Minería de Flujo de Datos: Trabajo Final

Juanjo Sierra

25 de abril de 2019

Ejercicio 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 modelo se entrena con el siguiente comando en la consola de MOA.

```
EvaluateModel -m (LearnModel -l trees.HoeffdingTree -s (generators.WaveformGenerator -i 1) \
-m 1000000) -s (generators.WaveformGenerator -i 4) -i 1000000
```

Realizamos una población de tamaño 30 utilizando 30 semillas distintas para comprobar cómo se comporta este algoritmo en general. Almacenamos los datos de accuracy y Kappa para las 30 ejecuciones en un dataframe.

```
accuracy = array(dim = 30)
kappa = array(dim = 30)
for (i in 1:30) {
  archivo =
    paste(c(paste(c("./Datos/Ejercicio1.1/hoeffdingEstacionario",i),
                  collapse = ""),".csv"),collapse="")
  con = file(archivo,open="r")
  datos = readLines(con)
  accuracyFinal =
    as.double(gsub(",", ".", gsub(".* = ", "", datos[2])))
  accuracy[i] = accuracyFinal
  kappaFinal =
    as.double(gsub(",", ".", gsub(".* = ", "", datos[3])))
  kappa[i] = kappaFinal
  close(con)
}

hoeffdingEst = as.data.frame(cbind(accuracy, kappa))
hoeffdingEst
```

```
##    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
```

Ahí se pueden ver los valores que ha obtenido para accuracy y Kappa el modelo estacionario.

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

Para ello utilizamos el mismo comando anterior pero sustituyendo `trees.HoeffdingTree` por `trees.HoeffdingAdaptiveTree`. De nuevo generamos una población de tamaño 30 mediante un script.

```
accuracy = array(dim = 30)
kappa = array(dim = 30)
for (i in 1:30) {
  archivo =
    paste(c(paste(c("./Datos/Ejercicio1.2/hoeffdingAdaptativo",i),
                  collapse = ""),".csv"),collapse="")
  con = file(archivo,open="r")
  datos = readLines(con)
  accuracyFinal =
    as.double(gsub(",", ".", gsub(".* = ", "", datos[2])))
  accuracy[i] = accuracyFinal
  kappaFinal =
    as.double(gsub(",", ".", gsub(".* = ", "", datos[3])))
  kappa[i] = kappaFinal
  close(con)
}

hoeffdingAd = as.data.frame(cbind(accuracy, kappa))
hoeffdingAd
```

```
##    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
## 21   84.553 76.831
## 22   84.432 76.650
## 23   84.686 77.030
## 24   84.485 76.729
## 25   84.589 76.886
## 26   84.372 76.559
## 27   84.476 76.716
## 28   83.978 75.968
## 29   84.379 76.570
## 30   84.222 76.335
```

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

Para responder a esta pregunta de forma adecuada deberíamos evaluar la similitud de los resultados de ambos algoritmos en base a tests estadísticos. Para saber si utilizar tests paramétricos o no paramétricos comprobamos con el test de Shapiro-Wilk si los valores siguen una distribución normal.

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

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

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

```
##
##  Shapiro-Wilk normality test
##
## data:  hoeffdingAd$accuracy
## W = 0.94445, p-value = 0.1199
```

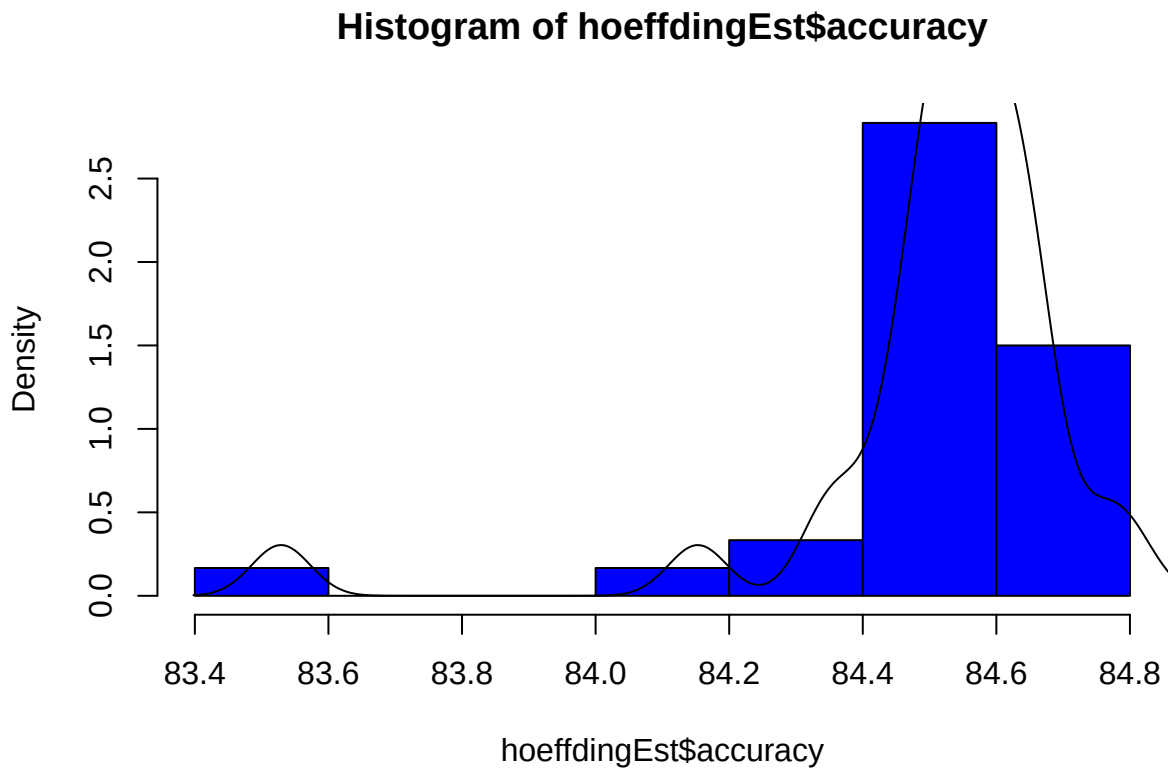
Como el accuracy del modelo estacionario no sigue una distribución normal según el test, para comparar esta variable tendré que usar un test no paramétrico como Wilcoxon.

```r
wilcox.test(hoeffdingEst$accuracy, hoeffdingAd$accuracy, exact = FALSE, alternative = )
```

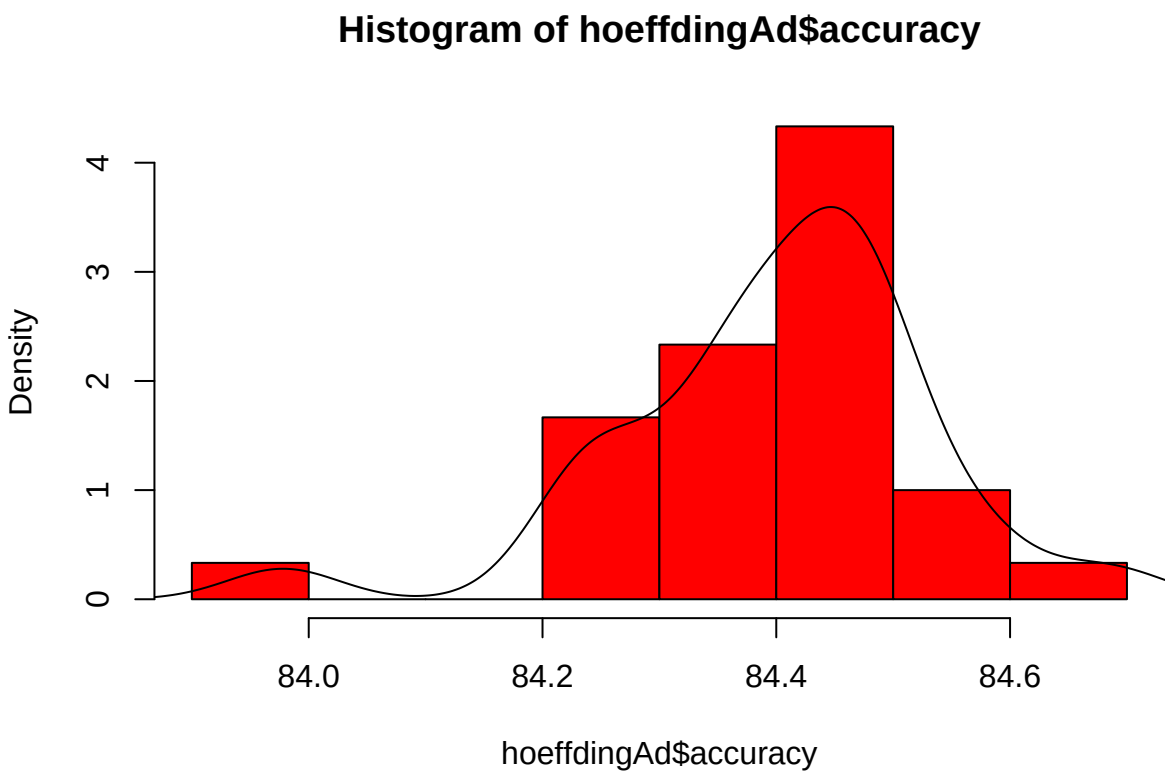
```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  hoeffdingEst$accuracy and hoeffdingAd$accuracy
## W = 720, p-value = 6.763e-05
## alternative hypothesis: true location shift is not equal to 0
```

El test dice que hay diferencias significativas entre ambas poblaciones. Observemos los histogramas de los valores para entender qué está ocurriendo.

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




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



Como se puede observar, el Hoeffding adaptativo (en color rojo) sigue una distribución más normal que el estacionario (en color azul). El estacionario no ha sido clasificado como distribución normal debido a la oblicuidad hacia la derecha que sufre. Además, comprobamos que la gran mayoría de los resultados que obtiene el estacionario están a la derecha de 84.4, por lo que se entiende que salga elegido por Wilcoxon como el mejor algoritmo.

En un flujo que no tiene cambios de concepto es asumible pensar que el algoritmo estacionario se mantenga estable y funcione mejor de forma general a lo largo del tiempo que un modelo adaptativo que puede estar aprendiendo cosas erróneas con los distintos momentos del flujo.

Ejercicio 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 para ejecutar uno de los métodos que piden en el enunciado es el siguiente.

```
EvaluateInterleavedTestThenTrain -l trees.HoeffdingAdaptiveTree -s \
(generators.WaveformGenerator -i 2) -i 1000000 -f 10000
```

Construimos igual que en el caso anterior una población de tamaño 30 utilizando un script, y añadimos los datos a una variable dataframe.

```
accuracy = array(dim = 30)
kappa = array(dim = 30)
for (i in 1:30) {
  archivo =
    paste(c(paste(c("./Datos/Ejercicio2.1/hoeffdingEstacionario",i),collapse = ""),".csv"),collapse="")
  datos = read.csv(archivo)
  accuracyFinal =
    datos$classifications.correct..percent.[length(datos$classifications.correct..percent.)]
  accuracy[i] = accuracyFinal
  kappaFinal =
    datos$Kappa.Statistic..percent.[length(datos$Kappa.Statistic..percent.)]
  kappa[i] = kappaFinal
}

hoeffdingEst2 = as.data.frame(cbind(accuracy, kappa))
hoeffdingEst2
```

```
##    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 para conseguir uno de los métodos que pide el enunciado es el que se muestra a continuación.

```
EvaluateInterleavedTestThenTrain -l trees.HoeffdingAdaptiveTree -s \
(generators.WaveformGenerator -i 2) -i 1000000 -f 10000
```

Construimos la población con el script y obtenemos los datos de accuracy y Kappa.

```
accuracy = array(dim = 30)
kappa = array(dim = 30)
for (i in 1:30) {
  archivo =
    paste(c(paste(c("./Datos/Ejercicio2.2/hoeffdingAdaptativo",i),collapse = ""),".csv"),collapse="")
  datos = read.csv(archivo)
  accuracyFinal =
    datos$classifications.correct..percent.[length(datos$classifications.correct..percent.)]
  accuracy[i] = accuracyFinal
  kappaFinal =
    datos$Kappa.Statistic..percent.[length(datos$Kappa.Statistic..percent.)]
  kappa[i] = kappaFinal
}

hoeffdingAd2 = as.data.frame(cbind(accuracy, kappa))
hoeffdingAd2
```

```
##    accuracy    kappa
## 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.

Vamos de nuevo a ejecutar el test de Shapiro-Wilk para ver qué test utilizar para comparar los resultados.

```
shapiro.test(hoeffdingEst2$accuracy)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  hoeffdingEst2$accuracy
## W = 0.9602, p-value = 0.3135
```

```
shapiro.test(hoeffdingAd2$accuracy)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  hoeffdingAd2$accuracy
## W = 0.98491, p-value = 0.9357
```

Como las dos distribuciones parecen normales según el test de Shapiro-Wilk podemos probar a utilizar el test T de Student para ver si hay diferencias significativas entre ambos clasificadores.

```
t.test(hoeffdingEst2$accuracy, hoeffdingAd2$accuracy)
```

```
##
##  Welch Two Sample t-test
##
## data:  hoeffdingEst2$accuracy and hoeffdingAd2$accuracy
## t = 3.3933, df = 57.972, p-value = 0.001251
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.02424015 0.09397985
## sample estimates:
## mean of x mean of y
##  83.87468  83.81557
```

Los dos clasificadores parecen comportarse igual según el test T de Student. Esto quiere decir que la diferencia entre sus medias no es suficientemente significativa. Tiene sentido dado que cuando es un aprendizaje online los datos que le van llegando al clasificador adaptativo ahora sí facilitan su aprendizaje y puede decidir mejor qué partes de su modelo son mejores. El modelo estacionario se mantiene en los mismos baremos que obtenía en el ejercicio anterior, igualando sus resultados con el modelo adaptativo, cuando antes era el que mejor clasificaba.

Ejercicio 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.

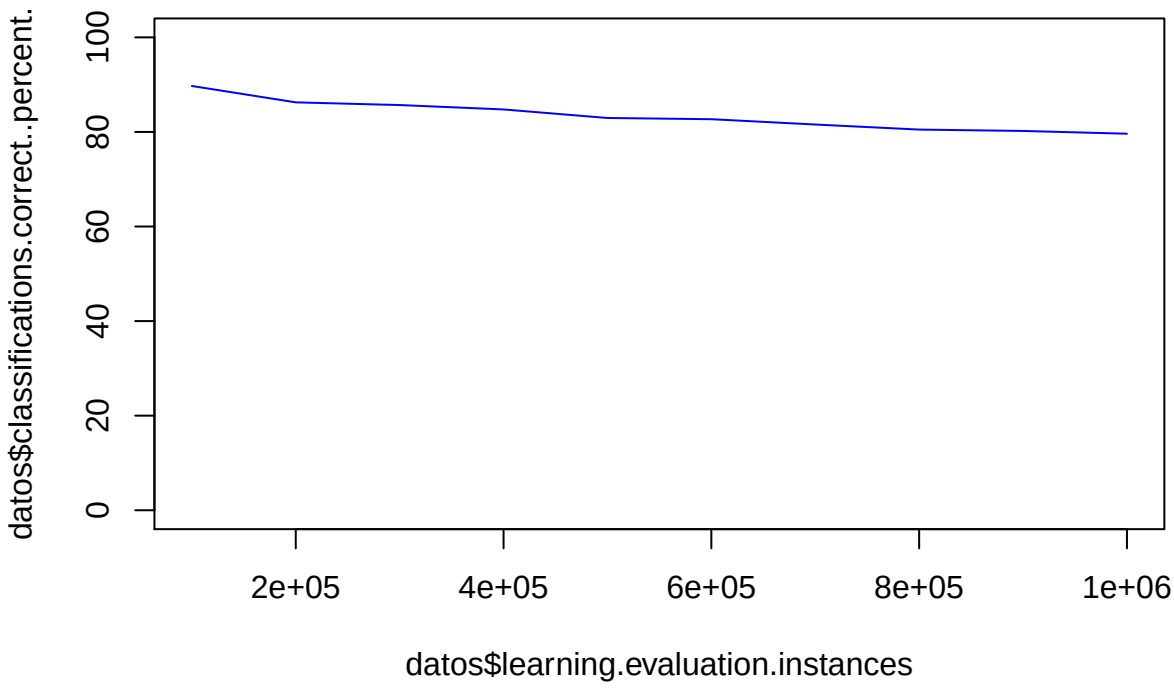
Así se genera el ejemplo que pide el enunciado.

```
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
```

Se genera una población con 30 ejecuciones con distintas semillas. Voy a mostrar solo una gráfica representativa de cómo se comporta este algoritmo para no llenar el PDF de gráficas similares.

```
accuracy = array(dim = 30)
kappa = array(dim = 30)
for (i in 1:30) {
  archivo =
    paste(c(paste(c("./Datos/Ejercicio3.1/hoeffdingEstacionario",i),collapse = ""),".csv"),collapse="")
  datos = read.csv(archivo)
  accuracyFinal =
    datos$classifications.correct..percent.[length(datos$classifications.correct..percent.)]
  accuracy[i] = accuracyFinal
  kappaFinal =
    datos$Kappa.Statistic..percent.[length(datos$Kappa.Statistic..percent.)]
  kappa[i] = kappaFinal

  if (i == 1) {
    plot(datos$learning.evaluation.instances,
     datos$classifications.correct..percent.,
     "l", ylim = c(0,100), col = "blue")
  }
}
```




```
hoeffdingEst3 = as.data.frame(cbind(accuracy, kappa))
hoeffdingEst3
```

```
##    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.

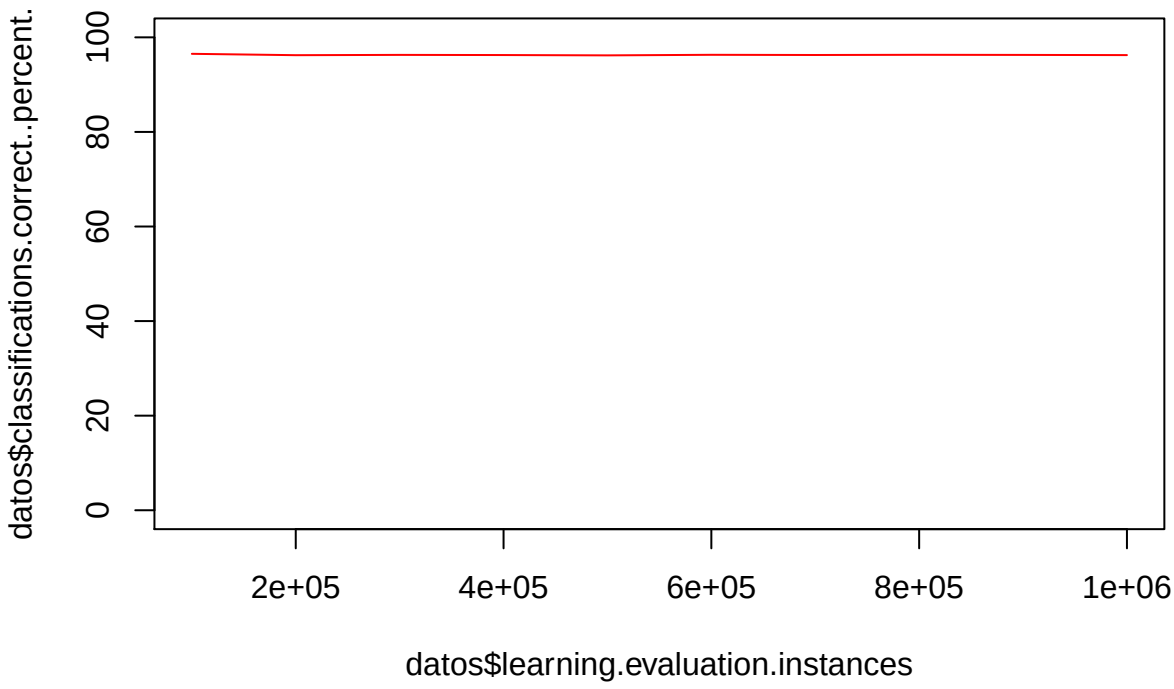
Una ejecución del algoritmo se lanzaría con el siguiente comando.

```
EvaluateInterleavedTestThenTrain -l trees.HoeffdingAdaptiveTree -s \
(generators.RandomRBFGeneratorDrift -r 1 -i 1 -c 2 -a 7 -n 3 -s 0.001 -k 3) -i 1000000 -f 100000
```

De nuevo generamos otra población con 30 de estas ejecuciones.

```
accuracy = array(dim = 30)
kappa = array(dim = 30)
for (i in 1:30) {
  archivo =
    paste(c(paste(c("./Datos/Ejercicio3.2/hoeffdingAdaptativo",i),collapse = ""),".csv"),collapse="")
  datos = read.csv(archivo)
  accuracyFinal =
    datos$classifications.correct..percent.[length(datos$classifications.correct..percent.)]
  accuracy[i] = accuracyFinal
  kappaFinal =
    datos$Kappa.Statistic..percent.[length(datos$Kappa.Statistic..percent.)]
  kappa[i] = kappaFinal

  if (i == 1) {
    plot(datos$learning.evaluation.instances,
     datos$classifications.correct..percent.,
     "l", ylim = c(0,100), col = "red")
  }
}
```




```
hoeffdingAd3 = as.data.frame(cbind(accuracy, kappa))
hoeffdingAd3
```

```
##    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.

Vamos a probar con el test de Shapiro-Wilk si las distribuciones de los valores de accuracy son normales.

```
shapiro.test(hoeffdingEst3$accuracy)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  hoeffdingEst3$accuracy
## W = 0.93769, p-value = 0.07882
```

```
shapiro.test(hoeffdingAd3$accuracy)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  hoeffdingAd3$accuracy
## W = 0.93686, p-value = 0.07489
```

Ambas siguen una distribución normal en base al test de Shapiro-Wilk, por lo que podemos ejecutar el test T de Student para ver si la diferencia entre sus medias es significativa. En base a lo observado en las gráficas parece obvio que sí, pero lo hacemos por asegurarnos.

```
t.test(hoeffdingEst3$accuracy, hoeffdingAd3$accuracy)
```

```
##
##  Welch Two Sample t-test
##
## data:  hoeffdingEst3$accuracy and hoeffdingAd3$accuracy
## t = -8.8619, df = 37.802, p-value = 9.186e-11
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -20.18217 -12.67505
## sample estimates:
## mean of x mean of y
##  76.76688  93.19549
```

El p-value es muy pequeño dejando claro que existen diferencias significativas, como habíamos visto antes claramente a favor del algoritmo adaptativo. Al ocurrir cambios de concepto en el flujo de datos, el algoritmo adaptativo es capaz de mejorar su modelo desechando aquellas partes que no clasifiquen bien. El estacionario por su parte no aprende tan rápido y por tanto va decayendo su acierto a medida que se suceden los cambios de concepto.