MINERÍA DE SERIES TEMPORALES Y FLUJO DE DATOS

MÁSTER EN CIENCIA DE DATOS E INGENIERIA DE COMPUTADORES

Trabajo autónomo II: Minería de Flujo de Datos

Autores
José Ángel Díaz García
joseangeldiazg02@correo.ugr.es

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Abril de 2018

Índice general

1. Introducción **4**

1.1. Problema a resolver 4

1.2. Objetivos 5

1.3. Organización del trabajo 6

2. Práctica **7**

2.1. Entrenamiento offline y evaluación posterior 7

2.2. Entrenamiento online 10

2.3. Entrenamiento online con CD 12

2.4. Entrenamiento online con CD y mecanismos para olvidar instancias pasadas 15

2.5. Entrenamiento online con CD y detección CD 16

3. Teoría **18**

3.1. Clasificación 18

3.2. Concept Drift 20

4. Conclusión **23**

Índice de figuras

1.1. Interfaz del software moa. 5

2.1. Comparación adaptativo y no adaptativo en online sin CD. . . 12
2.2. Evolución ACC para HT y HTAdaptativo para semilla 1. . . . 15
2.3. Evolución ACC para HT y HTAdaptativo para semilla 2. . . . 15
2.4. Evolución ACC para HT y HTAdaptativo para semilla 3 . . . 15

3.1. Tipos de concept drift. 21

Índice de tablas

2.1. Tabla de resultados del entrenamiento offline. 9
2.2. Tabla de resultados del entrenamiento online. 11
2.3. Tabla de resultados del entrenamiento online con CD. 14

Capítulo 1

Introducción

En este documento encontramos el resultado final alcanzado durante el estudio del apartado de **minería de flujos de datos**, enmarcado dentro de la asignatura de 'Minería de Series Temporales y Flujos de Datos' del máster en Ciencia de Datos de la Universidad de Granada. En este primer capítulo, veremos una introducción al problema a resolver así como a los objetivos a alcanzar con esta práctica y la organización del trabajo.

1.1. Problema a resolver

El problema a resolver en esta práctica se centrará en la resolución de ciertos problemas de clasificación con flujos de datos, de manera tanto estática como dinámica así como con cambio o sin cambio de contexto. Para resolver estos problemas, se propone el uso del software MOA [1] [2] el cual usaremos con la línea de comandos o con la interfaz gráfica que podemos ver en la figura 1.1.

Los problemas a resolver serán:

1. Entrenamiento offline (estacionario) y evaluación posterior.
2. Entrenamiento online.
3. Entrenamiento online en datos con concept drift.

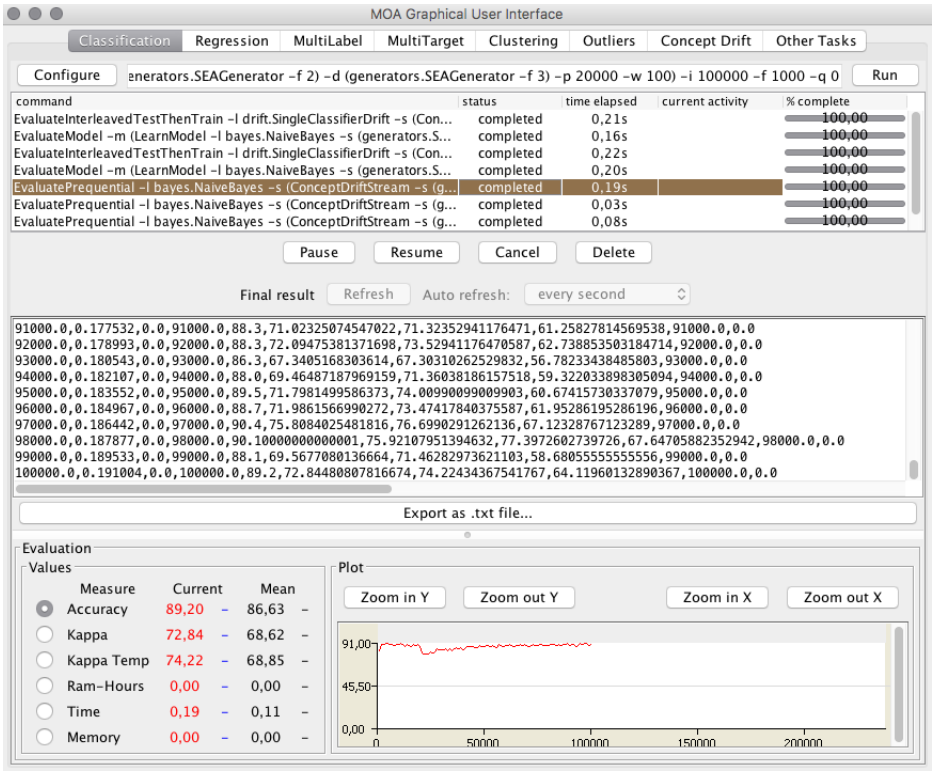

Figura 1.1: Interfaz del software moa.

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

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

Posteriormente a la solución de estos problemas, se propone una discusión teórica de los resultados, por lo que encontraremos la práctica dividida en dos apartados, por un lado el teórico (capítulo 3) y por otro el práctico (2).

1.2. Objetivos

Los objetivos de esta práctica serán:

- Asentar y comprender la materia teórica de la minería de flujo de datos vista durante el transcurso de la asignatura.
- Comprender el uso y formas de utilización del software MOA.
- Asentar el conocimiento sobre test estadísticos par comparación de modelos.
- Elaboración de una memoria donde se recojan todos los resultados de manera apropiada.

1.3. Organización del trabajo

La organización del presente documento, se centra en detallar cada uno de los pasos seguidos durante el estudio y resolución del problema planteado en esta introducción, tras la cual tendremos el contenido práctico en el capítulo 2 y el cual representa el grueso de esta memoria. Tras este capítulo encontramos el capítulo 3 donde desde un punto de vista teórico analizamos el concepto de **clasificación** y **concept drift** en minería de flujo de datos. Finalizaremos la memoria con las conclusiones obtenidas en el transcurso de finalización de la misma en el capítulo 4.

Capítulo 2

Práctica

En este capítulo encontramos el desarrollo práctico de este trabajo. El discurso de este capítulo está organizado por secciones, una para cada experimento a resolver con el software MOA.

2.1. Entrenamiento offline 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. Repetir el paso anterior, sustituyendo el clasificador por HoeffdingTree adaptativo. Responda a la pregunta: ¿Cree que algún clasificador es significativamente mejor que el otro en este tipo de problemas? Razone su respuesta.

Para resolver este problema y dar la respuesta a si un clasificador es significativamente mejor que otro deberemos usar test estadísticos. Para ello, primero generamos una muestra de resultados usando un script en el que

ejecutaremos varias veces nuestros experimentos para cada uno de los clasificadores. El resultado de ese script sería:

```
#!/bin/bash

for i in ‘seq 1 20‘;
    do
        eval "htnormal=htnormal$i.txt"
        eval "htadaptativo=htadaptativo$i.txt"
        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)" >  $htnormal \
        java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask  \
        "EvaluateModel -m (LearnModel -l trees.HoeffdingAdaptiveTree -s \
         (generators.WaveformGenerator -i $i)  -m 1000000) -s \
         (generators.WaveformGenerator -i 4)" > >  $htadaptativo
    done
```

En el anterior script, hemos añadido dos tareas del software MOA, una para cada clasificador (HoeffdingTree y HoeffdingTree Adaptativo) a las que añadimos el flujo generado por el WaveformGenerator con los parámetros que se nos pide en el ejercicio. Una vez ejecutado el algoritmo 20 veces, podemos ver los resultados en la tabla 2.1, sobre la cual, ejecutaremos los estadísticos necesarios para comprobar si hay diferencias significativas entre los algoritmos usados.

El primer paso será por tanto discernir sobre la normalidad de los datos para poder aplicar posteriormente test paramétricos o no paramétricos en función de la distribución de los mismos. Para comprobar la normalidad de los datos dado que estamos ante un problema con pocas muestras usaremos el test de **Cramer-von Mises**. Este test nos ofrece para cada uno de los vectores de muestras valores de p-value por encima de 0.05, lo que nos indica que podemos aceptar la hipótesis de distribución normal en los datos.

Realizada esta hipótesis, podemos aplicar un test paramétrico como el t-student para obtener información acerca de las diferencias en los resultados. Una vez aplicado el test sobre los datos, obtenemos un valor de p-value = 0.0006479 lo que nos lleva a concluir que si existen diferencias significativas

entre los datos y que el algoritmo **HoeffdingTree** con 84.52885acc de media es el mejor en este tipo de problemas frente al algoritmo HoeffdingTree Adaptativo con 84.39025acc de media.

Pese a lo que podríamos pensar, el algoritmo adaptativo se comporta peor en este tipo de problemas porque está pensado para adaptarse a cambios, es decir, es un algoritmo diseñado para los cambios de contexto en *streaming* de datos. Si carecemos de este problema y además, tenemos todos los datos disponibles de manera offline, algo que se asimilaría más al aprendizaje por lotes que al streaming propiamente dicho, el algoritmo HoeffdingTree con su 'simple' diseño incremental tiene más posibilidades de comportarse mejor.

	HT Acc	HT Kappa	HT Adaptative Acc	HT Adaptative Kappa
1	84.64	76.96	84.48	76.72
2	84.15	76.23	84.24	76.37
3	84.80	77.20	84.27	76.41
4	84.34	76.51	84.37	76.55
5	84.48	76.72	84.26	76.39
6	84.67	77.00	84.47	76.70
7	84.59	76.89	84.42	76.62
8	84.57	76.85	84.46	76.69
9	84.51	76.77	84.47	76.71
10	84.61	76.91	84.45	76.68
11	84.43	76.65	84.46	76.69
12	84.51	76.77	84.36	76.54
13	84.63	76.94	84.33	76.49
14	84.65	76.97	84.23	76.34
15	84.55	76.82	84.42	76.62
16	84.37	76.56	84.50	76.75
17	84.46	76.69	84.42	76.63
18	84.54	76.81	84.37	76.56
19	84.58	76.87	84.33	76.49
20	84.51	76.77	84.52	76.78

Tabla 2.1: Tabla de resultados del entrenamiento offline.

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. Repetir el paso anterior, sustituyendo el clasificador por HoeffdingTree Adaptativo. Responda a la pregunta: ¿Cree que algún clasificador es mejor que el otro en este tipo de problemas? Razone su respuesta.

En este caso hemos diseñado un problema online, con el generador WaveFormGenerator que genera un problema de predicción de 3 tipos de onda. Generaremos 100000 instancias con una frecuencia de 10000 por lo que los parámetros a usar serán -i 1000000 y -f 10000, para evaluar usaremos **Interleaved test then train** con la que cada instancia se usa para evaluar el modelo y luego para entrenar. Acorde a este uso, podemos predecir que el comportamiento del modelo irá mejorando a medida que llegan los datos pues habrá podido construir un modelo más solido cuantos más datos haya tratado.

El experimento para poder comparar ambos modelos sería el que podemos ver en el siguiente script.

```
#!/bin/bash

for i in ‘seq 1 20‘;
    do
        eval "htnormal=htnormal$i.txt"
        eval "htadaptativo=htadaptativo$i.txt"
        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" \
         >  $htnormal
        java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask  \
        "EvaluateInterleavedTestThenTrain -l trees.HoeffdingAdaptiveTree\
         -s (generators.WaveformGenerator -i $i) -i 1000000 -f 10000" \
         >  $htadaptativo
    done
```

Ejecutado el experimento, podemos ver los resultados obtenidos en la tabla 2.2. Sobre estos resultados, nuevamente ejecutaremos el test **Cramer-von Mises** que en ambos casos arrojará p-values altos que nos llevan a aceptar la hipótesis de distribución normal, por lo que nuevamente usando el test t-student podremos ver si hay diferencias significativas. El p-value en este caso es de 0.008351, lo que nos lleva nuevamente a concluir que las diferencias no se deben al azar y que el algoritmo no adaptativo vuelve a ser mejor en este tipo de problemas.

	HT Acc	HT Kappa	HT Adaptative Acc	HT Adaptative Kappa
1	83.89	75.84	83.80	75.71
2	83.79	75.68	83.73	75.60
3	83.89	75.83	83.79	75.68
4	84.05	76.07	83.80	75.70
5	83.84	75.76	83.71	75.57
6	83.91	75.86	83.84	75.76
7	83.89	75.83	83.78	75.67
8	83.87	75.80	83.90	75.85
9	83.79	75.68	83.83	75.74
10	83.85	75.77	83.90	75.85
11	83.75	75.62	83.74	75.61
12	83.84	75.76	83.74	75.61
13	83.98	75.96	83.89	75.84
14	83.88	75.82	83.86	75.79
15	83.98	75.98	83.87	75.81
16	83.83	75.75	83.89	75.83
17	83.90	75.84	83.74	75.61
18	83.84	75.76	83.76	75.64
19	83.79	75.68	83.75	75.63
20	83.82	75.72	83.82	75.73

Tabla 2.2: Tabla de resultados del entrenamiento online.

La conclusión a la que podemos llegar nuevamente en este punto es que al carecer de cambio de contexto el algoritmo no adaptativo se comporta mejor que el adaptativo, ya que nuevamente la capacidad de readaptarse en un problema en el que los datos seguirán llegando con la misma distribución no es necesaria. Igualmente, si analizamos el gráfico 2.1 donde hemos representado el comportamiento de ambos algoritmos para este ejemplo, podemos

extraer conclusiones interesantes. Por un lado, podemos ver como tal y como predijimos al inicio del apartado, el algoritmo aumenta su accuracy a medida que llegan mas muestras, pues como estas se usan para entrenar el modelo será mas rico al final. Otra conclusión interesante recae en la comparación de ambos algoritmos,, vemos como el normal baja en accuracy al inicio hasta que es capaz de construir un modelo solido, momento en el que empieza a mejorar. Por otro lado, vemos como en las etapas iniciales el algoritmo adaptativo, es capaz de mejorar bastante el no adaptativo ya que con pocos datos es capaz de readaptar sus ramas a los datos actuales sin necesidad de esperar nuevos.

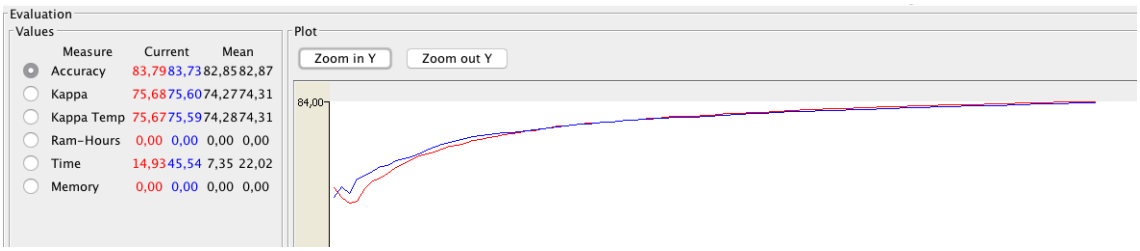

Figura 2.1: Comparación adaptativo y no adaptativo en online sin CD.

2.3. Entrenamiento online con CD

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. Repetir el paso anterior, sustituyendo el clasificador por HoeffdingTree adaptativo. Responda a la pregunta: ¿Cree que algún clasificador es mejor que el otro en este tipo de problemas? Razone su respuesta.

En este caso, se pide generar un flujo online con cambio de contexto, de primeras esto nos lleva a pensar que nuestro algoritmo adaptativo será capaz de readaptarse a los cambios de contexto y por tanto ofrecerá grandes resultados frente al no adaptativo. Para generar el flujo, usaremos el RandomRBFGeneratorDrift que genera una función radial aleatoria con cambio de contexto el evaluador y los clasificadores serán los mismos que los utilizados hasta ahora.

Nuevamente, como se pide saber si hay algún clasificador mejor para este problema, usaremos test estadísticos sobre una población de resultados construida con el siguiente script.

```
#!/bin/bash

for i in ‘seq 1 20‘;
    do
        eval "htnormal=htnormal$i.txt"
        eval "htadaptativo=htadaptativo$i.txt"
        java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask \
        "EvaluateInterleavedTestThenTrain -l trees.HoeffdingTree -s \
        (generators.RandomRBFGeneratorDrift -s 0.001 -k 3 -r 2 -i $i -a 7 -n 3) \
        -i 2000000" >  $htnormal
        java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask  \
        "EvaluateInterleavedTestThenTrain -l trees.HoeffdingAdaptiveTree -s \
         (generators.RandomRBFGeneratorDrift -s 0.001 -k 3 -r 3 -i $i -a 7 -n 3)\
 -i 2000000" >  $htadaptativo
    done
```

Una vez ejecutado el script y cargados los datos a R, podemos compilar los resultados en la tabla 2.3. Nuevamente, el test de normalidad nos dice con p-values elevados que estamos ante distribuciones normales de los resultados por lo que podemos aplicar un test paramétrico para saber si hay diferencias significativas, aunque en este caso, es claramente obvio que si que las habrá. El p-value del t-student nos ofrece un resultado **p-value <2.2e-16** lo que nos indica que si hay diferencias significativas y el **HoeffdingTree adaptativo** es el mejor algoritmo en este problema con una media de 96.28154acc frente al no adaptativo con 83.96403acc.

El resultado de este experimento era obvio, ya que al generar un cambio de contexto el algoritmo adaptativo se readapta y es capaz de hacer frente al mismo de manera sencilla, por decirlo de alguna manera. Por otro lado, el

algoritmo no adaptativo al generarse cambio de contexto con cierta regularidad hace que su comportamiento vaya empeorando ya que no es capaz de generalizar y cuando comienza a generalizar el cambio de contexto le hace empeorar.

	HT Acc	HT Kappa	HT Adaptative Acc	HT Adaptative Kappa
1	83.98	29.78	96.33	92.24
2	83.96	29.39	96.28	92.13
3	83.97	28.94	96.31	92.19
4	83.95	29.23	96.34	92.25
5	83.98	29.81	96.33	92.24
6	83.98	29.44	96.30	92.17
7	83.90	29.56	96.33	92.23
8	83.96	29.92	96.22	92.01
9	84.01	30.06	96.24	92.04
10	83.96	29.42	96.28	92.14
11	83.99	29.71	96.24	92.04
12	83.95	29.52	96.24	92.04
13	83.92	29.78	96.29	92.15
14	84.00	29.59	96.26	92.08
15	83.90	29.14	96.30	92.18
16	83.98	29.64	96.22	92.00
17	83.93	29.50	96.27	92.12
18	84.02	29.92	96.26	92.09
19	83.99	30.11	96.30	92.17
20	83.96	30.00	96.30	92.16

Tabla 2.3: Tabla de resultados del entrenamiento online con CD.

Dado que se pedía comprar el modelo con distintos valores de semilla y comprobar sus gráficas, se han elaborado una serie de baterías de prueba en las que se enfrentan algoritmo adaptativo y no adaptativo con semillas aleatorias que van de 1 a 3. Los resultados de estos experimentos pueden verse en los gráficos 2.2, 2.3 y 2.4, donde podemos ver tal y como era de esperar como el algoritmo adaptativo (rojo) se encuentra por encima en todos los casos además de mantenerse estable en el valor de accuracy ofrecido. Por otro lado, si nos fijamos en al algoritmo no adaptativo (azul) podemos concluir como los cambios de contexto le hacen empeorar teniendo siempre una tendencia descendente en los experimentos realizados.

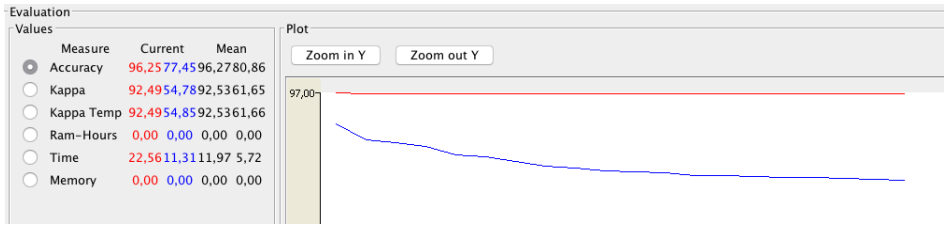

Figura 2.2: Evolución ACC para HT y HTAdaptativo para semilla 1.

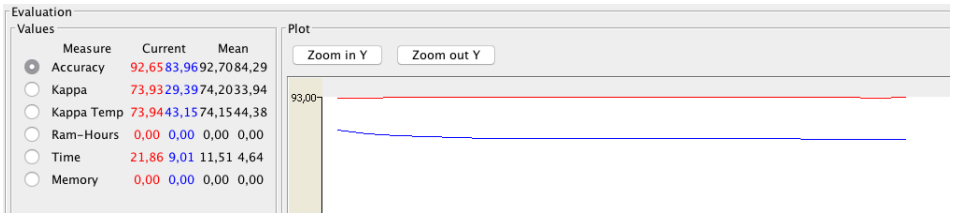

Figura 2.3: Evolución ACC para HT y HTAdaptativo para semilla 2.

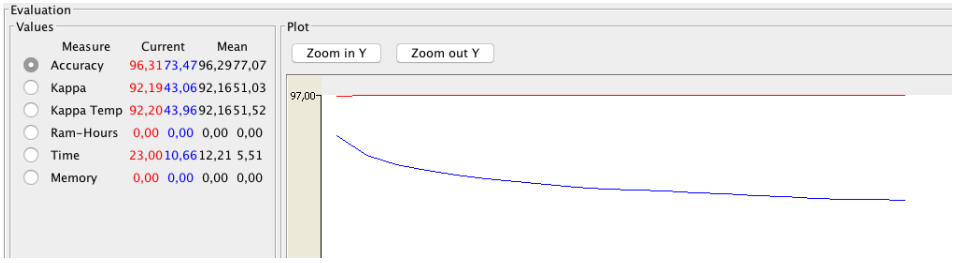

Figura 2.4: Evolución ACC para HT y HTAdaptativo para semilla 3 .

2.4. Entrenamiento online con CD y 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.

En este caso, dado que no se pide discernir entre ambos algoritmos no se ha elaborado una población de resultados, sino que se han ejecutado las siguientes tareas a través de la GUI de MOA para tener recursos gráficos para el estudio del experimento y su interpretación.

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

```
EvaluatePrequential -l trees.HoeffdingTree -s \
(generators.RandomRBFGeneratorDrift -s 0.001 -k 3 -a 7 -n 3)\
 -i 2000000
```

En las tareas anteriores hemos usado los mismos clasificadores que en el punto anterior, así como el mismo generador de flujo con cambio de contexto, el cambio en este punto está en añadir el evaluador Prequential con ventana deslizante de 1000, parámetro por defecto de esta evaluación y que no viene representado en la tarea por ese motivo.

2.5. Entrenamiento online con CD y detección CD

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.

```
EvaluateInterleavedTestThenTrain -l (drift.SingleClassifierDrift \
 -l trees.HoeffdingAdaptiveTree) -s (generators.RandomRBFGeneratorDrift\
```

```
 -s 1.0 -k 3 -a 7 -n 3) -i 2000000

EvaluateInterleavedTestThenTrain -l (drift.SingleClassifierDrift \
-l trees.HoeffdingTree) -s (generators.RandomRBFGeneratorDrift \
-s 1.0 -k 3 -a 7 -n 3) -i 2000000
```

Capítulo 3

Teoría

En este capítulo asentaremos los conceptos teóricos utilizados en el transcurso del anterior apartado. Comenzaremos con el concepto de **clasificación** y finalizaremos con el concepto de **concept drift**.

3.1. Clasificación

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.

El problema de la clasificación es uno de los problemas más ampliamente estudiado en ciencia de datos. Se basa en el entrenamiento y construcción de modelos predictivos en base a un conocimiento previo que viene dado en forma de datasets pre-etiquetados, una vez obtenido estos modelos, se deberá poder predecir la etiqueta o clase de una nueva muestra que se incluya al problema y de la cual no sabemos su clase de pertenencia. En minería de flujo de datos el problema es similar, salvo por que los datos nos llegan en un flujo continuo, de manera que no podemos tener todo el dataset a priori para aprender y validar con las consiguientes dificultades que esto aporta al problema.

Los clasificadores usados en los experimentos anteriores son dos, concretamente el **Hoeffding Tree** y el **Hoeffding Tree Adaptativo**.

- **Hoeffding Tree**: Es un algoritmo basado en árboles de decisión que trabaja de manera incremental de ahí su uso para trabajar con flujos de datos. Asume que la distribución de ejemplos no cambia en el tiempo y se basa en el principio matemático de Hoeffding, que nos dice el número de ejemplos que son estadísticamente necesarios para partir una rama del árbol en pos de mejorar o no empeorar las medidas de bondad. Acorde a este principio, los arboles explotan la idea de que un conjunto pequeño de muestras puede ser usada para crear un buen árbol de decisión y de ahí procede su potencia en determinados problemas como los de flujo de datos. Es un algoritmo bastante eficiente donde tanto el tiempo como la memoria para construir los modelos son lineales y no dependen del número de ejemplos en el stream.

- **Hoeffding Tree Adaptativo**: Este algoritmo es una versión del algoritmo anterior con idéntico funcionamiento pero que hace uso de ADWIN (Adaptative Window) para monitorizar los cambios de diferencia entre valores medios, con lo que detecta posibles descensos de accuracy en las ramas del árbol y las intercambia por mejores soluciones, acorde al accuracy, en la medida de lo posible por lo que a pesar de ser computacionalmente más complejo, los resultados serán en la mayoría de los casos mejores.

Si dejamos de lado los clasificadores y nos centramos en los métodos de evaluación usados encontramos tres diferentes:

1. **Evaluación offline**: Este método de evaluación es el tradicional en el que se usa un conjunto de datos para construir el modelo (train) y posteriormente se evalúa con otro conjunto de datos diferente al utilizado para entrenar. Es el típico proceso de evaluación de clasificadores utilizado para problemas fuera del ámbito del flujo de datos.

2. **Interleaved Test Then Train**: Este método es el más común en flujo de datos, se basa en utilizar cada nuevo dato del flujo para probar el modelo y posteriormente se añade este dato a la fase de entrenamiento. Este método concretamente se realiza de manera incremental es decir, cada nuevo dato que va llegando se va teniendo en cuenta desde el inicio hasta el fin, siendo esta la principal diferencia con el método prequential que ahora veremos.

3. **Prequential**: El método prequential en moa usa la misma filosofía de *test then train* pero usa una ventana deslizante cuyo valor podemos fijar. Esta ventana deslizante nace de la filosofía de que los datos nuevos son los mas relevantes por lo que el modelo de train se va construyendo bajo el umbral de esta ventana deslizante olvidando los datos mas antiguos.

3.2. Concept Drift

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

El ***concept drift***, o cambio de contexto es uno de los principales problemas a los que nos enfrentamos en la minería de flujo de datos y viene a significar que los datos que han llegado en el pasado difieren en mayor o menor medida en los datos que estamos trabajando en este mismo momento por lo que se entra en conflicto con asunciones pasadas sobre los datos y causará un descenso de las medidas de bondad que en algunos casos puede llegar a ser drástico. Detectar estos cambios de contexto apropiadamente y readaptar los modelos será por tanto un gran reto pero necesario en los problemas de minería de flujo de datos. Su importancia en estos problemas es tal que han propiciado una gran línea de investigación con trabajos muy recientes que tratan sobre soluciones en la materia [3] [4] [5] [6].

Como hemos introducido anteriormente, el ***concept drift*** se debe a cambios en los datos, pero estos no vendrán dados siempre de la misma manera sino que podrán presentarse de manera recurrente, gradual, incremental o brusco acorde a los ejemplos que podemos ver en la figura 3.1.

El cambio de contexto podrá deberse a diversos motivos como pueden ser, la variación de características, la presencia de ruido, aparición de nuevas características o la influencia del entorno, siendo este uno de los motivos más delicados. Igualmente, sea cual sea el motivo del cambio de contexto, encontramos los siguientes métodos u algoritmos para para solventar el problema:

- **Aprendizaje Online**: Estos algoritmos continuamente actualizan los parámetros de clasificación mientras el flujo de datos está activo. Hay que tener en cuenta que cada ejemplo solo debe ser procesado una vez,

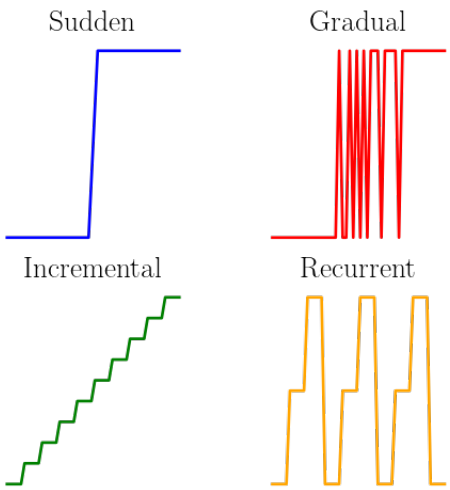

Figura 3.1: Tipos de concept drift.

su rendimiento no debería ser distinto a los algoritmos offline y y los requisitos de tiempo y procesado deben ser limitados. Un algoritmo muy común dentro de esta categoría sería el CVFDT [7].

- **Soluciones de ventana**: Son algoritmos que olvidan datos antiguos conforme llegan datos nuevos, asumiendo que estos últimos tienen más relevancia en el problema. Esta solución, hace que de cara a un cambio de contexto el algoritmo reaprenderá este cambio de contexto y aunque tendrá cierto descenso en sus medidas en algún momento temporal, se recuperará y volverá a ofrecer resultados similares a los ofrecidos antes del cambio de contexto. Las soluciones más afamadas son la ventana deslizante, ventana en función de hitos y la ponderación de datos en función de la antigüedad.

- **Ensembles**: Estos son algoritmos que entrenan diversos modelos de clasificadores elementales con ciertas variaciones de manera que la decisión final es una decisión colectiva entre todos ellos. Esta diversidad o variaciones para entrenar pueden deberse a las características, el modelo de clasificación o las instancias usadas para entrenar siendo estos conceptos muy relevantes en entornos cambiantes donde la diversidad

puede venir marcada por los nuevos datos del stream, incluido el hipotético cambio de contexto. Los algoritmos más famosos dentro de este área serían los de Street [8] y el de Wang [9].

- **Algoritmos de detección**: Estos algoritmos en lugar de buscar adaptabilidad buscan detectar cuando se producirá cambio de contexto para posteriormente paliar sus efectos. Para detectar un cambio de contexto se suele analizar la disminución de las medidas de bondad. En estos algoritmos encontramos el problema de cuando detectar el cambio de contexto, si esperamos mucho una vez iniciado el mismo estaremos ante un método que es robusto, pero que perderá accuracy al activar los procesos tarde. Por otro lado, hacen saltar la alarma rápidamente el algoritmo paliará los efectos cuando el cambio de contexto aún no haya perjudicado mucho el modelo, pero por contra se enfrenta al problema de un elevado número de falsos positivos.

Capítulo 4

Conclusión

Bibliografía

[1] Albert Bifet, Geoff Holmes, Richard Kirkby, Bernhard Pfahringer (2010); MOA: Massive Online Analysis; *Journal of Machine Learning Research* 11: 1601-1604

[2] Moa Web Site https://moa.cms.waikato.ac.nz

[3] Sunanda Gamage and Upeka Premaratne. 2017. Detecting and Adapting to Concept Drift in Continually Evolving Stochastic Processes. *Proceedings of the International Conference on Big Data and Internet of Thing (BDIOT2017)*. ACM, New York, NY, USA, 109-114

[4] Sylvio Barbon Junior, Gabriel Marques Tavares, Victor G. Turrisi da Costa, Paolo Ceravolo, and Ernesto Damiani. 2018. A Framework for Human-in-the-loop Monitoring of Concept-drift Detection in Event Log Stream. *Companion of the The Web Conference 2018 on The Web Conference 2018 (WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 319-326.

[5] Sun, Y., Wang, Z., Bai, Y., & Dai, H. A Classifier Graph based Recurring Concept Detection and Prediction Approach.

[6] Wang, Z., Tian, M., & Jia, C. (2017, December). An Active and Dynamic Botnet Detection Approach to Track Hidden Concept Drift. In *International Conference on Information and Communications Security* (pp. 646-660). Springer, Cham.

[7] Hulten, G., Spencer, L., & Domingos, P. (2001, August). Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD*

international conference on Knowledge discovery and data mining (pp. 97-106). ACM.

[8] Street, W. N., & Kim, Y. (2001, August). A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 377-382). ACM.

[9] Wang, H., Fan, W., Yu, P. S., & Han, J. (2003, August). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 226-235). AcM.