

Series temporales y minería de flujo de datos: Trabajo guiado - Flujo de datos

Máster en ciencia de datos e ingeniería de computadores - UGR

16-4-2018

M^a Cristina Heredia Gómez

crstnheredia@correo.ugr.es

Índice

Ejercicio 1	3
Solución	3
Ejercicio 2	4
Solución	4
Ejercicio 3	6
Solución	6
Ejercicio 4	7
Solución	7

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.

Solución

Comprobamos en el manual cómo evaluar con Test-Then-Train cada uno de los clasificadores y generadores, y luego lo aplicamos para Hoeffding Trees y Naïve Bayes.

```
cris@cris ~/Downloads/moa-release-2017.06b/moa-release-2017.06b $ java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s generators.RandomTreeGenerator -i 1000000 -f 10000"
```

```
cris@cris ~/Downloads/moa-release-2017.06b/moa-release-2017.06b $ java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l trees.HoeffdingTree -s generators.RandomTreeGenerator -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 cada uno de los métodos, para lo que se ha creado el siguiente script en Bash:

Listing 1: Script bash para generación de resultados

```
#!/bin/bash

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

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

done
```

Tras lo que finalmente recogeremos los resultados del porcentaje de aciertos en la clasificación, presentes en el último valor de cada columna *Classification Correct (percent)*, creando a partir de ellos dos poblaciones de resultados distintas (una por cada algoritmo) y las compararemos con un test estadístico.

Para automatizar el proceso de obtener los porcentajes de clasificación mejores de cada fichero, automatizo el proceso con el siguiente script en R:

Listing 2: Script R para lectura de aciertos train y creación de ficheros de resultados

```
setwd("/home/cris/Downloads/moa-release-2017.06b/moa-release-2017.06b")

naiveBayes<-cbind(seq(1:30), rep(0, 30))
hoeffdingTrees<-cbind(seq(1:30), rep(0, 30))

5 for(i in seq(1:30)){
  dathf<-read.csv(paste0("ht", i, ".txt"))
  datnb<-read.csv(paste0("nb", i, ".txt"))
  classhf<-dathf[nrow(dathf), 5]
```

```

10  classnb<-datnb[nrow(datnb),5]
    hoeffdingTrees[i,2]<-classhf
    naiveBayes[i,2]<-classnb
  }

15  write.table(hoeffdingTrees,"HoeffdingResults.txt")
    write.table(naiveBayes,"NaiveBayesResults.txt")

```

Resultando en dos ficheros *HoeffdingResults.txt* y *NaiveBayesResults.txt* que contienen dos columnas cada uno: por un lado, la semilla utilizada y por otro, el porcentaje de acierto en train obtenido.

Desde R, cargamos los datos y les aplicamos los test de normalidad *Shapiro Test* y *Jarque Bera Test* para comprobar la normalidad de los datos:

```

# leer datos
hoeff<-read.table("HoeffdingResults.txt")
naive<-read.table("NaiveBayesResults.txt")

# comprobamos normalidad
# shapiro
shapiro.test(hoeff$V2) # >0.05
shapiro.test(naive$V2) # >0.05
library(tseries)
#jarque bera
jarque.bera.test(hoeff$V2) # >0.05
jarque.bera.test(naive$V2) # >0.05

```

Como en todos los casos y para ambos algoritmos obtenemos un p-valor $> 0,05$, podemos concluir que los datos siguen una distribución normal, y por tanto, podemos usar un test paramétrico para comprobar si sus diferencias son significativas. Usamos para ello un *t-test*:

```

# test de equivalencia
t.test(hoeff$V2,naive$V2) # p-value=2.2e-16 sí hay diferencia

```

Y comprobamos que obtenemos un p-valor $< 0,05$ y por tanto no podemos afirmar que no haya una diferencia significativa, es decir, que hay un algoritmo mejor que otro. Si comprobamos el acierto en media para ambos algoritmos:

```

# comprobar medias de acierto
mean(hoeff$V2)
mean(naive$V2)

```

Obtenemos para *Hoeffding Trees* un acierto medio del 94.54905 %, mientras que para *Naive Bayes* el acierto medio es de 73.67307 %.

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?

Solución

- la tarea es evaluar en prequential, sobre el modelo Naïve Bayes: especificamos la tarea con **Evaluate-Prequential** y el modelo desado con el parámetro **-l**

- generando 100.000 instancias con frecuencia de muestreo de 1.000: lo especificamos con los parámetros **-i** y **-f** respectivamente
- el flujo debe tener un desvío de concepto en la instancia 20.000: especificamos el cambio de concepto con **ConceptDriftStream**, especificando con los parámetros **-s,-d** los datos antes y después del drift. Con el parámetro **-p** especificamos la instancia donde se producirá el cambio
- con una ventana de 100: lo especificamos con el parámetro **-w**
- el flujo de datos antes del drift debe proceder de la función 2 de SEAGenerator, pero el de después de la función 3: lo especificamos con **generators.SEAGenerator -f 2** y **generators.SEAGenerator -f 3** respectivamente

Por tanto, el flujo de datos del concept drift quedaría como:

Listing 3: flujo de datos del cambio de concepto

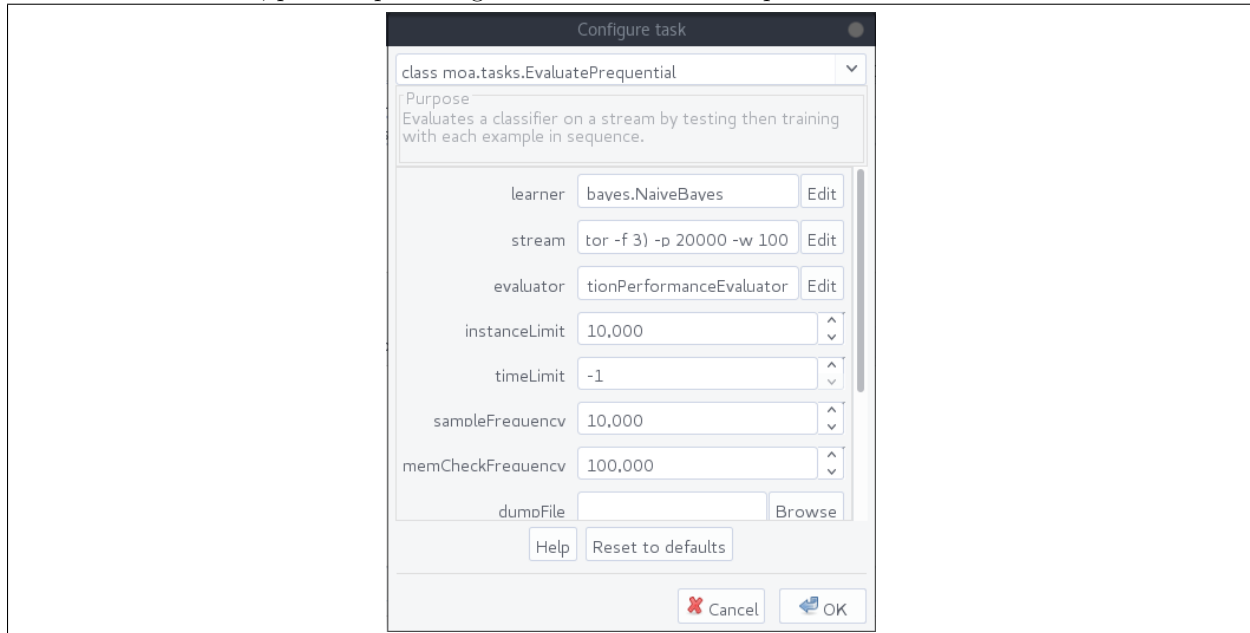
```
ConceptDriftStream -s "generators.SEAGenerator -f 2" -d
"generators.SEAGenerator -f 3" -p 20000 -w 100
```

y la tarea completa de evaluar en prequential sobre el modelo Naïve Bayes con el cambio de concepto dado, quedaría como:

Listing 4: sentencia para tarea completa

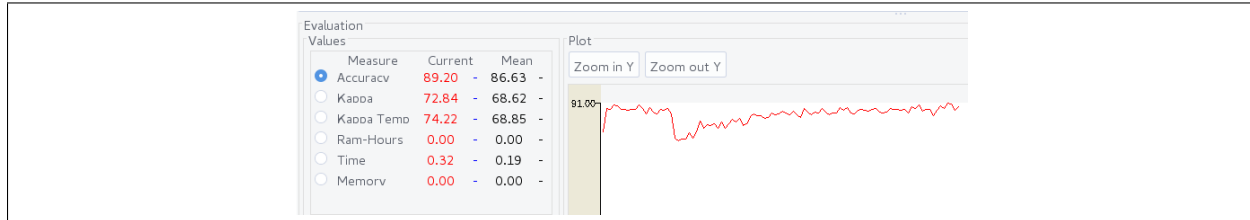
```
EvaluatePrequential -l bayes.NaiveBayes -s (ConceptDriftStream -s
(generators.SEAGenerator -f 2) -d (generators.SEAGenerator -f 3)
-p 20000 -w 100) -i 100000 -f 1000
```

Lo lanzamos en MOA, para lo que configuramos la tarea con los parámetros deseados.



Especificamos que el clasificador será Naive bayes, el flujo que se nos pide y ponemos el límite de instancias a 10.000. Dentro de las opciones de flujo, configuramos el concept drift tal y como se nos pide, de tal forma que en la opción stream configuramos que los datos antes del drift deben proceder de la función 2

de SEAGenerator, y en driftstream que procedan de la función 3. También ajustamos la instancia donde se produce el cambio (position) y el ancho (width):



Tras ejecutarlo, podemos observar en el gráfico una bajada en el acierto de clasificación cuando se produce el cambio de concepto, sin embargo, podemos ver que luego el sistema comienza a aprender de los nuevos datos y se recupera.

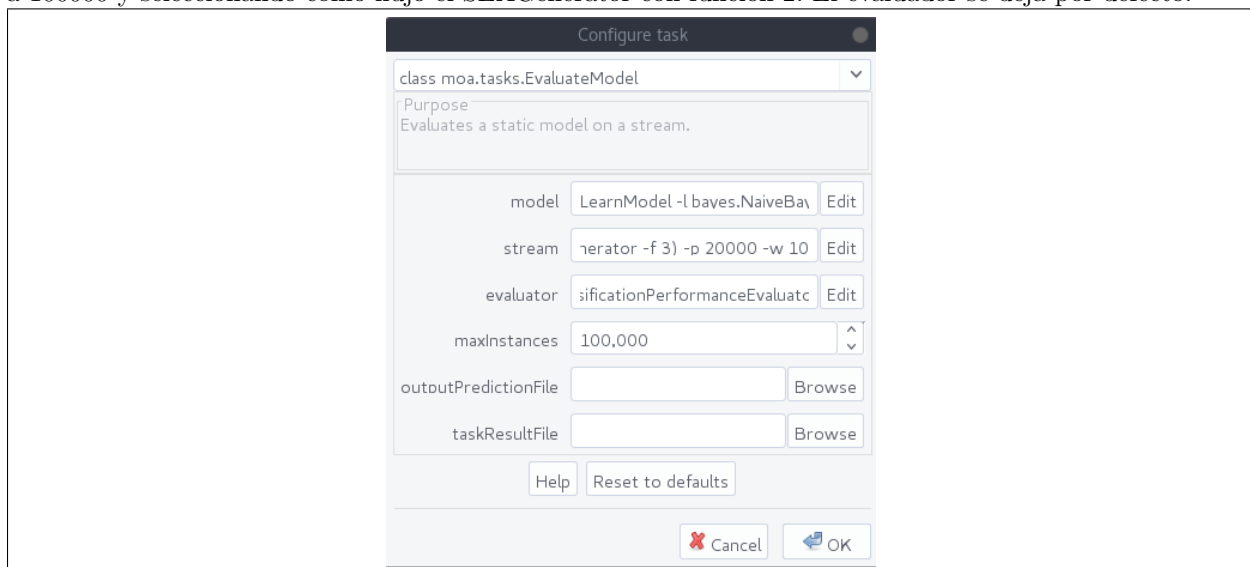
Ejercicio 3

Entrenar un modelo estático Naïve Bayes sobre 100.000 instancias de la función 2 del generador SEAGenerator. 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 $-f 3$.

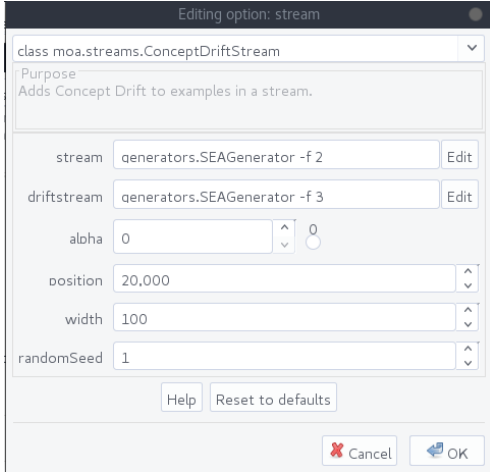
Solución

- Entrenar un modelo estático Naïve Bayes: lo especificamos con **LearnModel** y usamos la opción **-l** para especificar el modelo
- Sobre 100.000 instancias de la función 2 del generador SEAGenerator: especificamos el stream **SEAGenerator -f2** con la opción **-s** y usamos la opción **-m** para especificar el número de instancias
- Evaluar el modelo sobre 100.000 instancias de un flujo con concept drift: usamos **EvaluateModel -m** sobre lo anterior y la opción **-s** para especificar el flujo del ejercicio anterior y **-i** para especificar el número de instancias

Lo configuramos en MOA, seleccionando como modelo Naive Bayes y poniendo el parámetro **maxInstances** a 100000 y seleccionando como flujo el SEAGenerator con función 2. El evaluador se deja por defecto.



El concept drift lo configuramos como en el ejercicio anterior, dejando la instancia donde se provoca el concept drift, el tamaño de la ventana y el número de instancias al mismo valor.



```

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) = 1,936
  
```

En los resultados de la ejecución podemos ver que el 100 % de las instancias han sido clasificadas, si bien el 80.34 % han sido clasificadas correctamente, un porcentaje inferior al del ejercicio 2. También el valor Kappa es inferior en este caso al obtenido en el ejercicio anterior. Lo que sucede es, que a diferencia del modelo dinámico del ejercicio anterior, este modelo es estático, y ha sido entrenado con la función f2 del SEAGen, por tanto, falla cuando los datos cambian al ser generados por f3 cuando se produce el concept drift, evidenciando su dificultad para abordar un cambio de concepto.

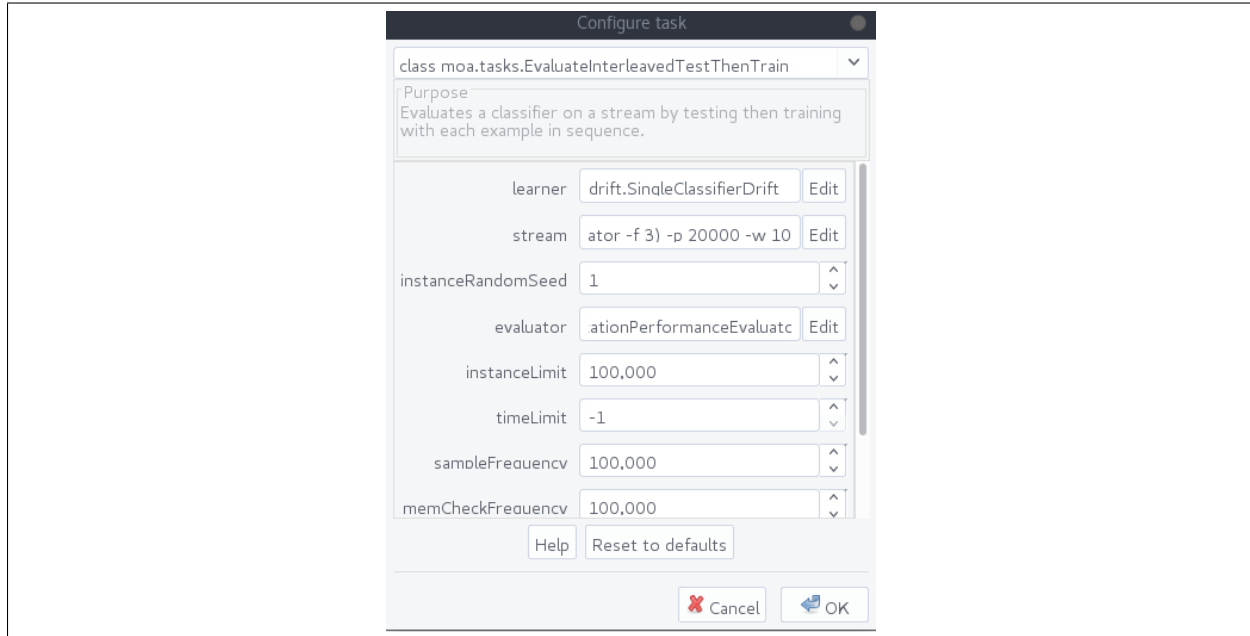
Ejercicio 4

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.

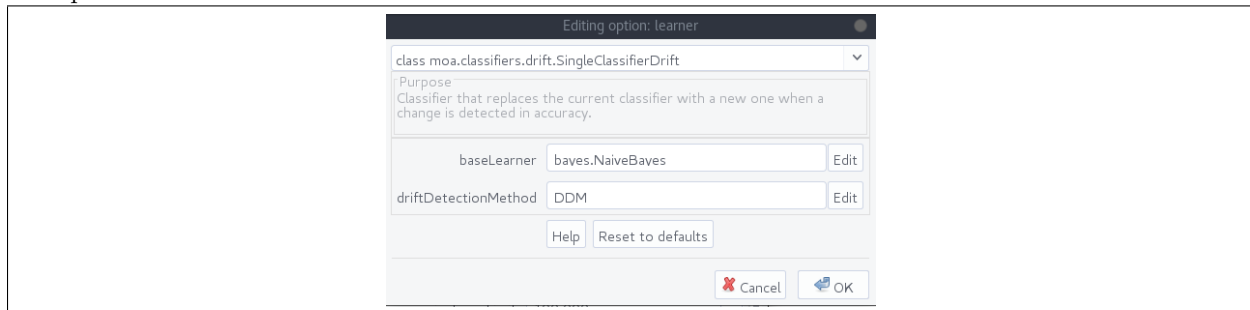
Solución

- Evaluar con **TestThenTrain** un modelo, en el flujo de datos del ejercicio anterior: con **EvaluateInterleavedTestThenTrain** especificamos que queremos evaluar con **TestThenTrain**, especificando con **-l** el modelo a evaluar y manteniendo el flujo del ejercicio anterior
- El modelo se obtiene tras reentrenar un Naïve Bayes cuando se detecta un cambio de concepto con DDM: para detectar un cambio de concepto con DDM se usa **SingleClassifierDrift**, al que se le especifica el modelo Naïve Bayes a usar

Lo configuramos con MOA. Para ello seleccionamos la tarea **EvaluateInterleavedTestThenTrain**, especificando como modelo **moa.classifiers.drift.SingleClassifierDrift**:



Dejando el flujo como el de la configuración para el ejercicio anterior, configuramos las opciones del learner, especificando que queremos que el clasificador base sea Naive Bayes pero que se use DDM para detectar el concept drift:



Acabando con un modelo con los siguientes resultados:

Evaluation			
Values			
Measure	Current	Mean	
<input checked="" type="radio"/> Accuracy	88.09	88.09	-
<input type="radio"/> Kappa	71.24	71.24	-
<input type="radio"/> Kappa Temp	72.47	72.47	-
<input type="radio"/> Ram-Hours	0.00	0.00	-
<input type="radio"/> Time	0.38	0.38	-
<input type="radio"/> Memory	0.00	0.00	-

Donde podemos apreciar un valor alto de acierto y de valor Kappa, lo cual indica un modelo que se está adaptando bien al cambio cuando se produce, en este caso porque se reentrena el modelo con la aparición de datos nuevos.