
UNIVERSIDAD
DE GRANADA

# SERIES TEMPORALES Y MINERIA DE FLUJO DE DATOS

## TRABAJO GUIADO: FLUJO DE DATOS

**AUTOR**

JOSÉ ÁNGEL DÍAZ GARCÍA
76139799R - joseadiazg02@correo.ugr.es

Abril del 2018

**EJERCICIO 1:** Se pide comparar la eficacia de un Hoeffding Tree con un clasificador Naïve Bayes, para un flujo de datos de 1.000.000 de instancias generadas con un generador RandomTreeGenerator, suponiendo una frecuencia de muestreo de 10.000 y con el método de evaluación Interleaved Test-Then-Train.

Para realizar esto lo primero que debemos hacer es crear un script de bash para automatizar la creación de los resultados.

```
#!/bin/bash

for i in `seq 1 30`;
    do
        eval "nombreNB=nb$i.txt"
        eval "nombreHT=ht$i.txt"
        java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask \
 "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s \
 (generators.RandomTreeGenerator -i $i) -i 1000000 -f 10000" >  $nombreNB

        java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask  \
"EvaluateInterleavedTestThenTrain -l trees.HoeffdingTree -s  \
(generators.RandomTreeGenerator -i $i) -i 1000000 -f 10000" >  $nombreHT
    done
```

Una vez tenemos todos los resultados, realizaremos una muestra con los valores de la columna **Classification Correct (Percent),** donde tenemos el porcentaje de acierto. Para ello cargaremos todos los datos en R y mostramos los resultados de la clasificación para cada clasificador.

```
> resultadosHT
 [1] 94.42556 94.54909 94.34374 94.68848 94.48616 94.34374 94.51162 94.45677
 [9] 94.49909 94.55919 94.50778 94.60788 94.51707 94.64354 94.75354 94.42677
[17] 94.57253 94.45414 94.46131 94.56172 94.48646 94.55838 94.62545 94.64667
[25] 94.63232 94.43051 94.49101 94.57152 94.48788 94.52343

> resultadosnb
 [1] 73.62707 73.76232 73.69616 73.66606 73.68919 73.73040 73.57303 73.67162
 [9] 73.75091 73.62283 73.56404 73.63566 73.70535 73.71505 73.63798 73.70848
[17] 73.66091 73.60202 73.73414 73.64869 73.61657 73.69333 73.65131 73.74232
[25] 73.69202 73.73374 73.71141 73.64727 73.64616 73.64556
```

Una vez tengamos estos datos, nos centramos en saber si las muestras son normales para ello, como tampoco tenemos muchas muestras usaremos el test de **Cramer-von Mises,** útil para ejemplos pequeños.  Como en ambos casos p value p >= Alfa (0.05) No se rechaza Ho es decir, los datos provienen de una distribución normal. Por lo que  podemos usar un test paramétrico para ver si hay diferencias. Usaremos el Test de Wilcoxon.

```
> wilcox.test(resultadosHT,resultadosnb, paired=T,conf.int = TRUE, exact = FALSE)

        Wilcoxon signed rank test with continuity correction

data:  resultadosHT and resultadosnb
V = 465, p-value = 1.825e-06
```

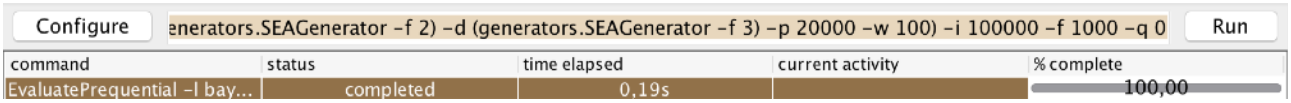
Con un p-value = 1.825e-06, el test de **Wilcoxon** nos dice que hay diferencias significativas por lo que el mejor de los algoritmos es: **Hoeffding Trees** esto es debido a que tenemos una muestra normal y diferencias significativas por lo que nos fijamos en la media de acierto ofreciendo este mejores resultados.

**EJERCICIO 2:** Se pide generar 100.000 instancias utilizando el generador de datos SEAGenerator, con un desvío de concepto centrado en la instancia 20.000 en una ventana de 100 instancias. Para simular el desvío de concepto, hacer que el simulador genere la función –f 2 al principio, y luego la función –f 3. Usar un clasificador Naïve Bayes evaluado con una frecuencia de muestreo de 1.000 instancias, usando el método prequential para evaluación. Inserte la configuración directamente en la GUI de MOA para visualizar la gráfica de la evolución de la tasa de aciertos (medida accuracy). ¿Qué se observa?

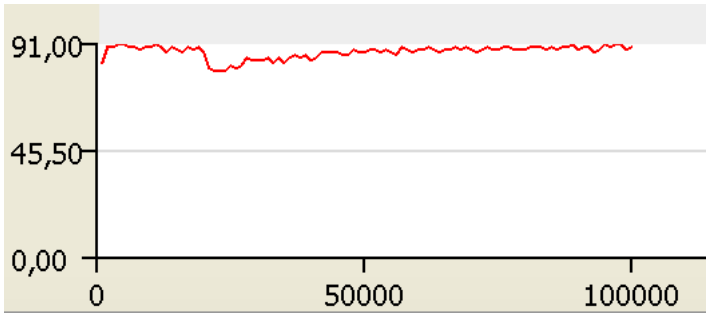
Para este ejercicio usaremos la GUI de MOA. Ejecutamos la misma e introducimos en cada una de las ventanas la configuración para la tarea que se nos pide:

```
EvaluatePrequential -l bayes.NaiveBayes -s (ConceptDriftStream -s
(generators.SEAGenerator -f 2) -d (generators.SEAGenerator -f 3) -p 20000 -w 100) -i
100000 -f 1000
```

Tras ejecutarlo y pulsar sobre run podemos ver que la tarea se ha ejecutado correctamente.



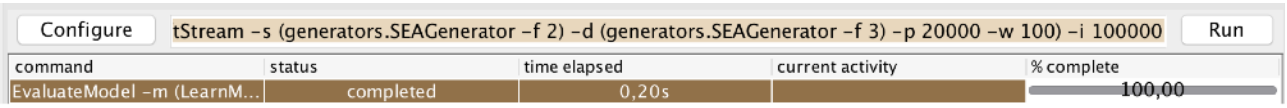

Tras esto, podremos observar el gráfico de accuracy, que sería el siguiente:




En el gráfico, podemos observar como en el momento del cambio de contexto, hay un descenso brusco del accuracy, pero al usar ventana deslizante para la validación, este comienza a recuperarse tras el cambio de contexto, hasta situarse nuevamente en valores similares a los que se dieron antes del cambio de contexto.

**EJERCICIO 3:** Entrenar un modelo estático Naïve Bayes sobre 100.000 instancias de la función 2 del generador SEAGenerator. Seguidamente, evaluarlo con un flujo de datos con desvío de concepto generado por el generador de datos SEAGenerator, con un desvío de concepto centrado en la instancia 20.000 en una ventana de 100 instancias. Para simular el desvío de concepto, hacer que el simulador genere la función –f 2 al principio, y luego la función –f 3.

Al igual que hicimos antes, configuramos nuestro experimento en la interfaz gráfica siguiendo las nociones que se nos dan en el ejercicio. Hecho esto, solo tenemos que ejecutar para ver los resultados.




```
classified instances = 100.000
classifications correct (percent) = 80,344
Kappa Statistic (percent) = 57,301
Kappa Temporal Statistic (percent) = 54,583
Kappa M Statistic (percent) = 39,098
model training instances = 100.000
model serialized size (bytes) = 0.0
```

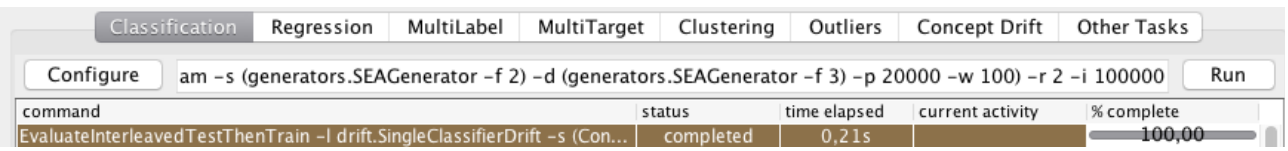
En los resultados vemos como las medidas de bondad de este experimento están muy por debajo de las obtenidas en el caso anterior, esto se debe a que como en ambos problemas hemos generado un cambio de contexto, el modelo dinámico se re entrena y se adapta al nuevo contexto mientras que el modelo estático no puede hacer tal cosa, por lo que ofrecerá peores resultados en evaluación.

**EJERCICIO 4:** ¿Qué ocurriría si pudiésemos detectar un cambio de concepto y re-entrenar un modelo estacionario? Se pide: Evaluar, y entrenar online con el método TestThenTrain, un modelo estacionario Naïve Bayes que se adapta (re-entrena) tras la detección de un cambio de concepto mediante el método DDM (función SingleClassifierDrift). Usar el flujo de datos del ejercicio anterior.

El resultado de detectar el cambio de concepto y re entrenar el modelo estacionario sería una conjunción de distintos modelos estacionarios entrenados al detectarse ese cambio de contexto.

En problemas de este tipo, nos encontramos ante la tesitura de la robusted de la detección del cambio de contexto, teniendo en cuenta que mayor tiempo en detectar, implicará más robusted a la hora de decidir si hay o no cambio de contexto. Menor tiempo de detección, se detectan antes pero estamos ante el problema de falsos positivos, donde daremos la “alarma” de cambio de concepto cuando quizá este no es tal.

El experimento en MOA sería el siguiente:



Una vez ejecutado, observamos valores de accuracy del 88,09, lo que nos indica a pensar que hemos tenido éxito en nuestros resultados, y que al detectar el cambio de contexto el modelo se ha re-entrenado y ofrecido mejores resultados. Aún así, comprobaremos estadísticamente que los cambios son reales y que no se deben al azar. Para ello ejecutaremos los experimentos 30 veces con nuestro script del ejercicio 1 con las tareas actuales:

```
#!/bin/bash

for i in `seq 1 30`;
    do
        eval "estatico=estatico$i.txt"
        eval "dinamico=dinamico$i.txt"
        java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask \
 "EvaluateModel -m (LearnModel -l bayes.NaiveBayes -s (generators.SEAGenerator -f 2 -i 2) -m 100000)
-s (ConceptDriftStream -s (generators.SEAGenerator -f 2) -d (generators.SEAGenerator -f 3 -i $i) -p
20000 -w 100) -i 100000" >  $estatico
        java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask \
  "EvaluateInterleavedTestThenTrain  -l  drift.SingleClassifierDrift  -s  (ConceptDriftStream  -s
(generators.SEAGenerator -f 2) -d (generators.SEAGenerator -f 3 -i $i) -p 20000 -w 100) -i 100000" >
$dinamico
    done
```

Una vez hecho esto y cargados los datos en R tenemos los siguientes vectores:

```
> resultadosestatico
 [1] 80.359 80.402 80.535 80.418 80.696 80.499 80.394 80.568 80.627 80.444 80.374 80.487 80.497
80.607 80.527 80.757 80.611
[18] 80.538 80.577 80.367 80.551 80.385 80.502 80.487 80.319 80.755 80.517 80.616 80.404 80.562
> resultadosdinamico
 [1] 88.086 88.147 87.962 88.108 88.311 87.828 88.188 87.880 87.813 88.127 88.023 87.919 87.997
88.041 87.808 88.117 87.993
[18] 88.060 88.157 87.939 88.360 88.196 88.036 87.973 88.182 88.171 87.991 88.210 88.118 87.812
```

Para comprobar la normalidad de los datos aplicamos, nuevamente al tener pocas muestras, el test de Cramer-von Mises.

```
> cvm.test(resultadosestatico[1:30])

        Cramer-von Mises normality test

data:  resultadosestatico[1:30]
W = 0.038622, p-value = 0.6955

> cvm.test(resultadosdinamico[1:30])

        Cramer-von Mises normality test

data:  resultadosdinamico[1:30]
W = 0.028461, p-value = 0.8614
```

Como ambos casos los p-vaues son > 0.05 (Alfa), no rechazamos la hipótesis 0, que nos indica que los datos provienen de una distribución normal. Por lo que podemos usar un test paramétrico para comprobar si hay diferencias significativas. Usaremos nuevamente el test de Wilcoxon:

```
> wilcox.test(resultadosestatico,resultadosdinamico, paired=T,conf.int = TRUE, exact = FALSE)

        Wilcoxon signed rank test with continuity correction

data:  resultadosestatico and resultadosdinamico
V = 0, p-value = 1.825e-06
```

Nuevamente con ese valor de p-value tan bajo, podemos concluir que hay diferencias significativas, y por tanto el modelo con detección del cambio de contexto, al ser el que mejores resultados ofrece es el mejor modelo.