Ejercicio Guiado - Mineria de Flujo de Datos.

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

18 de abril de 2019

1. Clasificación. Ejercicio 1.

Se pide comparar la eficacia de un Hoeffding Tree con un clasificador, 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 *Hoeffding Tree* usaremos la siguiente orden:

```
java -cp moa.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l trees.HoeffdingTree \
  -i 1000000 -f 10000"
```

Para *Naïve Bayes* usaremos la siguiente orden:

```
java -cp moa.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -i 1000000 \
  -f 10000"
```

Para saber si hay diferencias significativas, tendríamos que generar una población de resultados y escoger una medida de eficacia para comparar. Para ello, escogeremos 30 semillas diferentes y ejecutaremos 30 veces el mismo método (en total 30 ejecuciones para Naïve Bayes y otras 30 para Hoeffding Trees). Escogeremos los resultados del porcentaje de aciertos en la clasificación, y las compararemos con un test estadístico.

Para agilizar este proceso, he hecho un pequeño script qie ejecuta un orden 30 veces y guarda los resultados en los *csv* correspondientes. Dicho script es el siguiente:

```
1 for i in $(seq 30)
2 do
3   java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask "..." \
4   "-s (generators.RandomTreeGenerator -i $i)" > "$i.csv"
5 done
```

Sólo debemos cambiar los tres puntos de la línea 3 por la orden que queramos ejecutar. Primero lo haremos para Hoeffding Tree y luego Naïve Bayes. Una vez tenemos los resultados de las 30 ejecuciones de cada modelo, podemos aplicar los tests estadísticos. Para ello, he usado el siguiente código en R:

```
1  read.data <- function(folder) {
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 ultimo valor de
7    # la columna Classification Correct (Percent).
8    sapply(data, function(x) x[nrow(x), 5])
9  }
10
11 naive_bayes <- read.data(folder = "naive_bayes")
12 hoeffding_tree <- read.data(folder = "hoeffding_tree")
13
14 shapiro.test(naive_bayes)$p.value
15 shapiro.test(hoeffding_tree)$p.value
```

```
16 tseries::jarque.bera.test(naive_bayes)$p.value
17 tseries::jarque.bera.test(hoeffding_tree)$p.value
18 t.test(naive_bayes, hoeffding_tree)$p.value
19
20 mean(naive_bayes)
21 mean(hoeffding_tree)
```

Tras llamarlo, los resultados obtenidos son los siguientes:

Test	Resultado
p.value de shapiro para naive_bayes	0.6478472
p.value de shapiro para hoeffding_tree	0.8852122
p.value de jarque.bera para naive_bayes	0.6805229
p.value de jarque.bera para hoeffding_tree	0.8924791
p.value de t.test	1.206664e-101
media de naive_bayes	73.67307
media de hoeffding_tree	94.54905

Los resultados de los tests estadísticos (saphiro y jarquebera) indican que los datos siguen una distribución normal y que, por lo tanto, podemos aplicar un test paramétrico, como es el test de Student. Tras aplicarlo, podemos afirmar que hay diferencias significativas entre ambos algoritmos. Por eso, he calculado la media para ambos algoritmos y, efectivamente, Hoeffding tiene un *94.54905 %* de acierto mientras que Naïve Bayes sólo un *73.67307 %*.

2. Concept drift.

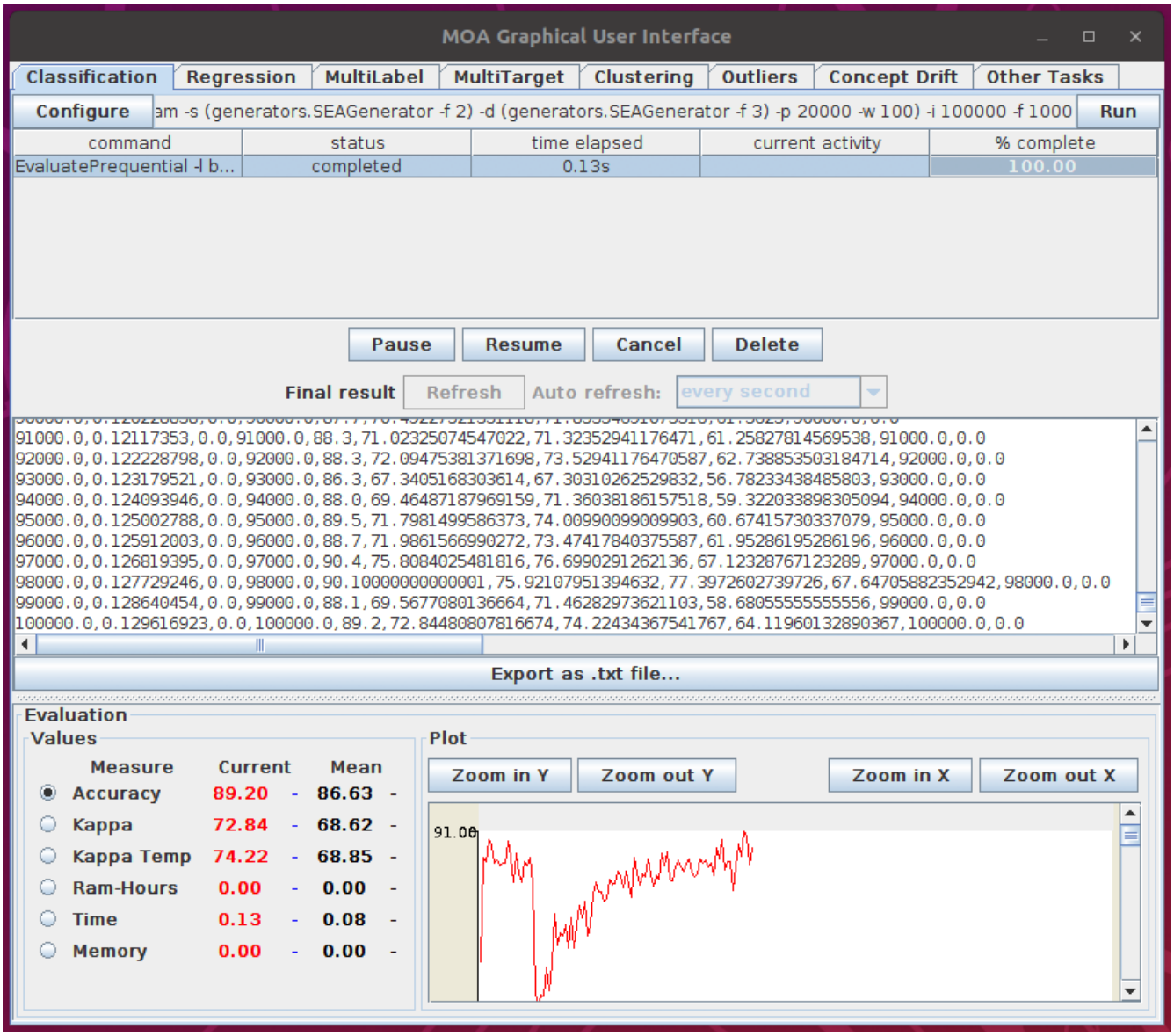
2.1. Ejercicio 2.

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 solucionar este ejercicio, el comando usado en MOA es el siguiente:

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

En la línea uno indicamos que vamos a usar Naïve como clasificador. En la línea dos indicamos que va a haber *Concept Drift*. En la línea tres indicamos que los datos se van a empezar a generar usando la función dos y en la línea tres indicamos que luego usaremos la función cuatro. En la línea cinco indicamos que el cambio va a estar centrado en la instancia 20000 con una ventana de 100 instancias. Finalmente, en la línea seis indicamos que se van a generar 100000 instancia con una frecuencia de muestreo de 1000. El resultado obtenido es el siguiente:



Podemos observar que, justo cuando se produce el cambio de concepto, la precisión del modelo cae drásticamente y que, conforme se va adaptando a los nuevos datos, esta precisión se recupera.

2.2. 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.

Para solucionar este ejercicio, el comando usado en MOA es el siguiente:

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

En la línea uno entrenamos el modelo asociado a la primera parte de la pregunta. A continuación, generamos el *Concept Drift*, al igual que hemos hecho en el ejercicio anterior. El resultado obtenido es el siguiente:

Métrica	Valor
classified instances	100000
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	100000
model serialized size (bytes)	0.0

Lo que sucede en este experimento es que estamos entrenando un modelo estacionario, es decir, en la primera parte de los datos, antes de que suceda el *Concept Drift*. Cuando este sucede, dato que el modelo no ha sido entrenado teniéndolo en cuenta, los resultados del experimento empeoran, tal y como podemos ver en las métricas *Kappa*.

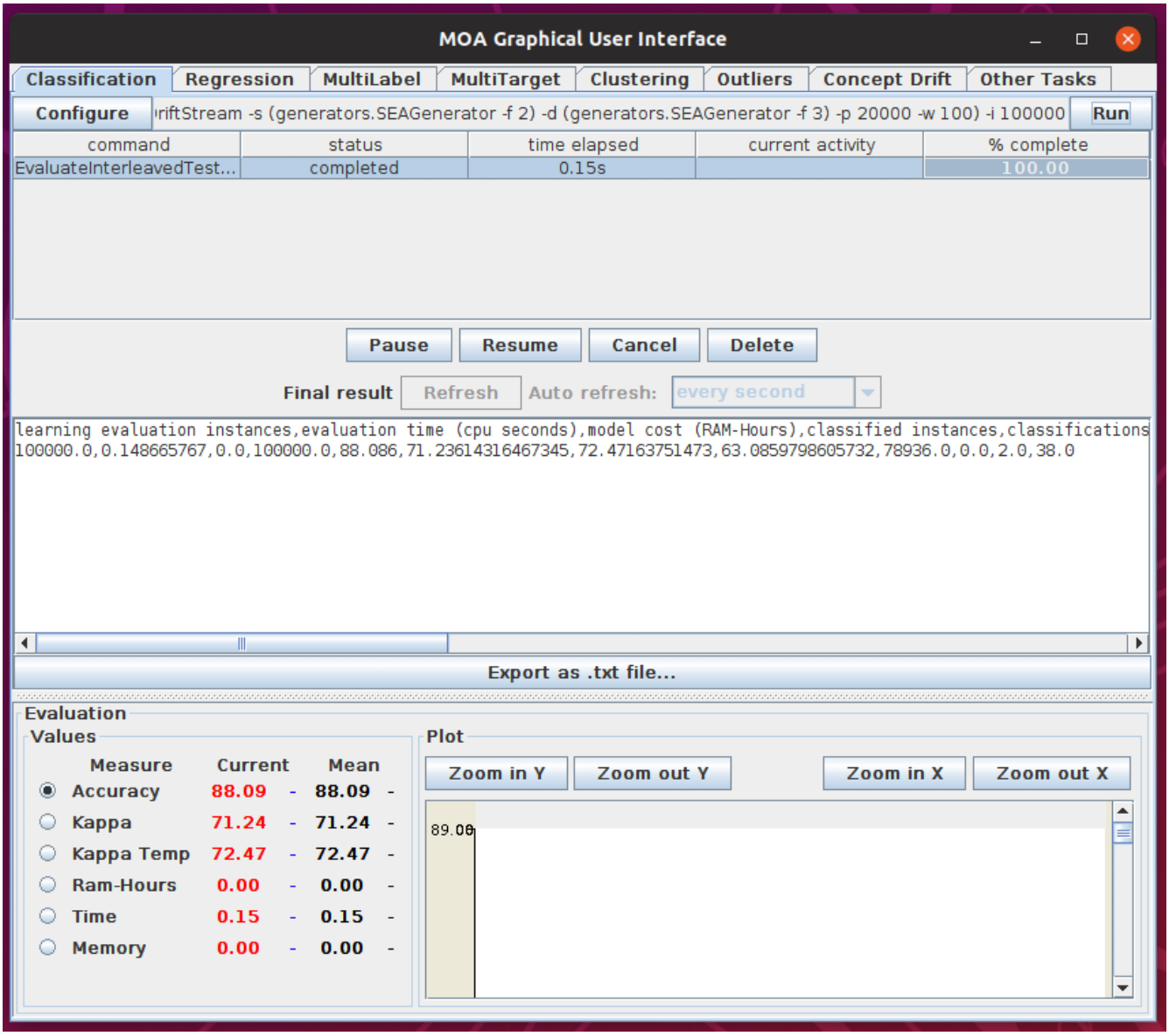
2.3. Ejercicio 4.

¿Qué ocurriría si pudiésemos detectar un cambio de concepto y re-entrenar un modelo estacionario? El resultado no sería un modelo "estacionario", sino múltiples de ellos entrenados tras detectar cambio de concepto. 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.

Para solucionar este ejercicio, el comando usado en MOA es el siguiente:

```
1 EvaluateInterleavedTestThenTrain
2   -l (moa.classifiers.drift.SingleClassifierDrift -l bayes.NaiveBayes -d DDM)
3   -s (ConceptDriftStream
4     -s (generators.SEAGenerator -f 2)
5     -d (generators.SEAGenerator -f 3)
6   -p 20000 -w 100)
7 -i 100000
```

El resultado obtenido es el siguiente:



Si construyésemos una población de resultados e hiciésemos tests estadísticos de comparación entre este modelo y el anterior:

- Vemos que el porcentaje de aciertos y el estadístico Kappa mejora sustancialmente.
- Es el resultado esperado: Tras un cambio de concepto, un modelo estacionario deja de funcionar. Se debe reentrenar para ser usado en el nuevo contexto.