Práctica Mineria de Flujo de Datos.

Néstor Rodríguez Vico. DNI: 75573052C - nrv23@correo.ugr.es

25 de abril de 2019

1. Parte teórica.

1.1. Preguntas tipo test.

Responda a las 12 preguntas tipo test del anexo. Cada respuesta incorrecta restará un tercio (si hay cuatro opciones) o una unidad (si tiene dos opciones) del valor de cada pregunta (un doceavo).

1. *El aprendizaje incremental es últil cuando...* d) se quiere ganar eficiencia.
2. *La minería de flujo de datos se considera cuando...* a) el problema genera datos continuamente.
3. *La cota de Hoeffding sirve para saber...* b) cuando hay suficientes datos para una estimación fiable.
4. *¿Que características de clusters mantiene el algoritmo BIRCH?* d) Suma linea, suma cuadrática y número de objetos.
5. *¿El algoritmo Stram maneja concept drift?* b) No.
6. *¿Qué es concept drift?* d) Cambios en la dinámica del problema.
7. *¿Cómo gestion CVFDT el concept drift?* d) Mantiene árboles alternativos.
8. *¿Por qué es útil el ensemble learning en concept drift?* c) Porque mantienen muchos modelos simultáneamente.
9. *¿Cuál es más eficiente entre DDM y ADWIN?*: c) ADAWIN es más eficiente.
10. *¿Por qué es controvertida la clasificación en flujo de datos?* a) Porque se requiere al oráculo por siempre.
11. *¿Cómo gestiona ClueStream el concept drift?*
12. *¿Por qué es complejo generar reglas de asociación en flujo de datos?*

Anexo: Teoría – Minería de Flujos de Datos. Curso 2018-2019.

Apellidos y nombre: Néstor Rodríguez Vico

1. El aprendizaje incremental es útil cuando...
   - ☐ se trabaja con datos no estacionarios
   - ☐ aumenta el tiempo de respuesta
   - ☐ se aprende sobre otro algoritmo
   - ☑ se quiere ganar eficiencia
2. La minería de flujo de datos se considera cuando...
   - ☑ el problema genera datos continuamente
   - ☐ los datos son estacionarios
   - ☐ se quiere mejorar principalmente la eficiencia
   - ☐ se quiere mejorar un modelo previamente aprendido
3. La cota de Hoeffding sirve para saber...
   - ☐ si la información recibida es fiable
   - ☑ cuándo hay suficientes datos para una estimación fiable
   - ☐ qué fiabilidad tienen los datos
   - ☐ qué precisión se puede alcanzar
4. ¿Que características de clusters mantiene el algoritmo BIRCH?
   - ☐ Suma lineal y número de objetos
   - ☐ Tiempo, suma lineal y tamaño
   - ☐ Tiempo, suma lineal y suma cuadrática
   - ☑ Suma lineal, suma cuadrática y número de objetos
5. ¿El algoritmo Stream maneja concept drift?
   - ☐ Sí
   - ☑ No
6. ¿Qué es concept drift?
   - ☐ Cambios en el número de clases
   - ☑ Variaciones en los parámetros del algoritmo
   - ☐ Variaciones en los tipos de variables
   - ☐ Cambios en la dinámica del problema
7. ¿Cómo gestiona CVFDT el concept drift?
   - ☐ Sustituye el atributo del nodo por el segundo mejor
   - ☐ Crea un cluster alternativo
   - ☐ Detecta el cambio de concepto
   - ☑ Mantiene árboles alternativos
8. ¿Por qué es útil el ensemble learning en concept drift?
   - ☐ Porque es robusto frente a cambios de párametros
   - ☐ Porque aprovecha la diversidad que se genera en los cambios
   - ☑ Porque matienen muchos modelos simultáneamente
   - ☐ Porque hace un muestreo selectivo de los datos
9. ¿Cuál es más eficiente entre DDM y ADWIN?
   - ☐ DDM es más eficiente
   - ☐ Los dos son muy ineficientes
   - ☑ ADWIN es más eficiente
   - ☐ Los dos son similares
10. ¿Por qué es controvertida la clasificación en flujo de datos?
    - ☑ Porque se requiere al oráculo por siempre
    - ☐ Porque el etiquetado es costoso
    - ☐ Porque la clase puede cambiar
    - ☐ Porque el oráculo puede cambiar su valoración
11. ¿Cómo gestiona ClueStream el concept drift?
    - ☐ Expande los microclusters cuando llega un cambio de concepto
    - ☐ Reduce las fronteras inter-microclusters si es necesario
    - ☐ Mantiene información sobre el tiempo
    - ☐ Reduce el tamaño de los microclusters cada cierto tiempo
12. ¿Por qué es complejo generar reglas de asociación en flujo de datos?
    - ☐ Porque los frequent itemsets no pueden ser dinámicos
    - ☐ Porque las reglas se evalúan sobre el histórico de datos
    - ☐ Porque para calcular la confianza se requieren muchos datos
    - ☐ Porque no maneja bien variables continuas

1.2. Pregunta 1.

Explique el problema de clasificación, los clasificadores utilizados en los experimentos de la sección 2, y en qué consisten los diferentes modos de evaluación/validación en flujos de datos. Desarrolle con suficiente detalle este apartado.

En el problema de clasificación en flujo de datos no se disponen de los datos previamente. En estos problemas, los datos van llegando conforme pasa el tiempo. Debido a la cantidad de datos producidos, lo que se suele hacer es procesar el dato y, seguídamente, descartarlo. Al igual que en propblemas de clasificación estándar, los algoritmos usado en clasificación en flujo de datos deben ser capaces de maximizar la precisión. De forma adicional, estos algoritmos deben ser capaces de detectar y reaccionar ante los cambios de concepto que se pueden producir.

Los algoritmos usados en la parte de prácticas son:

- *HoeffdingTree:* Se trata de un algoritmo que permite construir un árbol incremental a partir de un flujo de datos, asumiendo que no se producen cambios de convepto en el flujo de datos. La principal característica de este algoritmo es que hace uso de un valor denominado *cota de Hoeffding*, la cual representa el número de instancias necesarias para calcular los datos estadísticos de un flujo de datos de forma precisa. Esta cota es usada para decidir cuando vamos a dividir un nodo.
- *HoeffdingAdaptiveTree:* La diferencia de *HoeffdingAdaptiveTree* con *HoeffdingTree* es que usa *ADWIN* para analizar el rendimiento de las ramas del árbol, lo cual permite reemplazarlas por nuevas si su rendimiento decrece.

Los modelos de evaluación empleados son:

- *EvaluateModel:* Se produce una evluación estática del modelo.
- *InterleavedTest-Then-Train:* cuando llega un dato nuevo se usa para evaluar el modelo y, a continuación, se le da al modelo para que lo aprenda.
- *Prequential:* técnica que permite usar una ventana deslizante para olvidar instancias antiguas.

1.3. Pregunta 2.

Explique en qué consiste el problema de concept drift y qué técnicas conoce para resolverlo en clasificación. Desarrolle con suficiente detalle este apartado.

Una de las principales dificultades que se aborda en flujo de datos es el desvío de concepto o concept drift, esto es, un cambio en la dinámica de los datos. Este cambio puede ser de tres tipos:

- Repentino: se produce un cambio de forma rápida.
- Incremental: el cambio en la dinámica del flujo de datos se produce lentamente.
- Gradual: el cambio de produce de forma lenta y, por lo tanto, hay un periodo de tiempo en el que ambas dinámicas (la antigua y la nueva) coexisten en el flujo de datos.
- Recurrente: el cambio en la dinámica se produce varias veces a lo largo del flujo de datos.

Las técnicas usadas para abordar el *Concept Drift* pueden dividirse en los siguientes enfoques:

- *Aprendizaje online:* Aquí encontramos los clasificadores que se actualizan con cada nueva instancia. No todos los clasificadores pueden actuar como aprendizaje online, tienen que cumplir una serie de requisitos básicos. Primero: cada objeto debe procesarse solo una vez durante el entrenamiento. Segundo: el sistema debe consumir una memoria y procesamiento limitados con independencia de la cantidad de datos procesada. Tercero: el aprendizaje se puede pausar en cualquier momento y su precisión no debería ser peor que la de un clasificador entrenado offline sobre los datos vistos hasta el momento.

- *Soluciones basadas en instancias (ventana):* Aquí encontramos los algoritmos que incorporan mecanismos para olvidar datos antiguos. Se basan en asumir que los datos llegados recientemente son más relevantes porque contienen características del contexto actual y su relevancia disminuye con el paso del tiempo. Por tanto, ajustar el rango de datos a aquellos que han llegado más recientemente puede ayudar a formar un conjunto de datos que representa al contexto actual.

- *Aprendizaje de múltiples modelos (ensemble):* Entre los enfoques ensemble para flujo de datos más populares, podemos destacar: *Streaming Ensemble Algorithm (SEA)* y *Accuracy Weighted Ensemble (AWE)*. Ambos algoritmos mantienen un número fijo de clasificadores. Los datos que van llegando se recogen en fragmentos que se usan para entrenar nuevos clasificadores. Los clasificadores se evalúan según su precisión y el peor es reemplazado por uno nuevo si este último es más preciso. *SEA* usa voto por mayoría, mientras que *AWE* usa un voto ponderado.

- *Algoritmos de detección del desvío:* En lugar de dotar de adaptabilidad al proceso de aprendizaje, otra alternativa es detectar cuándo se produce el *Concept Drift* y actuar para paliar sus efectos (por ejemplo, reseteando total o parcialmente el conocimiento adquirido por el algoritmo). El objetivo de cualquier detector de *Concept Drift* es reconocer los cambios en flujos de datos no estacionarios de manera precisa y oportuna. El algoritmo explicado en clase de este enfoque es *DDM (Drift Detection Method)*. *DDM* detecta el desvío de concepto en función de la precisión del clasificador o mediante un análisis de distribución de la clase. Para ello, almacena datos estadísticos de dos ventanas de tiempo. La primera de ellas contiene información de los errores obtenidas durante todo el proceso. La segunda contiene los datos desde el inicio hasta que el error empieza a incrementar considerablemente, es decir, cuando se empieza a producir el cambio de convepto. *DDM* se comporta bien si el cambio de concepto es bruscos, pero no se comporate bien cuando el cambio es lento. Otro algoritmo es *ADAWIN (ADaptive sliding WINdow)*. Dada la ventana W, si existen dos subventanas W_0 y W_1 suficientemente grandes y con medias suficientemente distintas, podemos concluir que los valores esperados son diferentes y se puede eliminar la parte antigua de W. Finalmente, otro algoritmo que podemose usar es *HSP*. *HSP* basa la deteccioón exclusivamente en las muestras, sin necesidad de monitorizar al algoritmo, con las siguientes ventajas:

  - Complejidad de tiempo y espacio O(nm) -siendo n el número de características y m el número de clases- para procesar cada muestra: es extremadamente eficiente.
  - Implementación e integración ágil, lo que facilita la creación de prototipos y el análisis exploratorio de nuevos problemas de flujo de datos. Su diseño es simple, por lo que es fácil de entender su comportamiento y depurar el proceso.
  - La verdadera detección de *Concept Drift*, incluso cuando los algoritmos eficaces no se degradan significativamente, puede mejorar el proceso de predicción y obtener una mayor comprensión sobre la naturaleza del problema.
  - En último término, *HSP* es compatible con el uso de detectores de *Concept Drift* tradicionales basados en el rendimiento del algoritmo, por lo que se pueden complementar para corroborar detecciones o adaptarse dinámicamente a las características del problema.

2. Parte práctica.

Para ejecutar los experimentos de esta sección se ha usado el mismo script que se ha usado en el trabajo guiado:

```
1 for i in $(seq 5)
2 do
3   java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask "..." > "$i.txt"
4 done
```

Sólo debemos cambiar los tres puntos de la línea 3 por la orden que queramos ejecutar. Una vez tenemos los resultados de las 5 ejecuciones, podemos aplicar los tests estadísticos. Para ello, he usado el siguiente código en R:

```
1  read.data <- function(folder, col) {
2    # Listamos los ficheros
3    ficheros <- list.files(path = folder, full.names=TRUE)
4    # Leemos los conjuntos de datos
5    data <- lapply(ficheros, read.csv)
6    # Tal y como dice la transparencia 19, nos quedamos el
7    # ultimo valor de la columna deseada.
8    sapply(data, function(x) x[nrow(x), col])
9  }
10
11 accuracy.standard <- read.data(folder = "...", col = 5)
12 kappa.standard <- read.data(folder = "...", col = 6)
13 accuracy.adaptative <- read.data(folder = "...", col = 5)
14 kappa.adaptative <- read.data(folder = "...", col = 6)
```

Este código simplemente lee todos los ficheros almacenados en un directorio (que son los obtenido con el script anterior) y selecciona el último valor de la columna indicada, para así obtener la tasa de aciertos y el valor *Kappa* para cada ejecución con una semilla distinta. Esos valores los almacena en un vector que será usado posteriormente.

2.1. Entrenamiento offline (estacionario) y evaluación posterior.

2.1.1. Ejercicio 1.

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.

Para resolver este problema se ha usado la siguiente orden:

```
1 EvaluateModel -m
2   (LearnModel
3     -l trees.HoeffdingTree
4     -s (generators.WaveformGenerator -i 2)
5     -m 1000000
6   ) -s (generators.WaveformGenerator -i 4) -i 1000000
```

En la línea tres difinimos el modelo que vamos a aprender. En la línea cuatro indicamos como se van a generar los datos, en este caso usando el generador *WaveFormGenerator* con semilla *2*. En la línea cinco indicamos el número de instancias a generar. Finalmente, en la línea seis evaluamos el modelo con 100000 instancias generador por el generado *WaveFormGenerator* con semilla *4*. Para automatizar la prueba de varias semillas voy a usar el script comentado al principio de la parte práctica y modificar el comando de *MOA* para que la semilla de entrenamiento sea distinta. Los resultados obtenidos son los siguientes:

Semilla	% Acierto	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

2.1.2. Ejercicio 2.

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

Para resolver este problema se ha usado la siguiente orden:

```
1 EvaluateModel -m
2   (LearnModel
3     -l trees.HoeffdingAdaptiveTree
4     -s (generators.WaveformGenerator -i 2)
5     -m 1000000
6   ) -s (generators.WaveformGenerator -i 4) -i 1000000
```

El código es el mismo, sólo tenemos que cambiar el modelo que vamos a aprender en la línea 3. Los resultados obtenidos son los siguientes:

Semilla	% Acierto	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

2.1.3. Ejercicio 3.

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 pergunta podríamos usar un test de *Student*, al igual que hemos hecho en el trabajo guiado. Para ello, primero debemos comprobar la normalidad de los datos. Este proceso se va a hacer con R:

```
1  accuracy.estandar <- c(84.509, 84.512, 84.590, 84.666, 84.481)
2  kappa.estandar <- c(76.765, 76.770, 76.887, 77.001, 76.723)
3
4  accuracy.adaptative <- c(84.521, 84.474, 84.416, 84.465, 84.262)
5  kappa.adaptative <- c(76.783, 76.712, 76.625, 76.699, 76.395)
6
7  shapiro.test(accuracy.standar)$p.value # 0.3359973
8  shapiro.test(kappa.standar)$p.value # 0.340034
9  shapiro.test(accuracy.adaptative)$p.value # 0.2657517
10 shapiro.test(kappa.adaptative)$p.value # 0.2691799
```

Cómo podemos ver, los datos son normales, así que podemos aplicar un test de *Student*:

```
1 t.test(accuracy.standar, accuracy.adaptative)$p.value # 0.06014706
2 t.test(kappa.standar, kappa.adaptative)$p.value # 0.05956797
```

Los test no pasan, ya que el p-valor obtenido es mayor que 0.05, lo cual nos indica que no hay diferencia entre los algoritmos.

2.2. Entrenamiento online.

2.2.1. Ejercicio 1.

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.

Para resolver este problema se ha usado la siguiente orden:

```
1 EvaluateInterleavedTestThenTrain -l trees.HoeffdingTree
2   -s (generators.WaveformGenerator -i 2)
3   -i 1000000
4   -f 10000
```

La idea de este comando es bastante similar al usado en el ejercicio anterior. Primero definimos el método a usar y el modelo en la línea uno, a continuación generamos los datos usando un generado *WaveformGenerator* como podemos ver en la línea dos. Finalmente, definimos los datos usado por el experimento en la línea tres y cuatro. Los resultados obtenidos son los siguientes:

Semilla	% Acierto	Kappa
1	83.8903	75.836
2	83.7851	75.677
3	83.8876	75.829
4	84.0451	76.069
5	83.8402	75.759

2.2.2. Ejercicio 2.

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

Para resolver este problema se ha usado la siguiente orden:

```
1 EvaluateInterleavedTestThenTrain -l trees.HoeffdingAdaptativeTree
2   -s (generators.WaveformGenerator -i 2)
3   -i 1000000
4   -f 10000
```

El código es el mismo, sólo tenemos que cambiar el modelo que vamos a aprender en la línea uno. Los resultados obtenidos son los siguientes:

Semilla	Kappa	% Acierto
1	75.707	83.8042
2	75.596	83.7313
3	75.679	83.7875
4	75.696	83.7961
5	75.571	83.7144

2.2.3. Ejercicio 3.

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

Para responder a esta pergunta podríamos usar un test de *Student*, al igual que hemos hecho en el trabajo guiado. Para ello, primero debemos comprobar la normalidad de los datos. Este proceso se va a hacer con R:

```
shapiro.test(accuracy.standard)$p.value # 0.4709823
shapiro.test(kappa.standard)$p.value # 0.4709826
shapiro.test(accuracy.adaptative)$p.value # 0.1879361
shapiro.test(kappa.adaptative)$p.value # 0.2112991

tseries::jarque.bera.test(accuracy.standard)$p.value # 0.7643144
tseries::jarque.bera.test(kappa.standard)$p.value # 0.759523
tseries::jarque.bera.test(accuracy.adaptative)$p.value # 0.6988181
tseries::jarque.bera.test(kappa.adaptative)$p.value # 0.7029419
```

Cómo podemos ver, los datos son normales, así que podemos aplicar un test de *Student*:

```
t.test(accuracy.standard, accuracy.adaptative)$p.value # 0.04417184
t.test(kappa.standard, kappa.adaptative)$p.value # 0.04499021
```

Los test si pasan, ya que el p-valor obtenido es menor que 0.05, lo cual nos indica que hay diferencia entre los algoritmos. Para decidir cual es el mejor, simplemente calculamos la media de acierto:

```
mean(accuracy.standard) # 83.88966
mean(accuracy.adaptative) # 83.7667
```

En este caso, usar *HoeffdingTree* proporciona unos mejores resultados que *HoeffdingAdaptiveTree*.

2.3. Entrenamiento online en datos con concept drift.

2.3.1. Ejercicio 1.

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.

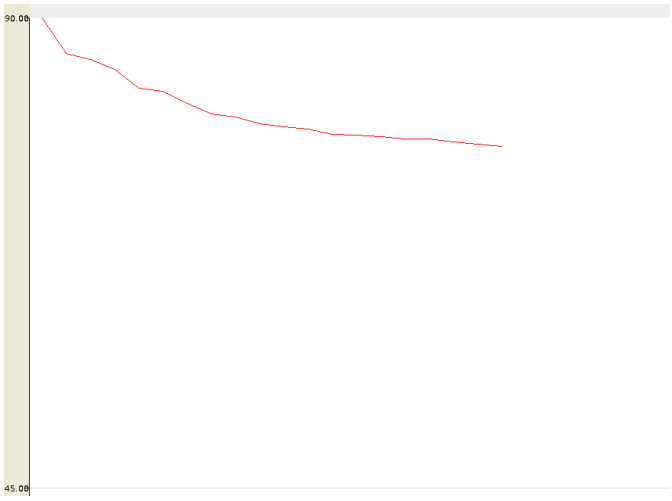
Para resolver este problema se ha usado la siguiente orden:

```
EvaluateInterleavedTestThenTrain -l trees.HoeffdingTree
  -s (generators.RandomRBFGeneratorDrift
    -r 1
    -i 1
    -a 7
    -n 3
    -k 3
    -s 0.001
  ) -i 2000000
  -f 100000
```

En la lína uno indicamos el modelo a aprender. En la línea se indica el generador de flujos a usar, *RandomRBFGeneratorDrift*. De las líneas tres a ocho indicamos los parámetros de dicho generador de datos. Línea tres: semilla para la generación del modelo. Línea cuatro: semilla para la generación de instancias. Línea cinco: número de atributos a generar. Línea seis: número de centroides a generar. Línea siete: número de centroides con drift. Línea ocho: velocidad de cambio. Finalmente, en la línea nueve indicamos que vamos a generar 2000000 instancias y en la línea diez que la frecuencia de muestreo va a ser 100000. Las semillas se han cambiado tanto para la generación del modelo como para la generación de las instancias. Los resultados obtenidos son los siguientes:

Semilla	% Acierto	Kappa
1	77.454	54.775
2	83.956	29.391
3	73.473	43.058
4	73.046	44.985
5	72.061	43.152

La evolución de la curva de aciertos es la siguiente:



Como podemos ver, la curva de aciertos es decreciente, es decir, conforme va pasando el tiempo, el modelo es peor.

2.3.2. Ejercicio 2.

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

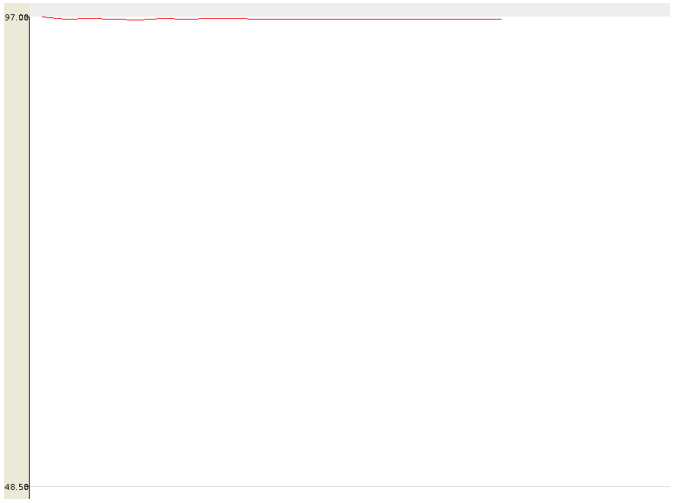
Para resolver este problema se ha usado la siguiente orden:

```
1  EvaluateInterleavedTestThenTrain -l trees.HoeffdingAdaptiveTree
2    -s (generators.RandomRBFGeneratorDrift
3      -r 1
4      -i 1
5      -a 7
6      -n 3
7      -k 3
8      -s 0.001
9    ) -i 2000000
10   -f 100000
```

El código es el mismo, sólo tenemos que cambiar el modelo que vamos a aprender en la línea uno. Las semillas se han cambiado tanto para la generación del modelo como para la generación de las instancias. Los resultados obtenidos son los siguientes:

Semilla	% Acierto	Kappa
1	96.249	92.491
2	92.647	73.926
3	96.309	92.194
4	95.983	91.771
5	90.996	81.657

La evolución de la curva de aciertos es la siguiente:



Como podemos ver, la curva de aciertos en este caso es más estables, es decir, conforme va pasando el tiempo, el modelo mantiene su calidad, lo cual nos hace pensar que *HoeffdingAdaptiveTree* es mejor que *HoeffdingTree*.

2.3.3. Ejercicio 3.

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

Para responder a esta pergunta podríamos usar un test de *Student*, al igual que hemos hecho en el trabajo guiado. Para ello, primero debemos comprobar la normalidad de los datos. Este proceso se va a hacer con R:

```
shapiro.test(accuracy.standard)$p.value # 0.1587713
shapiro.test(kappa.standard)$p.value # 0.5099431
shapiro.test(accuracy.adaptative)$p.value # 0.08033221
shapiro.test(kappa.adaptative)$p.value # 0.07773439

tseries::jarque.bera.test(accuracy.standard)$p.value # 0.6533606
tseries::jarque.bera.test(kappa.standard)$p.value # 0.9239095
tseries::jarque.bera.test(accuracy.adaptative)$p.value # 0.6975661
tseries::jarque.bera.test(kappa.adaptative)$p.value # 0.6996095
```

Cómo podemos ver, los datos son normales, así que podemos aplicar un test de *Student*:

```
t.test(accuracy.standard, accuracy.adaptative)$p.value # 0.0003108372
t.test(kappa.standard, kappa.adaptative)$p.value # 5.072058e-05
```

Los test si pasan, ya que el p-valor obtenido es menor que 0.05, lo cual nos indica que hay diferencia entre los algoritmos. Para decidir cual es el mejor, simplemente calculamos la media de acierto:

```
mean(accuracy.standard) # 75.99857
mean(accuracy.adaptative) # 94.43709
```

En este caso, usar *HoeffdingAdaptiveTree* proporciona unos resultados bastante superiores a *HoeffdingTree*.

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

2.4.1. Ejercicio 1.

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.

Para resolver este problema se ha usado la siguiente orden:

```
1  EvaluatePrequential -l trees.HoeffdingTree
2    -s (generators.RandomRBFGeneratorDrift
3      -r 1
4      -i 1
5      -a 7
6      -n 3
7      -k 3
8      -s 0.001
9    ) -i 2000000
10 -f 100000
```

El código es el mismo que en el experimento anterior, sólo tenemos que cambiar el método de evaluación en la línea uno. Las semillas se han cambiado tanto para la generación del modelo como para la generación de las instancias. Los resultados obtenidos son los siguientes:

Semilla	% Acierto	Kappa
1	76.6	52.558
2	86.5	39.825
3	52.9	-0.106
4	65.3	29.603
5	63.3	25.284

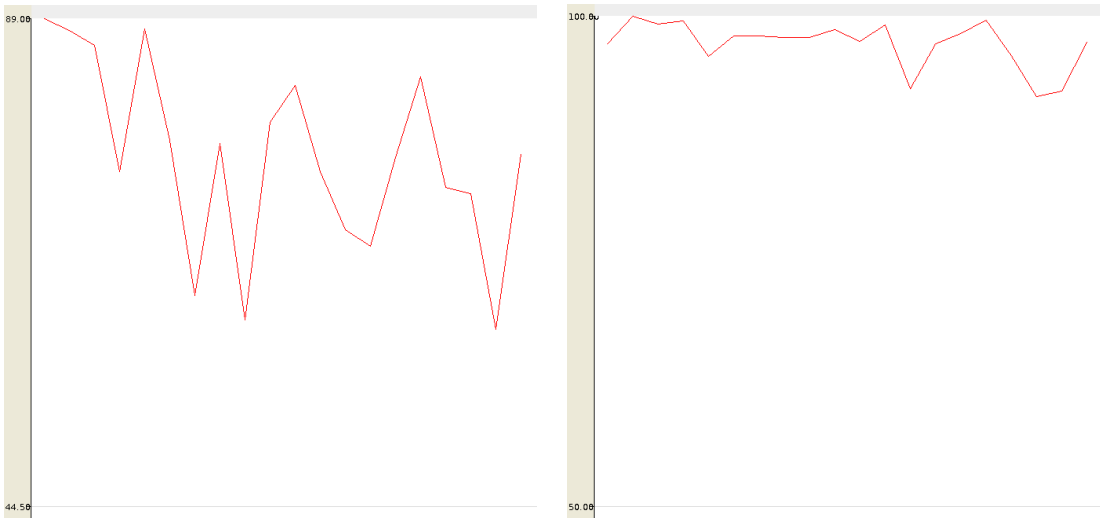
A continuación se ha pasado a usar un modelo *HoeffdingAdaptiveTree*. Para ello se ha usado la siguiente orden:

```
1  EvaluatePrequential -l trees.HoeffdingAdaptiveTree
2    -s (generators.RandomRBFGeneratorDrift
3      -r 1
4      -i 1
5      -a 7
6      -n 3
7      -k 3
8      -s 0.001
9    ) -i 2000000
10 -f 100000
```

El código es el mismo, sólo tenemos que cambiar el método de evaluación en la línea uno. Las semillas se han cambiado tanto para la generación del modelo como para la generación de las instancias. Los resultados obtenidos son los siguientes:

Semilla	% Acierto	Kappa
1	96.6	93.190
2	95.3	82.198
3	98.0	95.755
4	96.3	92.468
5	91.9	83.400

Las curvas de acierto son las siguientes:



A la izquierda podemos ver la obtenida usando un modelo *HoeffdingTree* y a la derecha obtenida con un modelo *HoeffdingAdaptiveTree*. Vamos a comparar dichos modelos con la misma metodología usada en los ejercicio anteriores:

```
shapiro.test(accuracy.standard)$p.value # 0.9041046
shapiro.test(kappa.standard)$p.value # 0.8608652
shapiro.test(accuracy.adaptative)$p.value # 0.4519026
shapiro.test(kappa.adaptative)$p.value # 0.1972134

tseries::jarque.bera.test(accuracy.standard)$p.value # 0.8574582
tseries::jarque.bera.test(kappa.standard)$p.value # 0.8639967
tseries::jarque.bera.test(accuracy.adaptative)$p.value # 0.7225708
tseries::jarque.bera.test(kappa.adaptative)$p.value # 0.7078028
```

Cómo podemos ver, los datos son normales, así que podemos aplicar un test de *Student*:

```
t.test(accuracy.standard, accuracy.adaptative)$p.value # 0.00907
t.test(kappa.standard, kappa.adaptative)$p.value # 0.001481889
```

Los test si pasan, ya que el p-valor obtenido es menor que 0.05, lo cual nos indica que hay diferencia entre los algoritmos. Para decidir cual es el mejor, simplemente calculamos la media de acierto:

```
mean(accuracy.standard) # 68.92
mean(accuracy.adaptative) # 95.62
```

En este caso, usar *HoeffdingAdaptiveTree* proporciona unos resultados bastante superiores a *HoeffdingTree*, cosa que podemos ver visualmente en las tablas de resultados mostradas en las curvas de aciertos.

2.4.2. Ejercicio 2.

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

Como podemos ver, lo que se produce es un modelo más fluctuante. Si comparamos las curvas de acierto, podemos ver que ambos modelos fluctuan bastante más. Esto tiene setndido ya que hemos introducido mecanismos para olvidar instancias y por eso se producen las bajadas en los porcentajes de acierto.

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

2.5.1. Ejercicio 1.

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.

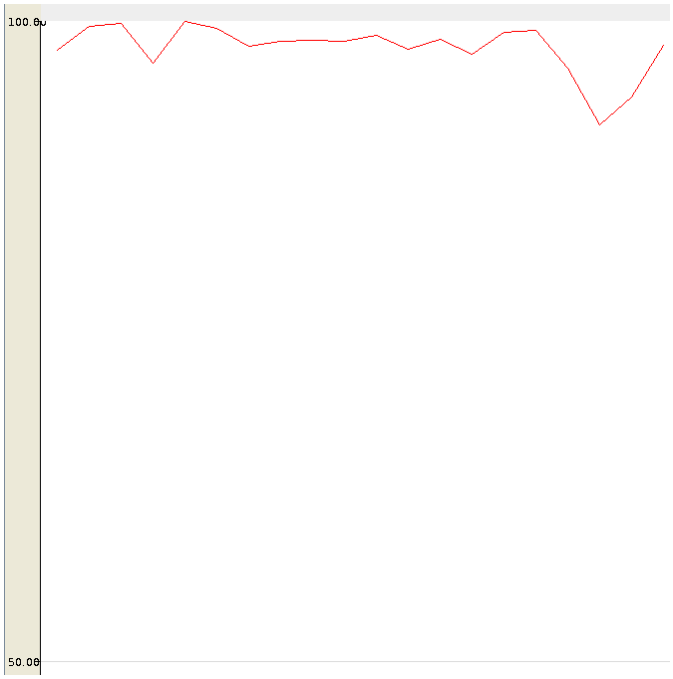
Para resolver este problema se ha usado la siguiente orden:

```
1  EvaluatePrequential
2    -l (drift.SingleClassifierDrift -l trees.HoeffdingTree)
3    -s (generators.RandomRBFGeneratorDrift
4      -r 1
5      -i 1
6      -a 7
7      -n 3
8      -k 3
9      -s 0.001
10   ) -i 2000000
11 -f 100000
```

El código es el mismo que en el experimento 2.3, sólo tenemos que cambiar el modelo a un clasificador simple basado en reemplazar el clasificador actual cuando se detecta un cambio de concepto. Este cambio lo podemos apreciar en la línea dos. Las semillas se han cambiado tanto para la generación del modelo como para la generación de las instancias. Los resultados obtenidos son los siguientes:

Semilla	**% Acierto**	**Kappa**
1	97.7	95.383
2	92.5	68.849
3	97.7	95.095
4	96.2	92.267
5	92.4	84.716

La evolución de la curva de aciertos es la siguiente:



2.5.2. Ejercicio 2.

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

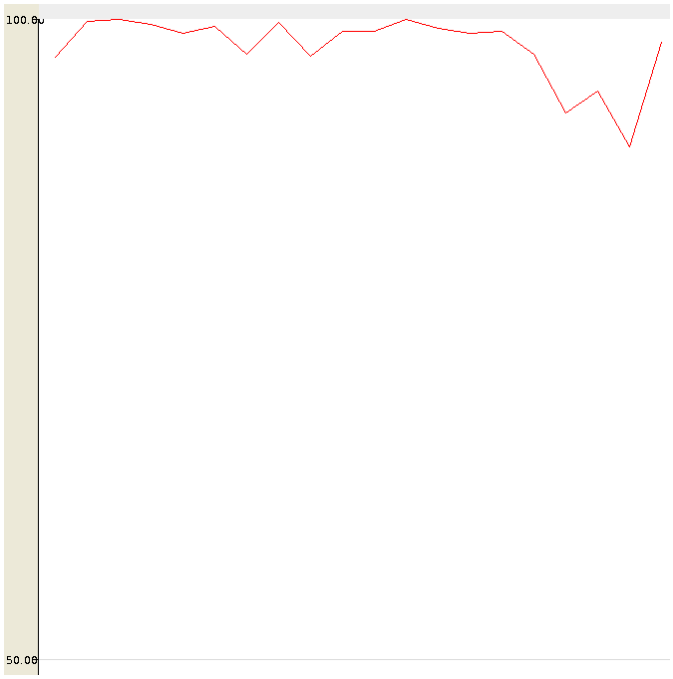
Para resolver este problema se ha usado la siguiente orden:

```
1  EvaluatePrequential
2    -l (drift.SingleClassifierDrift -l trees.HoeffdingAdaptiveTree)
3    -s (generators.RandomRBFGeneratorDrift
4      -r 1
5      -i 1
6      -a 7
7      -n 3
8      -k 3
9      -s 0.001
10   ) -i 2000000
11 -f 100000
```

El código es el mismo, sólo tenemos que cambiar el modelo que vamos a aprender en la línea dos. Las semillas se han cambiado tanto para la generación del modelo como para la generación de las instancias. Los resultados obtenidos son los siguientes:

Semilla	**% Acierto**	**Kappa**
1	97.3	94.579
2	93.3	71.189
3	95.4	90.269
4	95.2	90.223
5	92.7	85.220

La evolución de la curva de aciertos es la siguiente:



Como parte extra, vamos a comparar ambos experimentos, tal y como hemos hecho en los apartados anteriores:

```
shapiro.test(accuracy.standard)$p.value # 0.07903595
shapiro.test(kappa.standard)$p.value # 0.106312
shapiro.test(accuracy.adaptative)$p.value # 0.7089297
shapiro.test(kappa.adaptative)$p.value # 0.2067483

tseries::jarque.bera.test(accuracy.standard)$p.value # 0.6983299
tseries::jarque.bera.test(kappa.standard)$p.value # 0.6258841
tseries::jarque.bera.test(accuracy.adaptative)$p.value # 0.8432267
tseries::jarque.bera.test(kappa.adaptative)$p.value # 0.6330029
```

Cómo podemos ver, los datos son normales, así que podemos aplicar un test de *Student*:

```
t.test(accuracy.standard, accuracy.adaptative)$p.value # 0.7302075
t.test(kappa.standard, kappa.adaptative)$p.value # 0.8844594
```

Los test no si pasan, ya que el p-valor obtenido es mayor que 0.05, lo cual nos indica que no hay diferencia entre los algoritmos.

2.5.3. Ejercicio 3.

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.

Por lo general, dentro de cada experimento, el mejor resultado lo hemos obtenido usando la versión adaptativa del modelo *HoeffdingTree*. Si comparamos los resultados obtenidos en los distintos experimentos podemos ver que, según avanzan los experimentos, obtenemos un modelo más estable. Veamos una tabla comparativa del acierto para la versión adaptativa del algoritmo:

Semilla	**2.3**	**2.4**	**2.5**
1	96.249	96.6	97.3
2	92.647	95.3	93.3
3	96.309	98.0	95.4
4	95.983	96.3	95.2
5	90.996	91.9	92.7
media	94.4368	95.62	94.78

Aunque no haya mucha diferencia, podemos ver como, los modelos de los experimentos *2.4* y *2.5* son mejores. Esto se debe a que son experimentos donde se han introduzo técnicas para que el modelos se adapte mejor a los cambios de contexto, lo cual es la mayor diferencia de estos experimentos con el *2.3*. La diferencia entre los experimentos *2.4* y *2.5* es el cómo se trabaja el cambio de contexto. En el primero (*2.4*) se olvidan las instancias pasadas cuando se detecta el cambio de contexto mientras que en el segundo (*2.5*) se reinicializa el modelo cuando se detecta el cambio de contexto.