## 25  83.9650 75.94696
## 26  83.9033 75.85570
## 27  83.8202 75.72857
## 28  83.9000 75.85055
## 29  83.8360 75.75415
## 30  83.8987 75.84717
```

Repetir el paso anterior, sustituyendo el clasificador por HoeffdingTree adaptativo

El comando que vamos a utilizar para entrenar el el siguiente:
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l trees.HoeffdingAdaptativeTree -s (generators.WaveformGenerator -i 2) -i 1000000 -f 10000"

Creamos la población de resultados con distintas semillas igual que en el ejercicio anterior.

Leemos los datos de Hoeffding Trees y creamos su población a partir de los ficheros generados con el script

```
accuracy = array(dim = 30)
kapp = array(dim = 30)
for (i in 1:30)
{
  nombreFichero = paste(c(paste(c("./datosEj2/ej2HFA",i),collapse=""),".csv"),collapse="")
  nuevoFichero = read.csv(nombreFichero)
  nuevoDatoAcc = nuevoFichero$classifications.correct..percent.[length(nuevoFichero$classifications.cor
  nuevoDatoKap = nuevoFichero$Kappa.Statistic..percent.[length(nuevoFichero$Kappa.Statistic..percent.)]
  accuracy[i] = nuevoDatoAcc
  kapp[i] = nuevoDatoKap
}
datosHFA1 = as.data.frame(cbind(accuracy,kapp))
names(datosHF1) = c("accuracy","kappa")
datosHFA1
```

```
##    accuracy     kapp
## 1   83.8042 75.70717
## 2   83.7313 75.59676
## 3   83.7875 75.67920
## 4   83.7961 75.69604
## 5   83.7144 75.57129
## 6   83.8406 75.76071
## 7   83.7784 75.66688
## 8   83.8968 75.84507
## 9   83.8282 75.74278
## 10  83.9000 75.84955
## 11  83.7407 75.61050
## 12  83.7414 75.61238
## 13  83.8943 75.84157
## 14  83.8576 75.78550
## 15  83.8748 75.81208
## 16  83.8876 75.83154
## 17  83.7386 75.60797
## 18  83.7614 75.64276
## 19  83.7500 75.62571
## 20  83.8226 75.73431
```

```
## 21  83.8498 75.77413
## 22  83.8633 75.79487
## 23  83.6508 75.47647
## 24  83.8110 75.71615
## 25  83.8657 75.79789
## 26  83.8185 75.72845
## 27  83.8520 75.77618
## 28  83.8143 75.72198
## 29  83.8220 75.73313
## 30  83.9733 75.95890
```

Responda a la pregunta: ¿Cree que algún clasificador es mejor que el otro en este tipo de problemas? Razone su respuesta.

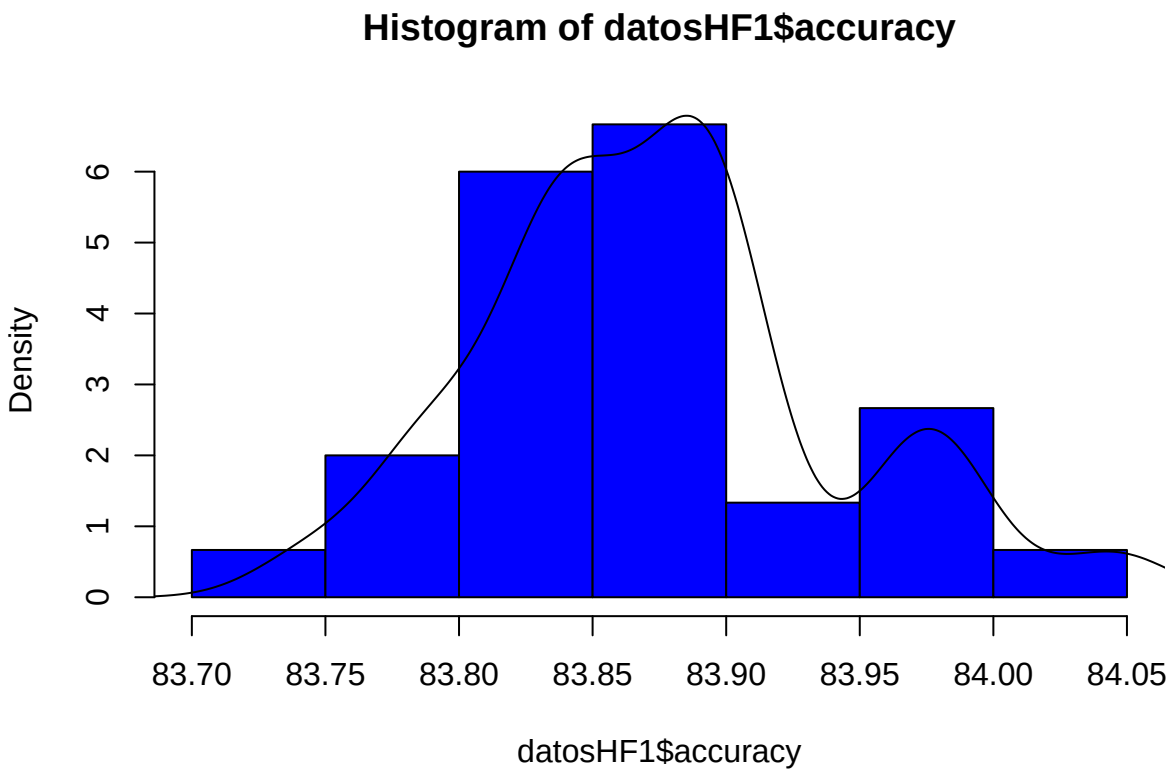
Para averiguar si alguno de los clasificadores es mejor que el otro, tal como hicimos en el ejercicio anterior, procedemos a utilizar tests estadísticos. Para ser que test utilizar, testeamos la normalidad de ambas poblaciones con sus histogramas y tests de Saphiro Wilk.

Para Hoeffding Trees:

```
shapiro.test(datosHF1$accuracy)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  datosHF1$accuracy
## W = 0.9602, p-value = 0.3135
```

```
hist(datosHF1$accuracy, col="blue", prob = TRUE)
lines(density(datosHF1$accuracy))
```

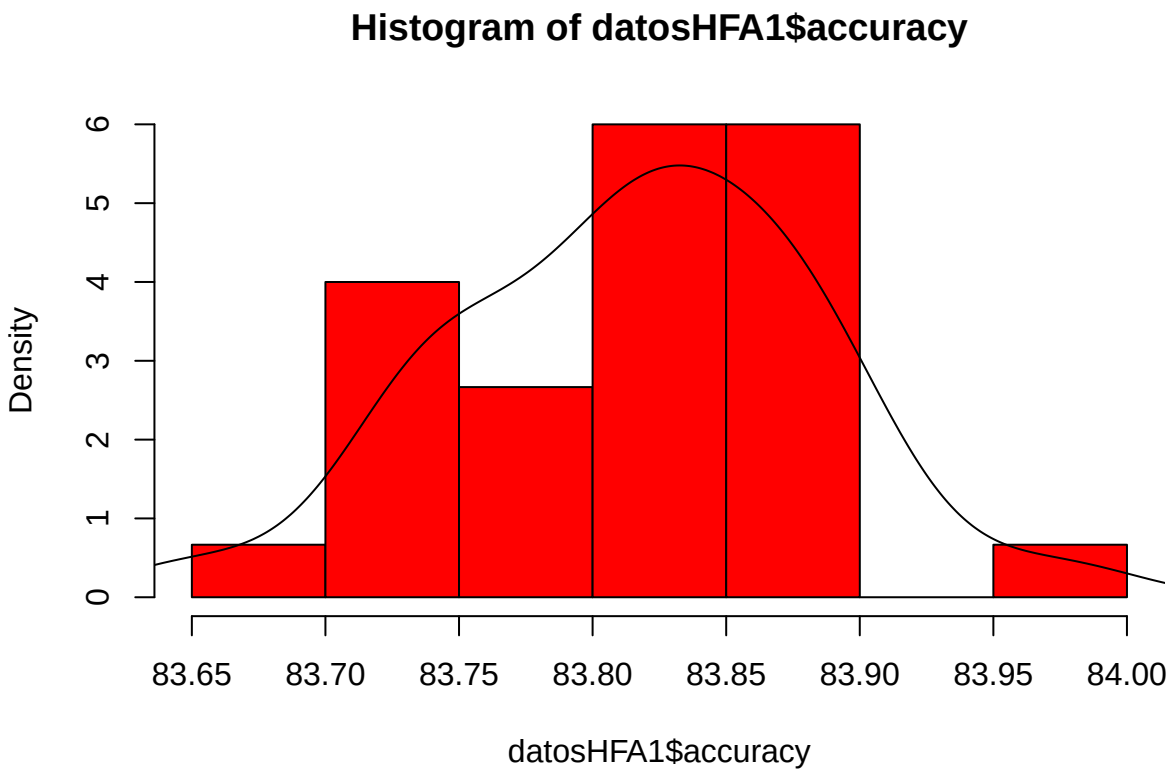


Para Adaptive Hoeffding Trees:

```
shapiro.test(datosHFA1$accuracy)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  datosHFA1$accuracy
## W = 0.98491, p-value = 0.9357
```

```
hist(datosHFA1$accuracy, col="red", prob = TRUE)
lines(density(datosHFA1$accuracy))
```



Como podemos observar ambas poblaciones tienen un p-value en el test de Saphiro wilk por encima de 0.05, por tanto concluimos que ambos siguen una distribución normal, asñi que haremos uso de un test paramétrico:

```
t.test(datosHF1,datosHFA1)
```

```
##
##  Welch Two Sample t-test
##
## data:  datosHF1 and datosHFA1
## t = 0.099364, df = 118, p-value = 0.921
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.398944  1.546750
## sample estimates:
## mean of x mean of y
##  79.84332  79.76942
```

Al resultar un p-value superior a 0.05, concretamos con alta confianza que la media de las poblaciones de datos no difieren de manera significativa, por lo que concretamos que ninguno de los modelos es mejor que el otro. Tiene sentido el resultado obtenido en comparación con el anterior, debido a que gracias al aprendizaje online el modelo Adaptative Hoeffding Trees consigue ponerse al nivel de Hoeffding estacionario en cuanto a acierto.

2.3. Entrenamiento online en datos con concept drift.

Entrenar un clasificador HoeffdingTree online, mediante el método Interleaved Test-Then-Train, sobre un total de 2.000.000 de instancias muestreadas con una frecuencia de 100.000, sobre datos procedentes de un generador de flujos RandomRBFGeneratorDrift, con semilla aleatorio igual a 1 para generación de modelos y de instancias, generando 2 clases, 7 atributos, 3 centroides en el modelo, drift en todos los centroides y velocidad de cambio igual a 0.001. Pruebe con otras semillas aleatorias. Anotar los valores de porcentajes de aciertos en la clasificación y estadístico Kappa. Compruebe la evolución de la curva de aciertos en la GUI de MOA.

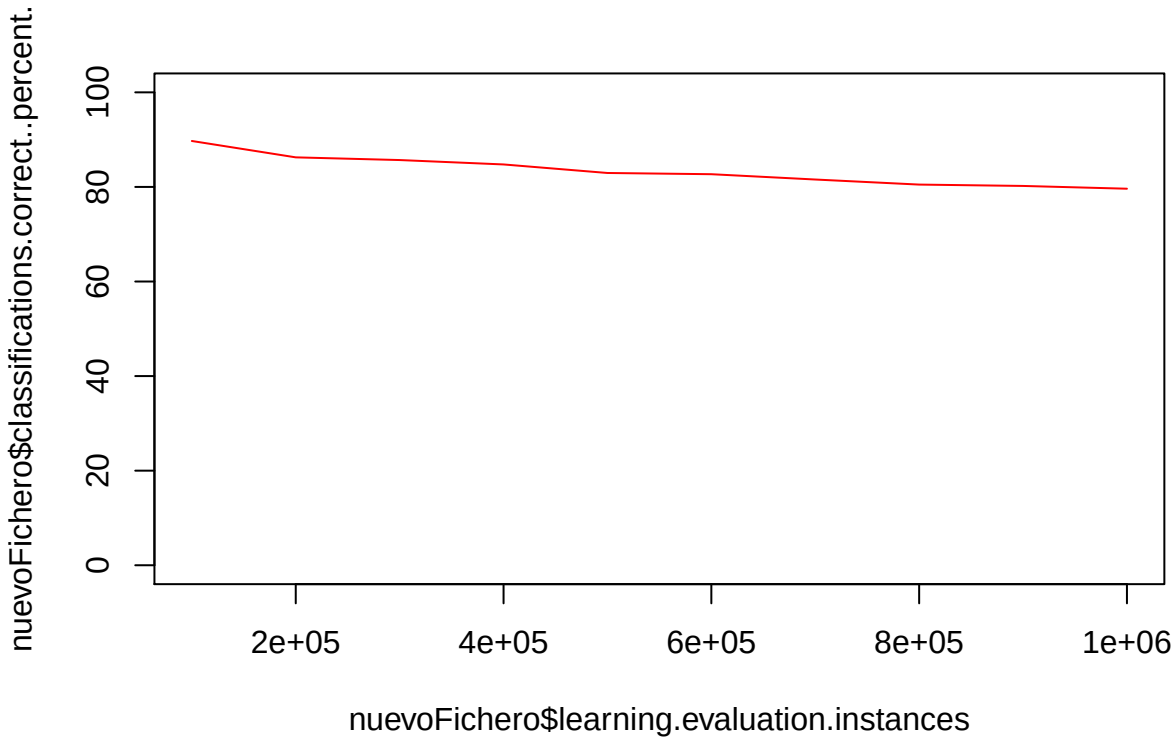
El comando que vamos a utilizar para entrenar el el siguiente:
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l trees.HoeffdingTree -s (generators.RandomRBFGeneratorDrift -r 1 -i 1 -c 2 -a 7 -n 3 -s 0.001 -k 3) -i 1000000 -f 100000"

Creamos la población de resultados con distintas semillas igual que en el ejercicio anterior.

Leemos los datos de Hoeffding Trees y creamos su población a partir de los ficheros generados con el script

```
accuracy = array(dim = 30)
kapp = array(dim = 30)
for (i in 1:30)
{
  nombreFichero = paste(c(paste(c("./datosEj3/ej3",i),collapse=""),".csv"),collapse="")
  nuevoFichero = read.csv(nombreFichero)
  nuevoDatoAcc = nuevoFichero$classifications.correct..percent.[length(nuevoFichero$classifications.cor
  nuevoDatoKap = nuevoFichero$Kappa.Statistic..percent.[length(nuevoFichero$Kappa.Statistic..percent.)]
  accuracy[i] = nuevoDatoAcc
  kapp[i] = nuevoDatoKap

  # Comprobamos la evolución de la curva de aciertos:
  if(i==1)
  {
    plot(nuevoFichero$learning.evaluation.instances,
     nuevoFichero$classifications.correct..percent.,
     "l", ylim = c(0,100), col = "red")
  }
}
```




```
datosHF1 = as.data.frame(cbind(accuracy,kapp))
names(datosHF1) = c("accuracy","kappa")
datosHF1
```

```
##    accuracy     kappa
## 1   79.6246 59.175309
## 2   84.0631 32.126047
## 3   76.0180 48.746459
## 4   77.4269 53.994961
## 5   74.0719 47.186231
## 6   83.8462 28.398960
## 7   63.1274 25.182392
## 8   78.4236 16.052193
## 9   98.1723 19.761424
## 10  66.1212 30.470425
## 11  77.0990 54.094039
## 12  61.3644 19.682953
## 13  70.3873 28.199609
## 14  77.4524 34.607491
## 15  96.9061  7.570966
## 16  83.1698 62.846431
## 17  72.8917 32.475568
## 18  97.5555  7.966872
## 19  84.7143 19.876100
## 20  77.7282 41.495610
## 21  69.4774 15.491647
## 22  75.7646 45.343128
```

```
## 23  79.2858 56.185548
## 24  69.2202 30.312455
## 25  66.6869 28.480743
## 26  73.6728 46.567448
## 27  70.2505 25.055482
## 28  73.5802 36.185264
## 29  81.1943 59.694112
## 30  63.7098 26.064701
```

Repetir el paso anterior, sustituyendo el clasificador por HoeffdingTree adaptativo

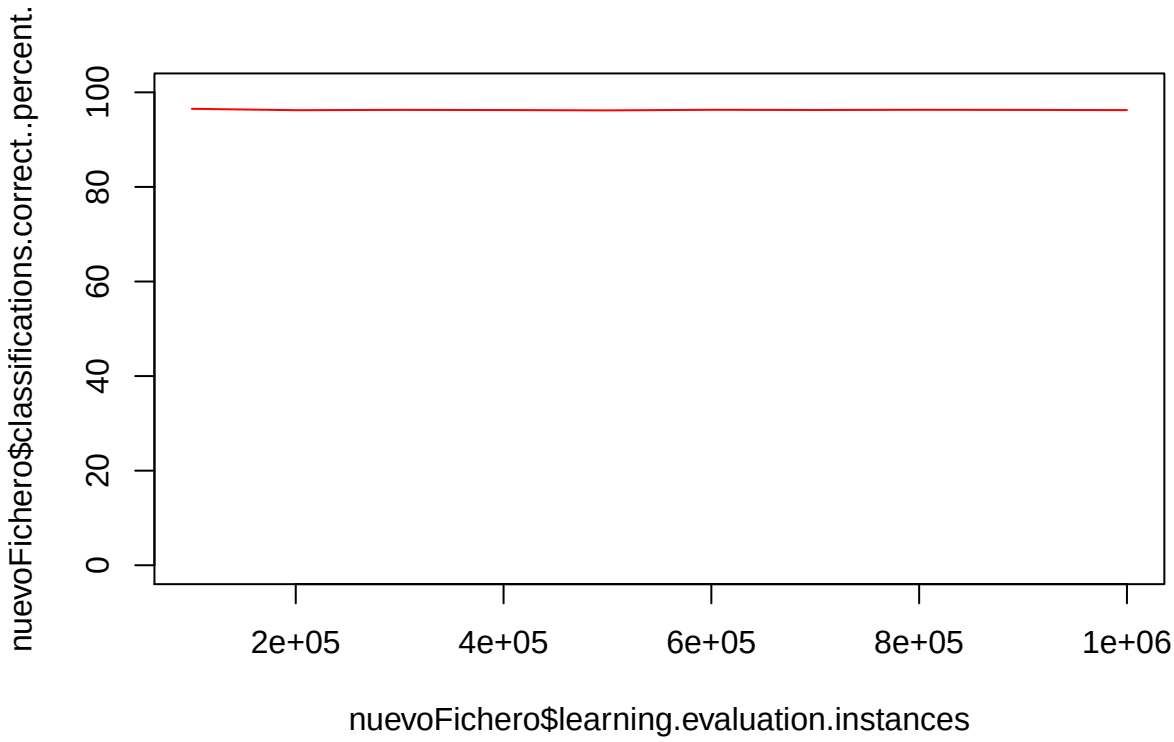
El comando que vamos a utilizar para entrenar el el siguiente:
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l trees.HoeffdingAdaptativeTree -s (generators.RandomRBFGeneratorDrift -r 1 -i 1 -c 2 -a 7 -n 3 -s 0.001 -k 3) -i 1000000 -f 100000"

Creamos la población de resultados con distintas semillas igual que en el ejercicio anterior.

Leemos los datos de Adaptative Hoeffding Trees y creamos su población a partir de los ficheros generados con el script

```
accuracy = array(dim = 30)
kapp = array(dim = 30)
for (i in 1:30)
{
  nombreFichero = paste(c(paste(c("./datosEj3/eja3",i),collapse=""),".csv"),collapse="")
  nuevoFichero = read.csv(nombreFichero)
  nuevoDatoAcc = nuevoFichero$classifications.correct..percent.[length(nuevoFichero$classifications.cor
  nuevoDatoKap = nuevoFichero$Kappa.Statistic..percent.[length(nuevoFichero$Kappa.Statistic..percent.)]
  accuracy[i] = nuevoDatoAcc
  kapp[i] = nuevoDatoKap

  # Comprobamos la evolución de la curva de aciertos:
  if(i==1)
  {
    plot(nuevoFichero$learning.evaluation.instances,
     nuevoFichero$classifications.correct..percent.,
     "l", ylim = c(0,100), col = "red")
  }
}
```




```
datosHFA1 = as.data.frame(cbind(accuracy,kapp))
names(datosHFA1) = c("accuracy","kappa")
datosHFA1
```

```
##    accuracy    kappa
## 1   96.2456 92.48482
## 2   92.7486 74.40884
## 3   96.3494 92.27763
## 4   96.0264 91.86172
## 5   91.1620 81.98867
## 6   94.9624 81.92326
## 7   88.8138 77.50804
## 8   90.7426 73.47702
## 9   98.5414 45.02214
## 10  87.8436 75.21215
## 11  96.0276 91.99684
## 12  86.1809 71.84526
## 13  94.2937 87.17514
## 14  91.9916 79.05731
## 15  97.3782 46.05219
## 16  97.4486 94.40259
## 17  93.4501 84.75879
## 18  97.8093 48.71953
## 19  92.2843 69.92409
## 20  95.6192 89.40132
## 21  86.6338 68.98161
## 22  89.6471 77.51989
```

```
## 23  96.1478 91.85386
## 24  92.1309 82.68515
## 25  85.5665 69.78983
## 26  97.0977 94.13481
## 27  90.4430 77.83045
## 28  94.3825 86.89818
## 29  96.4446 92.33815
## 30  91.4514 82.74081
```

Responda a la pregunta: ¿Cree que algún clasificador es mejor que el otro en este tipo de problemas? Razone su respuesta

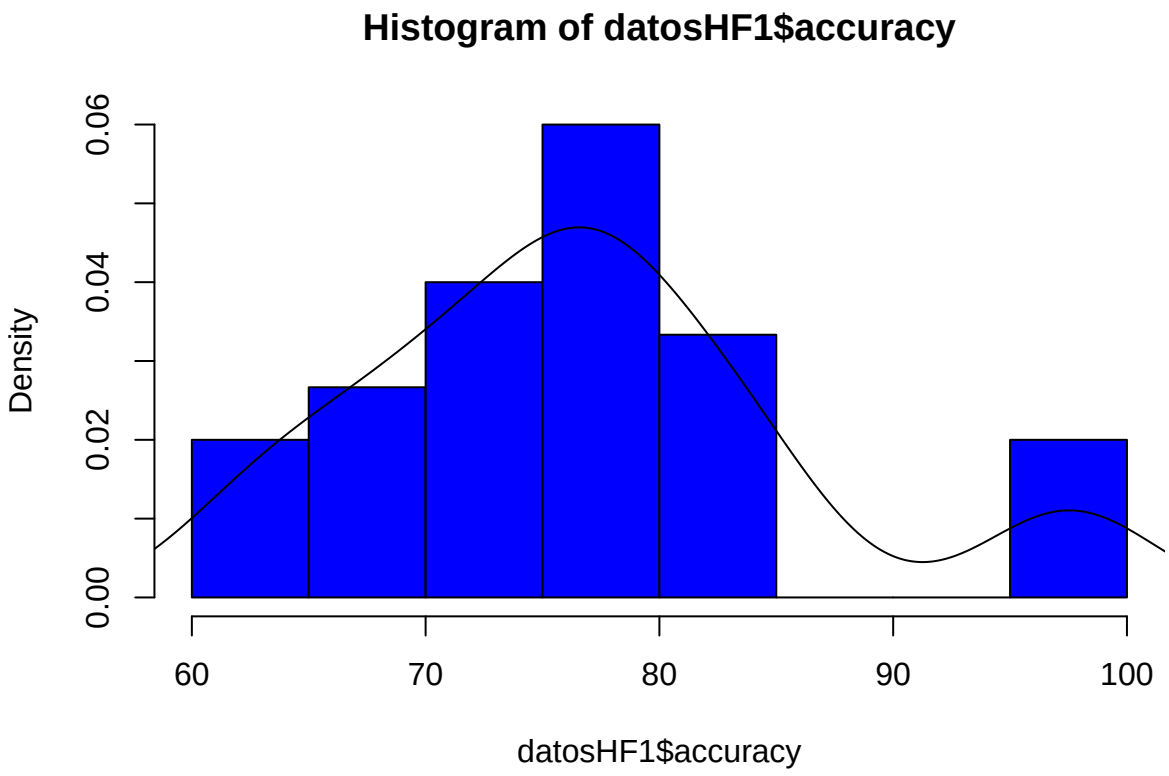
Para averiguar si alguno de los clasificadores es mejor que el otro, tal como hicimos en el ejercicio anterior, procedemos a utilizar tests estadísticos. Para ser que test utilizar, testeamos la normalidad de ambas poblaciones con sus histogramas y tests de Saphiro Wilk.

Para Hoeffding Trees:

```
shapiro.test(datosHF1$accuracy)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  datosHF1$accuracy
## W = 0.93769, p-value = 0.07882
```

```
hist(datosHF1$accuracy, col="blue", prob = TRUE)
lines(density(datosHF1$accuracy))
```

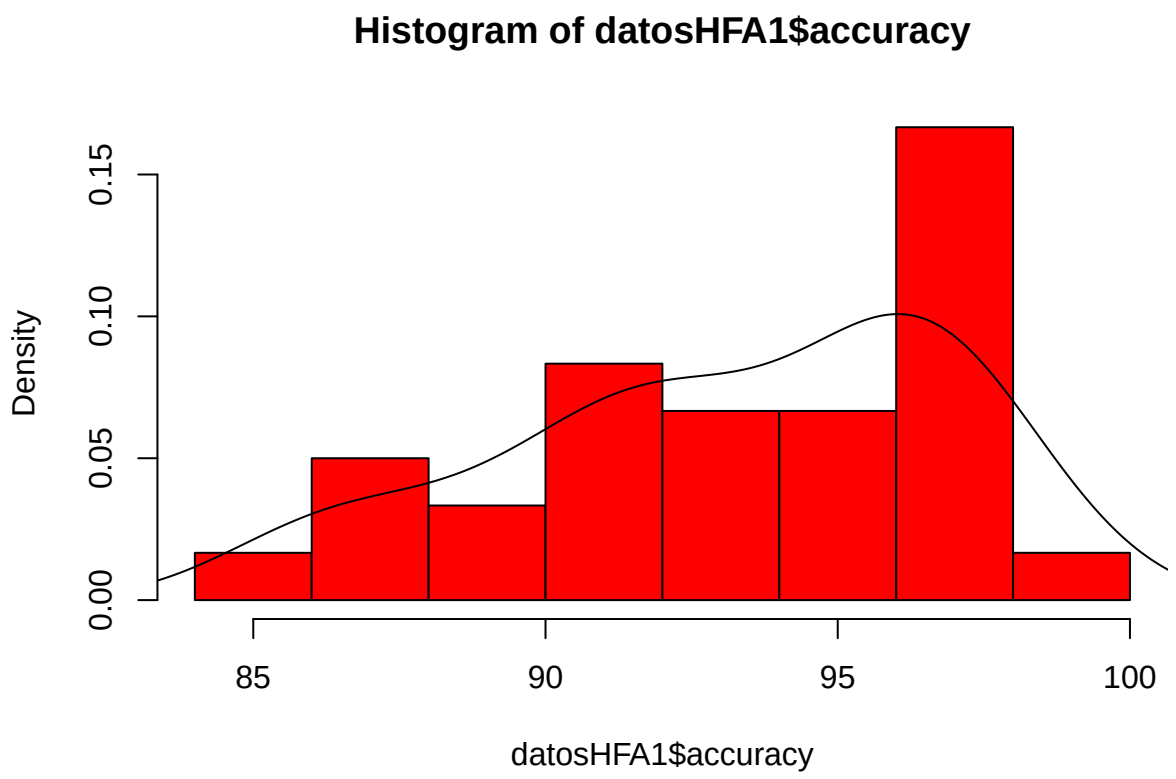


Para Adaptive Hoeffding Trees:

```
shapiro.test(datosHFA1$accuracy)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  datosHFA1$accuracy
## W = 0.93686, p-value = 0.07489
```

```
hist(datosHFA1$accuracy, col="red", prob = TRUE)
lines(density(datosHFA1$accuracy))
```



Como podemos observar, ambos tests siguen una distribución normal, por lo que aplicamos un test paramétrico para comprobar si sus resultados difieren en gran medida:

```
t.test(datosHF1,datosHFA1)
```

```
##
##  Welch Two Sample t-test
##
## data:  datosHF1 and datosHFA1
## t = -8.537, df = 85.792, p-value = 4.382e-13
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -37.55806 -23.36986
## sample estimates:
## mean of x mean of y
```

```
##  55.70495  86.16891
```

Dado el p-value del test obtenido, concretamos con un alto nivel de confianza que los resultados de ambos modelos poseen diferencias significativas, por lo que a la hora de escoger uno de ellos nos quedaríamos con el adaptativo, ya que, tal como vemos en las gráficas de evolución de acierto a lo largo del tiempo, en absoluto le influye el cambio de concepto, por lo que su nivel de acierto se mantiene constante, mientras que en el estacionario disminuye conforme avanza el tiempo.

2.4. Entrenamiento online en datos con concept drift, incluyendo mecanismos para olvidar instancias pasadas.

Repita la experimentación del apartado anterior, cambiando el método de evaluación "Interleaved Test-Then-Train" por el método de evaluación "Prequential", con una ventana deslizante de tamaño 1.000.

¿Qué efecto se nota en ambos clasificadores? ¿A qué es debido? Justifique los cambios relevantes en los resultados de los clasificadores.

2.5. Entrenamiento online en datos con concept drift, incluyendo mecanismos para reinicializar modelos tras la detección de cambios de concepto.

Repita la experimentación del apartado 2.3, cambiando el modelo (learner) a un clasificador simple basado en reemplazar el clasificador actual cuando se detecta un cambio de concepto (SingleClassifierDrift). Como detector de cambio de concepto, usar el método DDM con sus parámetros por defecto. Como modelo a aprender, usar un clasificador HoeffdingTree.

Repita el paso anterior cambiando el clasificador HoeffdingTree por un clasificador HoeffdingTree adaptativo.

Responda a la siguiente pregunta: ¿Qué diferencias se producen entre los métodos de los apartados 2.3, 2.4 y 2.5? Explique similitudes y diferencias entre las diferentes metodologías, y discuta los resultados obtenidos por cada una de ellas en el flujo de datos propuesto.