Series temporales y minería de flujos de datos

Máster Universitario en Ciencia de Datos e Ingeniería de Computadores

Trabajo autónomo II: Minería de Flujo de Datos

Jesús Sánchez de Castro

jsdc94@correo.ugr.es

1 de Septiembre de 2018

Parte teórica

Explicar 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.

La clasificación es un problema de aprendizaje supervisado en el que la salida o variable objetivo a predecir es categórica. Existen diferentes modelos que buscan estimar una función que prediga el valor de la variable objetivo con un conjunto de variables de entrada. La función f(variables de entrada) = variable salida real se desconoce, a partir de los datos de entrenamiento se pretende estimar la f() más similar a la que predeciría el valor exacto de salida con las entradas dadas.

La clasificación realizada en la práctica es estática (offline) o dinámica (online). Los algoritmos offline son incrementales. El modelo se va construyendo conforme va analizando los datos. Estos modelos pueden consultar más de una vez el dato, lo que los diferencia de los algoritmos online. Estos algoritmos buscan mejorar la eficiencia al hacer que la complejidad sea independiente del número de datos.

Por otro lado, el aprendizaje con flujos de datos está diseñando para ir construyendo y mejorando el modelo conforme llegan los datos de forma indefinida. Además, cuentan con mecanismos de detección y tratamiento de concept drift, lo que ayuda al modelo a recuperarse y no perder la robustez que perdería un modelo sin esta capacidad. Estos modelos solo analizan una vez los datos, posteriormente son desechados. Esto se debe a que estos algoritmos están pensados para sistemas en los que la llegada de datos es constante y la memoria y almacenamiento del sistema son limitadas.

En este trabajo se emplean los algoritmos HoeffdingTree y HoeffdingAdaptiveTree:

HoeffdingTree: es un algoritmo de construcción de árboles de decisión incremental. Se basan en la desigualdad de Hoeffding de Teoría de la Probabilidad, la cual proporciona una cota superior a la probabilidad de que la suma de variables aleatorias se desvíe una cierta cantidad de su valor esperado. Esto permite tener cierta información sobre la capacidad de variación que existe en los datos.

Los nodos intermedios de este árbol contienen pruebas que se aplican a las instancias para dividir los datos en las ramas siguientes. Lo importante en los árboles de decisión es cuando dividir un nodo y sobre que variable será la prueba discriminatoria. Es común emplear medidas

basadas en la teoría de la información (incertidumbre) para la selección de la prueba. Al igual que ocurre con los RandomForest, una pequeña cantidad de datos puede resultar en un particionado bueno de los nodos del árbol, esta es una de las razones por las que estos modelos son rápidos y robustos. Posteriormente, se puede generar mejores pruebas con el uso incremental de los datos. A diferencia de la versión Adaptive, este algoritmo no contempla la detección ni tratamiento de concept drift.

HoeffdingAdaptiveTree: Este algoritmo pertenece a la familia de soluciones basadas en instancias (ventana). Este tipo de algoritmos contiene mecanismos para olvidar datos antiguos, esto se basa en la idea de que los datos más recientes contienen características del contexto actual y por lo tanto la relevancia decrece con el tiempo.

En este algoritmo se usa la técnica de detección de desvío de concepto ADWIN (ADaptative sliding WINdow) para para monitorizar el rendimiento de las ramas del árbol generado y sustituirlas por nuevas ramas. Se mantienen estadísticas en todos los nodos y se tienen en cuenta divisiones candidatas. Periódicamente se evalúa la validez del modelo con los datos nuevos de la ventana y si una rama alternativa es mejor que un nodo existente se sustituye.

Validación/evaluación: En flujos de datos es común testear nuevas instancias no vistas anteriormente y después incluirlas en el conjunto de entrenamiento. En esta práctica se han empleado test-then-train. Cuando se evalúa en este orden, se mantiene actualizado la precisión del modelo constantemente. Este modelo tiene la ventaja de que no es necesario separar una parte de los datos como se haría en una estrategia de validación hold out clásica. De esta forma, se extraer el máximo valor de los datos de los que se dispone. Además, esto genera una gráfica de precisión a lo largo del tiempo que irá perdiendo significancia gradualmente.

También existen otras posibilidades como la evaluación periódica del modelo con un conjunto de hold out. Existen versiones de validación test-then-train con ventanas (de nuevo aplicando el concepto de lo más reciente es importate). Es importante destacar que existen diferentes formas de evaluar con test-then-train, elemento a elemento secuencialmente o en “chunks” o particiones de los datos secuencialmente. Estas técnicas son mucho más adecuadas que los clásicos hold out o cross validation.

Explicar en qué consiste el problema de concept drift y qué técnicas conoce para resolverlo en clasificación.

Dentro del problema de aprendizaje con flujos de datos, existe el concepto de concept drift. Este fenómeno ocurre cuando la naturaleza de los datos cambia con el tiempo. Existen muchas razones que propician este fenómeno, existe una gran cantidad de ruido en el flujo por alguna razón, varían características que no estaban completadas en el modelo original, existen alteraciones estacionales, etc. Esto se debe a la naturaleza cambiante de algunas características de los fenómenos que se intentan modelar (Predicciones sobre clientes en una tienda al llegar la navidad).

La forma en que se da el concept drift puede ser de diferentes maneras, un gran cambio en poco tiempo, cambios más graduales, cambios incrementales, recurrentes, etc. Esto dependerá de la naturaleza del problema.

Como se ha mencionado en el apartado anterior existen diferentes familias de algoritmos cuando se trata del concept drift. Un ejemplo de algoritmo es el VFDT adaptativo que se ha empleado en la práctica. Los algoritmos se basan en aprendizaje online, uso de ventanas,

ensembles o técnicas de detección de desvío. Los algoritmos que emplean ventanas tienen la dificultad del ajuste de tamaño de ventana. También se puede aplicar un factor de decadencia gradual para que los datos pierda relevancia.

Los algoritmos basados en ensembles realizan combinaciones de modelos. Esta técnica es muy exitosa en problemas de datos estáticos y la decisión colectiva obtiene buenos resultados. Además, la introducción de variabilidad o aleatoriedad en la construcción de múltiples modelos puede resultar en soluciones que difícilmente se habrían encontrado de otra forma.

Los dos enfoquen más destacados son Streaming Ensemble Algorithm (SEA) y Accuracy Weighted Ensemble (AWE). Conforme llegan datos, se va cogiendo particiones de estos para entrenar modelos y aquellos que no son precisos son reemplezados por nuevos modelos más prometedores. Como en todo problema de emsembles, el voto puede ser por mayoría (SEA) o puede haber esquemas más sofisticados como con AWE.

Dynamic Weighted Majority es un modelo que va ponderando clasificadores por su precisión y si bajan de un umbral son eliminados. Los métodos vistos hasta ahora dotan al aprendizaje de adaptabilidad a los nuevos datos. El siguiente paradigma se basa en la detección del concept drift y de su contrarresto inmediato.

Se puede detectar un cambio de concepto mediante variaciones considerables de la precisión o analizando la distribución de los datos. Ante se mencionó ADWIN, empleado por el VFDT de MOA. Este método contiene dos ventanas que va analizando, si las medidas de estas son lo suficientemente grandes se eliminan los datos de la ventana más antigua. Estos algoritmos buscan identificar los cambios en el rendimiento de los algoritmos o variaciones en los datos (distribución). DDM es otro algoritmo empleado. Se trata de un detector de cambios de concepto. Analiza la variación del error y vuelve a entrenar el modelo cuando dicha variación pasa un umbral.

Parte práctica

Sección 2.1 Entrenamiento offline (estacionario) y evaluación posterior.

En esta sección se pide entrenar un modelo HoeffdingTree estacionario con un total de 1.000.000 instancias provenientes del flujo de WaveFromGenerator con semilla aleatoria 2. Después, se evalúa dicho modelo con otro millón de instancias generadas con el mismo generador, pero con semilla aleatoria 4. Se ha de crear una población de resultados remitiendo el proceso con diferentes semillas de entrenamiento.

El comando necesario para realizar esta tarea es:

```
java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask "EvaluateModel -m (LearnModel -l
trees.HoeffdingTree -s (generators.WaveformGenerator -i $i) -m 1000000) -s
(generators.WaveformGenerator -i 4) -i 1000000" > salida$i.csv
```

A continuación, se explican las decisiones tomadas en cada función:

- **EvaluateModel**: Permite evaluar un modelo entrenado. Para especificar que modelo evaluar se emplea el parámetro “-m”. Debido a que MOA permite usar concatenar tareas con los pipeline, se pasa a -m el modelo entre paréntesis. Con “-s” se especifica el stream de datos (generators.WaveformGenerator) que emplea en la evaluación, es el mismo empleado para train pero con otra semilla. Por último, se especifica con “-i” el número de instancias de evaluación, un millón.
- **LearnModel**: Permite entrenar un modelo con el algoritmo especificado con “-l”, en este caso HoeffdingTree, con flujo de datos de generators.WaveformGenerator . Con “-m” se elige el número de instancias de entrenamiento, se usan un millón de instancias.

- **generators.WaveformGenerator**: genera un flujo de datos para un problema de clasificación. Con el parámetro "-i" se establece la semilla aleatoria que definirá el flujo.

Una vez explicado el funcionamiento de la orden, se diseña un script con un bucle for que itere sobre 30 semillas empleadas en train y evaluadas con la semilla 4. Para realizar esta tarea se emplea el siguiente script.

```
#!/bin/bash
# Script tarea 2.1 HoeffdingTree estacionario con 30 soluciones

for i in `seq 5 35`
do
    echo "modelo $i"
    "tareaMOA"
done

read  -n 1 -p "Exit." mainmenuinput
```

Ilustración 1. Script para la tarea 2.1

Para obtener diferentes salidas se crea un bucle for con la secuencia 5…35, para no pisar la semilla 4 que empleamos para evaluar. En la orden descrita arriba para el primer ejercicio solo hay que sustituir "$i" en la "-i" de la semilla de entrenamiento y en el nombre del fichero de salida para guardar distintas soluciones. El procedimiento será el mismo siempre que se requieran varias semillas.

El siguiente paso es emplear el algoritmo HoeffdingAdaptativeTree. Se realiza el cambio en la llamada del algoritmo y se ejecuta un script con las mismas iteraciones. A continuación, se realizan los test estadísticos para ver si son diferentes los modelos. La población de resultados se muestra a continuación:

Tabla 1. Resultados apartado 2.1.

Semilla	2.1 precision	2.1 kappa	2.1 a precision	2.1bkappa
5	84,481	76,723	84,262	76,395
6	84,342	76,514	84,368	76,554
7	84,799	77,2	84,271	76,408
8	84,153	76,231	84,243	76,367
9	84,641	76,963	84,478	76,719
10	84,578	76,869	84,326	76,491
11	84,539	76,81	84,371	76,558
12	84,457	76,688	84,416	76,627
13	84,369	76,555	84,498	76,749
14	84,547	76,822	84,415	76,624
15	84,648	76,974	84,229	76,345
16	84,626	76,94	84,328	76,494
17	84,513	76,772	84,358	76,539
18	84,434	76,653	84,456	76,685
19	84,605	76,91	84,451	76,679
20	84,568	76,853	84,459	76,69
21	84,518	76,779	84,553	76,831
22	84,657	76,988	84,432	76,65
23	84,543	76,815	84,686	77,03
24	84,754	77,132	84,485	76,729
25	84,646	76,971	84,589	76,886
26	84,586	76,88	84,372	76,559
27	84,488	76,734	84,476	76,716
28	83,529	75,294	83,978	75,968

29	84,591	76,888	84,379	76,57
30	84,505	76,759	84,222	76,335
31	84,553	76,83	84,326	76,49
32	84,528	76,793	84,489	76,736
33	84,609	76,915	84,495	76,744
34	84,344	76,519	84,313	76,471
35	84,502	76,755	84,276	76,415

Como se observa en la tabla los resultados son prácticamente idénticos a diferencia del primer decimal. A continuación, se muestra el resultado de los tests.

Tabla 2. Tests de normalidad y diferencia en los modelos.

```
        Shapiro-Wilk normality test

data:  data[, 1]
W = 0.70025, p-value = 1.573e-06


        Shapiro-Wilk normality test

data:  data[, 2]
W = 0.95512, p-value = 0.2313


        Wilcoxon signed rank test

data:  data[, 1] and data[, 2]
V = 391, p-value = 0.0006666
alternative hypothesis: true location shift is not equal to 0

[1] "Media 1:84.5057333333333"
[1] "Media 2:84.3912666666667"
```

Como se puede observar, el primer modelo no tiene datos normales y el segundo si, se emplea un test no paramétrico y se ve que los modelos son significativamente diferentes entre ellos. En este caso es mejor el primer modelo pues tiene una precisión media un poco mayor. Lo resultados son muy similares pues al aplicar un enfoque estático y en el que no existe concept drift, estos algoritmos VFDT y la versión adaptive deben tener el mismo funcionamiento.

Sección 2.2 Entrenamiento online

Para esta sección se va a cambiar la forma de entrenar y evaluar el modelo. Se emplea la orden EvaluateInterleavedTestThenTrain. Se requiere 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. Se trata de una tarea muy similar a la anterior, se cambia el método de train y test y se añade la frecuencia de muestreo. La orden para esta tarea es:

java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask “EvaluateInterleavedTestThenTrain -l trees.HoeffdingTree -s (generators.WaveformGenerator -i $i) -i 1000000 -f 10000” > salida$i.csv

A continuación, se explican las decisiones tomadas en cada función:

- **EvaluateInterleavedTestThenTrain**: Evalúa un clasificador testeando y después entrenando con particiones de datos en secuencia. Esta es la técnica test-then-train, en la cada instancia que llega al modelo, primero se usa para evaluarlo en test y posteriormente se añade como entrenamiento. El parámetro “-i” es el número de instancias que empleará el método para testear y después entrenar el modelo y “-f” es la frecuencia de muestro, el tamaño de las particiones de datos que coge para realizar test-then-train del flujo de datos.

Una vez definida la orden, se usa un script como el anterior para ejecutar varias semillas. Se emplean las semillas del 2 al 32. Después, para la versión adaptativa, se sigue el mismo proceso que en la sección 2.1, se cambia trees.HoeffdingTree por tree.HoeffdingAdaptiveTree y se ejecuta con las mismas semillas. A continuación, se muestran los resultados de los modelos y los tests.

Tabla 3. Resultados apartado 2.2.

Semilla	2.2 precision	2.2 kappa	2.2bprecision	2.2bkappa
2	83.785	75.677	83.731	75.597
3	83.888	75.830	83.788	75.679
4	84.045	76.069	83.796	75.696
5	83.840	75.760	83.714	75.571
6	83.906	75.859	83.841	75.761
7	83.887	75.829	83.778	75.667
8	83.869	75.803	83.897	75.845
9	83.788	75.682	83.828	75.743
10	83.848	75.771	83.900	75.850
11	83.746	75.618	83.741	75.611
12	83.839	75.759	83.741	75.612
13	83.976	75.964	83.894	75.842
14	83.880	75.819	83.858	75.786
15	83.984	75.976	83.875	75.812
16	83.834	75.752	83.888	75.832
17	83.896	75.845	83.739	75.608
18	83.841	75.762	83.761	75.643
19	83.788	75.683	83.750	75.626
20	83.815	75.723	83.823	75.734
21	83.862	75.793	83.850	75.774
22	83.978	75.967	83.863	75.795
23	83.846	75.770	83.651	75.476
24	83.886	75.829	83.811	75.716
25	83.965	75.947	83.866	75.798
26	83.903	75.856	83.819	75.728
27	83.820	75.729	83.852	75.776
28	83.900	75.851	83.814	75.722
29	83.836	75.754	83.822	75.733
30	83.899	75.847	83.973	75.959
31	83.878	75.817	83.422	75.132
32	83.872	75.809	83.887	75.831

Tabla 4. Resultados tests estadísticos apartado 2.2.

```
        Shapiro-Wilk normality test

data:  data[, 1]
W = 0.95625, p-value = 0.2477


        Shapiro-Wilk normality test

data:  data[, 2]
W = 0.83414, p-value = 0.0002937


        Wilcoxon signed rank test

data:  data[, 1] and data[, 2]
V = 396, p-value = 0.0004184
alternative hypothesis: true location shift is not equal to 0

[1] "Media 1:83.8771733333333"
[1] "Media 2:83.808"
```

Como se ve de nuevo, no se puede asumir la normalidad de los datos del modelo adaptive por lo que se emplea wilconxon de nuevo y el test asume que de nuevo son diferentes. Es ligeramente mejor el primero. Ocurre algo similar al apartado anterior. Debido a que no existe concept drift en los datos, el algoritmo se debe comportar de forma casi idéntica, dando resultados muy similares.

Sección 2.3 Entrenamiento online con 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.

RandomRBFGeneratorDrift: Es un generador de datos con concept drift que emplea un generador de función de base radial (RBF). Los parámetros empleados son “-i” para la semilla de generación de instancias, “-r” la semilla para la generación del modelo (centroides generados aleatoriamente, etc.), “-c” para elegir el número de clases, “-a” para elegir el número de atributos y “-n” para elegir el número de centroides. Estos atributos los comparte con la versión RandomRBFGenerator sin drift. Los parámetros de drift son “-s” la velocidad con que cambian los centroides y “-k” el número de centroides con drift. La orden para esta tarea es:

(generators.RandomRBFGeneratorDrift -i $i -r $i -c 2 -a 7 -n 3 -s 0.001 -k 3)

Configurado desde el GUI:

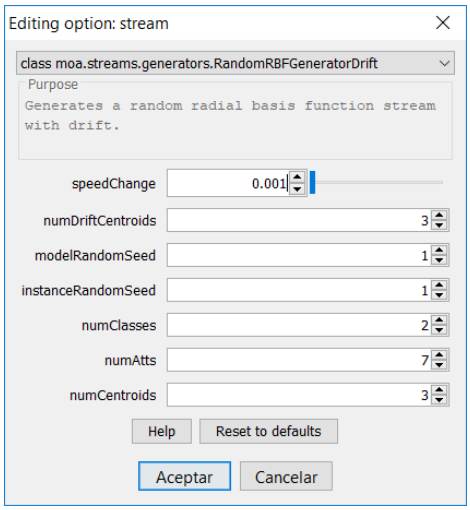

Ilustración 2. Configuración del generador de concept drift desde GUI.

La orden entera desde GUI es: EvaluateInterleavedTestThenTrain -l trees.HoeffdingTree -s (generators.RandomRBFGeneratorDrift -s 1.0 -k 3 -a 7 -n 3) -i 2000000. (La frecuencia es la predeterminada). Con 2 millones de instancias la evolución del HoeffdingTree en el GUI se muestra en la siguiente Ilustración.

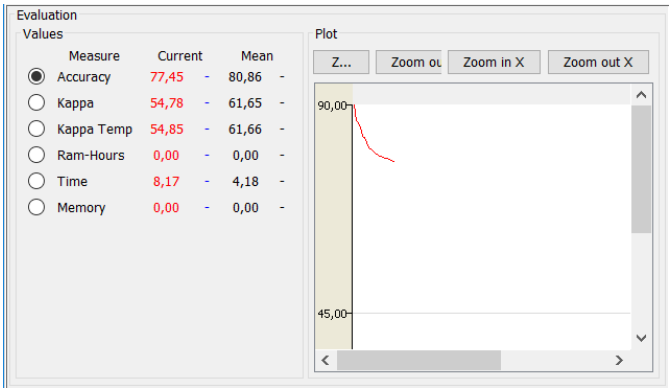

Ilustración 3. HoeffdingTree con concept drift.

La versión con HoeffdingAdaptiveTree se muestra en la siguiente Ilustración. Se emplea la siguiente orden:

EvaluateInterleavedTestThenTrain -l trees.HoeffdingAdaptiveTree -s (generators.RandomRBFGeneratorDrift -s 0.001 -k 3 -a 7 -n 3) -i 2000000.

Como se puede ver en las Ilustraciones 3 y 4, la versión HoeffdingTree sufre bastante ante el concept drift decayendo la precisión hasta 77,45% comenzando por un valor de 90%. Se aprecia en la Ilustración de la versión adaptativa, acaba teniendo un resultado muy superior del 96% de precisión. Esto era de esperar pues el segundo algoritmo está preparado para identificar y adaptarse a los concept drift.

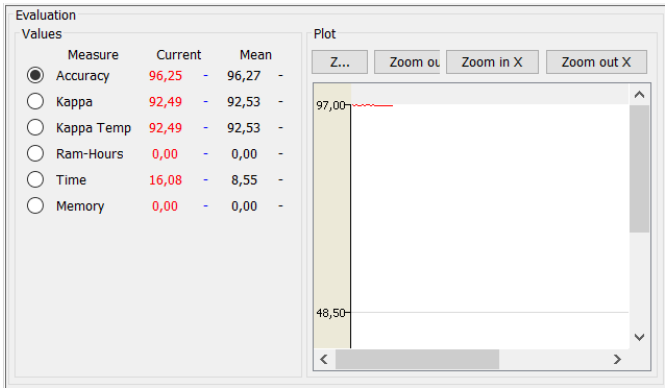

Ilustración 4. HoeffdingTree con concept drift.

A continuación, se muestra una versión de las ejecuciones con más 100.000.000 instancias.

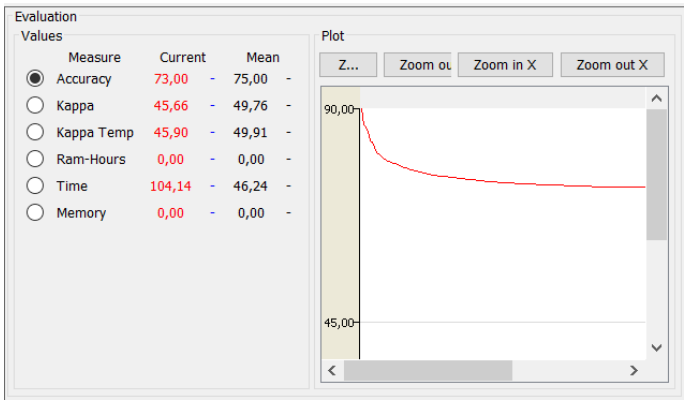

Ilustración 5. HoeffdingTree con concept drift 20 millones de instancias.

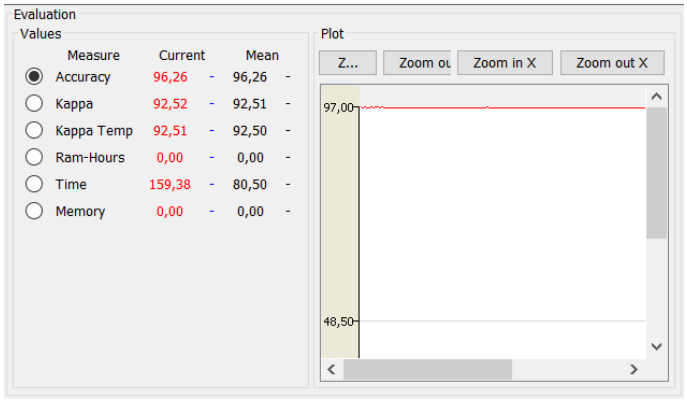

Ilustración 6. HoeffdingAdaptiveTree con concept drift 20 millones de instancias.

Los comportamientos son los esperados, la versión sin concept drift decae un poco más pero finalmente se estabiliza, por otro lado, la versión preparada para concept drift tiene un comportamiento constante. A continuación, se presentan los resultados.

Tabla 5. Resultados apartado 2.3.

Semilla	2.3precision	2.3kappa	2.3bprecision	2.3bkappa
1	77.455	54.776	96.249	92.491
2	77.337	54.528	96.194	92.380
3	77.235	54.316	96.204	92.400
4	77.303	54.458	96.170	92.333

5	77.120	54.116	96.159	92.311
6	77.171	54.196	96.252	92.496
7	76.946	53.753	96.189	92.372
8	77.302	54.477	96.165	92.322
9	77.284	54.412	96.236	92.465
10	77.223	54.310	96.186	92.365
11	76.862	53.553	96.262	92.516
12	76.957	53.775	96.178	92.348
13	77.277	54.406	96.231	92.454
14	76.589	53.020	96.240	92.472
15	76.813	53.467	96.239	92.469
16	77.301	54.455	96.191	92.374
17	77.343	54.543	96.299	92.591
18	77.159	54.199	96.270	92.532
19	77.858	55.575	96.215	92.422
20	77.398	54.639	96.251	92.495
21	77.935	55.713	96.271	92.533
22	77.486	54.851	96.248	92.488
23	76.979	53.804	96.286	92.564
24	77.788	55.446	96.205	92.403
25	77.662	55.196	96.203	92.399
26	77.464	54.781	96.209	92.410
27	77.776	55.407	96.298	92.587
28	76.803	53.462	96.189	92.371
29	77.065	53.979	96.191	92.375
30	76.933	53.740	96.209	92.410

Ilustración 6. Resultados tests estadísticos sección 2.3.

```
        Shapiro-Wilk normality test

data:  data[, 1]
W = 0.97077, p-value = 0.5807


        Shapiro-Wilk normality test

data:  data[, 2]
W = 0.95319, p-value = 0.2214


        Welch Two Sample t-test

data:  data[, 1] and data[, 2]
t = -303.81, df = 28.801, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -19.09557 -18.84011
sample estimates:
mean of x mean of y
 77.25409  96.22194

[1] "Media 1:77.2540948275862"
[1] "Media 2:96.2219362068965"
```

En este caso ambas soluciones siguen una distribución normal y se aplica el test de student. El test con un valor inferior a 0.05 indica que los resultados son diferentes. Por lo tanto, es obvio que el árbol adaptativo es mucho mejor trabajando ante concept drift, la versión original no contiene mecanismos para esto.

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

En esta sección se realiza la misma experimentación que la anterior, pero cambiando la evaluación a Prequential. Se trata de una versión de test-then-train con una ventana. Se establece el tamaño a 1.000. La orden para esta sección es:

java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask "EvaluatePrequential -l trees.HoeffdingAdaptiveTree -s (generators.RandomRBFGeneratorDrift -i $i -r 1 -c 2 -a 7 -n 3 -s 0.001 -k 3) -i 2000000 -w 1000"

Solo habría que cambiar los nombres de los algoritmos de árboles. En este caso, EvaluatePrequential tiene el parámetro "-w" para definir el tamaño de la ventana.

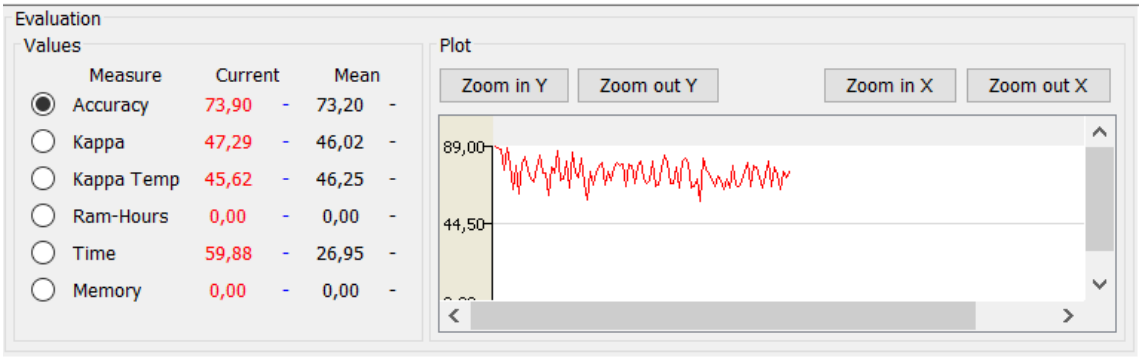

Ilustración 7 HoeffdingTree con evaluación de ventana tamaño 1000.

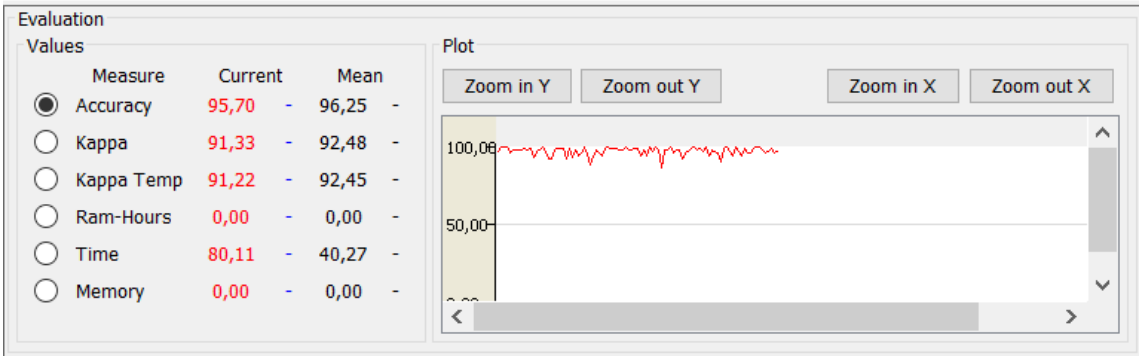

Ilustración 8. HoeffdingAdaptiveTree con Prequential de ventana 1000.

Como era de esperar, parece que la versión adaptativa con el mecanismo para olvidar datos pasados obtiene mejor resultado que le árbol no adaptativo. Aún así, cabe destacar que este mecanismo sirve para que el primer algoritmo se recupere de algunos concept drifts. A continuación, se muestran los resultados.

Tabla 7. Resultados sección 2.4.

Semilla	2.4precision	2.4kappa	2.4bprecision	2.4kappa
1	76.600	52.559	96.600	93.191
2	84.900	69.042	97.700	95.355
3	77.600	55.097	96.100	92.200
4	83.200	66.375	97.400	94.801
5	80.900	61.659	97.000	94.000
6	79.400	58.382	98.000	95.989
7	81.000	61.708	95.800	91.596
8	79.800	59.590	97.200	94.400

9	76.400	52.467	97.700	95.397
10	81.300	62.667	96.100	92.190
11	81.300	62.554	95.600	91.205
12	77.400	54.177	96.600	93.178
13	81.100	62.060	98.000	95.997
14	82.200	63.969	97.700	95.371
15	84.200	67.990	97.700	95.383
16	81.400	62.800	96.300	92.601
17	76.200	51.896	97.400	94.792
18	78.500	56.523	98.100	96.195
19	73.500	46.428	97.000	93.995
20	77.200	54.544	97.200	94.398
21	74.900	49.543	97.200	94.389
22	82.400	64.456	96.600	93.198
23	78.600	56.912	97.000	93.969
24	80.400	60.605	95.800	91.600
25	81.100	62.001	95.100	90.200
26	83.300	66.576	97.800	95.599
27	81.100	62.130	97.400	94.799
28	77.300	53.729	97.900	95.784
29	74.400	48.721	95.300	90.604
30	80.600	61.061	97.800	95.599

```
        Shapiro-Wilk normality test

data:  data[, 1]
W = 0.96435, p-value = 0.4184


        Shapiro-Wilk normality test

data:  data[, 2]
W = 0.91368, p-value = 0.02115


        Wilcoxon signed rank test with continuity correction

data:  data[, 1] and data[, 2]
V = 0, p-value = 2.701e-06
alternative hypothesis: true location shift is not equal to 0

[1] "Media 1:79.7103448275862"
[1] "Media 2:96.9827586206897"
```

Ilustración 9. Test estadísticos sección 2.4

En este caso, no se puede asumir la normalidad de los segundos resultados, se aplica Wilcoxon. El test indica que existen diferencias significativas y se puede concluir que el segundo modelo es mucho más robusto gracias a la ventana y el algoritmo adaptativo. Es importante remarcar que la ventana para un algoritmo no adaptativo ha tenido recuperaciones.

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

En este caso, en lugar de emplear ventanas, se reinicializa el modelo tran un concept drift. Se emplea el detector DDM, se detecta la variación en el error obtenido por el modelo y reinicializa el modelo. Esto depende de un umbral de error. La orden para lanzar este modelo es:

EvaluateInterleavedTestThenTrain -l (drift.SingleClassifierDrift -l trees.HoeffdingTree) -s (generators.RandomRBFGeneratorDrift -s 0.001 -k 3 -a 7 -n 3) -i 2000000

Para esta sección hay que cambiar en la evaluación el algoritmo solo e incluir el detector. Con "-l" le se indica al detector DDM que algoritmo se quiere emplear. El resto de la orden es mantener el EvaluateInterleavedTestThenTrain como método de evaluación. Los resultados se muestran a continuación.

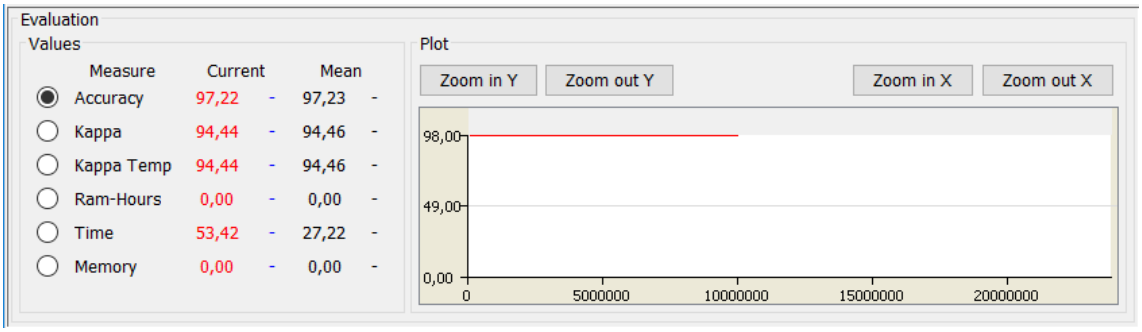

Ilustración 10. Modelo HoeffdingTree con DDM.

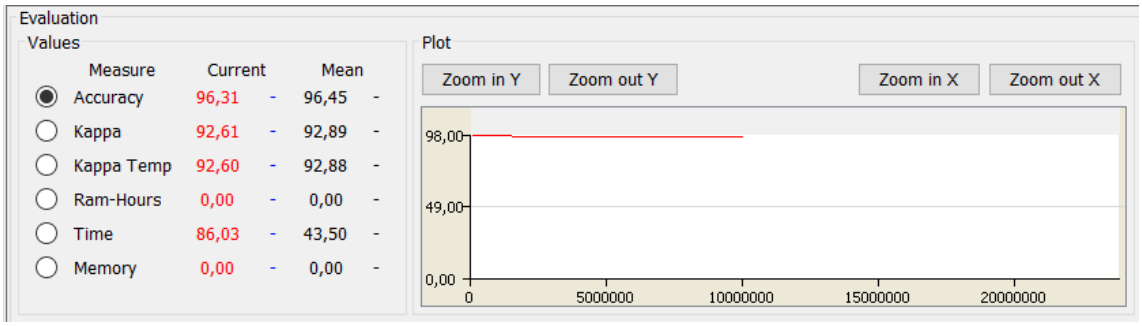

Ilustración 11. HoeffdingAdaptiveTree DDM.

Ambas soluciones son robustas. Sorprendentemente, la versión de HoeffdingTree ha dado un resultado muy estable. Esto puede indicar que aun que el algoritmo no tenga mecanismo para tratar el concept drift, el mecanismo externo DDM ha dado buenos resultados. En la versión adaptativa se ve una leve variación.

Semilla	2.5precision	2.5kappa	2.5bprecision	2.5bkappa
1	96.523	93.039	96.249	92.491
2	96.432	92.856	96.194	92.380
3	97.119	94.229	96.204	92.400
4	96.361	92.713	96.170	92.333
5	97.141	94.275	96.159	92.311
6	96.721	93.435	96.252	92.496
7	97.214	94.422	96.189	92.372
8	97.182	94.356	96.165	92.322
9	97.136	94.264	96.236	92.465
10	96.297	92.587	96.186	92.365
11	96.356	92.705	96.262	92.516
12	97.141	94.274	96.178	92.348
13	97.064	94.120	96.231	92.454
14	97.156	94.305	96.240	92.472
15	96.524	93.040	96.239	92.469

16	97.204	94.400	96.191	92.374
17	97.078	94.148	96.299	92.591
18	97.043	94.079	96.270	92.532
19	97.145	94.283	96.215	92.422
20	97.160	94.314	96.251	92.495
21	96.455	92.903	96.271	92.533
22	97.170	94.332	96.248	92.488
23	97.093	94.179	96.286	92.564
24	96.304	92.601	96.205	92.403
25	96.240	92.473	96.203	92.399
26	96.831	93.655	96.209	92.410
27	97.226	94.443	96.298	92.587
28	97.117	94.228	96.189	92.371
29	97.130	94.252	96.191	92.375
30	97.152	94.297	96.209	92.410

Ilustración 12. Resultados sección 2.5

```
        Shapiro-Wilk normality test

data:  data[, 1]
W = 0.75708, p-value = 1.554e-05


        Shapiro-Wilk normality test

data:  data[, 2]
W = 0.95289, p-value = 0.2174


        Wilcoxon rank sum test with continuity correction

data:  data[, 1] and data[, 2]
W = 829.5, p-value = 2.109e-10
alternative hypothesis: true location shift is not equal to 0

[1] "Media 1:96.9031724137931"
[1] "Media 2:96.2220689655172"
```

Ilustración 13. Resultados tests sección 2.5

No pudiendo asumir la normalidad de los datos, el test de Wilcoxon determina que existe diferencia entre los modelos y sorprendentemente el modelo básico junto a DDM da mejor resultado que el adaptive con DDM. Esto puede ser a que las dos mecánicas se “solapan” y ninguna de ellas deja funcionar adecuadamente a la otra. Se trata de una simple suposición, se habla de modelos de 96% de acierto con diferencias decimales, son ambos excelentes.