Minería de Flujos de Datos: práctica guiada

Carlos Manuel Sequí Sánchez

24 de abril de 2019

DNI: 20 48 69 26 K
e-mail: sequi96@correo.ugr.es

EJERCICIO 1 (clasificación). 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.

Una ejecución para Naïve Bayes:
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s generators.RandomTreeGenerator -i 1000000 -f 10000"

Una ejecución para Hoeffding Trees:
java -cp moa.jar -javaagent:sizeofag.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 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.

A continuación una imagen del script creado para generación de la población de 30 instancias de distintas semillas con Naïve Bayes (he procedido de la misma forma para la creación de la población de Hoeffding Trees)

```
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 1) -i 1000000 -f 10000" > nb1.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 2) -i 1000000 -f 10000" > nb2.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 3) -i 1000000 -f 10000" > nb3.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 4) -i 1000000 -f 10000" > nb4.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 5) -i 1000000 -f 10000" > nb5.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 6) -i 1000000 -f 10000" > nb6.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 7) -i 1000000 -f 10000" > nb7.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 8) -i 1000000 -f 10000" > nb8.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 9) -i 1000000 -f 10000" > nb9.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 10) -i 1000000 -f 10000" > nb10.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 11) -i 1000000 -f 10000" > nb11.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 12) -i 1000000 -f 10000" > nb12.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 13) -i 1000000 -f 10000" > nb13.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 14) -i 1000000 -f 10000" > nb14.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 15) -i 1000000 -f 10000" > nb15.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 16) -i 1000000 -f 10000" > nb16.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 17) -i 1000000 -f 10000" > nb17.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 18) -i 1000000 -f 10000" > nb18.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 19) -i 1000000 -f 10000" > nb19.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 20) -i 1000000 -f 10000" > nb20.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 21) -i 1000000 -f 10000" > nb21.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 22) -i 1000000 -f 10000" > nb22.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 23) -i 1000000 -f 10000" > nb23.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 24) -i 1000000 -f 10000" > nb24.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 25) -i 1000000 -f 10000" > nb25.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 26) -i 1000000 -f 10000" > nb26.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 27) -i 1000000 -f 10000" > nb27.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 28) -i 1000000 -f 10000" > nb28.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 29) -i 1000000 -f 10000" > nb29.csv
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l bayes.NaiveBayes -s (generators.RandomTreeGenerator -i 30) -i 1000000 -f 10000" > nb30.csv
```

Una vez generados los ficheros de datos, procedemos a crear las poblaciones en si

Leemos los datos de Naïves Bayes y creamos su población a partir de los fichers .csv

```
datosNaives = array(dim = 30)
for (i in 1:30)
{
```

```
  nombreFichero = paste(c(paste(c("./datosNaiveBayes/nb",i),collapse = ""),".csv"),collapse="")
  nuevoFichero = read.csv(nombreFichero)
  nuevoDato = nuevoFichero$classifications.correct..percent.[length(nuevoFichero$classifications.correct
  datosNaives[i] = nuevoDato
}
```

Leemos los datos de Hoeffding trees y creamos su población a partir de los fichers .csv

```
datosHoeffding = array(dim = 30)
for (i in 1:30)
{
  nombreFichero = paste(c(paste(c("./datosHoeffdingTrees/HF",i),collapse = ""),".csv"),collapse="")
  nuevoFichero = read.csv(nombreFichero)
  nuevoDato = nuevoFichero$classifications.correct..percent.[length(nuevoFichero$classifications.correct
  datosHoeffding[i] = nuevoDato
}
```

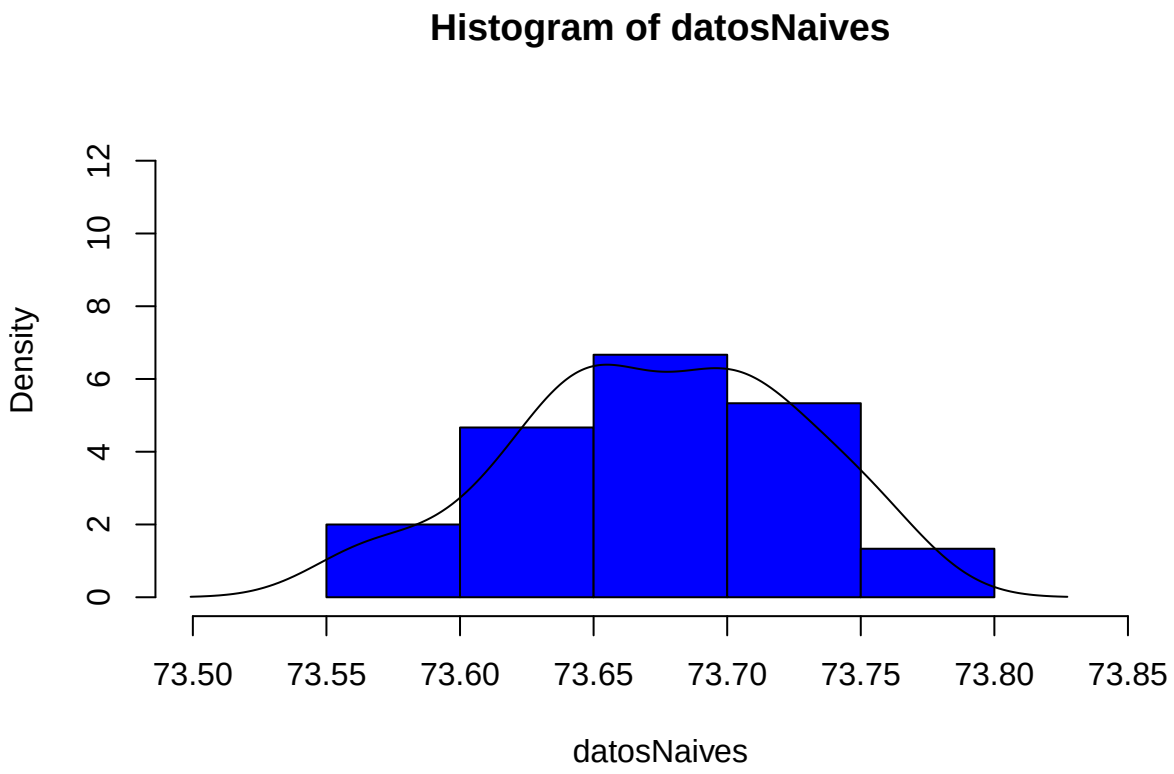
Una vez generadas las poblaciones, comprobamos la distribución de ambas para conocer que tipo de test utilizar.

Comprobamos si los datos de Naive Bayes siguen una distribución normal con el test de Saphiro Wilk y un histograma de los datos

```
shapiro.test(datosNaives)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  datosNaives
## W = 0.97381, p-value = 0.6478
```

```
hist(datosNaives, col="blue", prob=T,ylim=c(0,13), xlim=c(73.5,73.85))
lines(density(datosNaives))
```

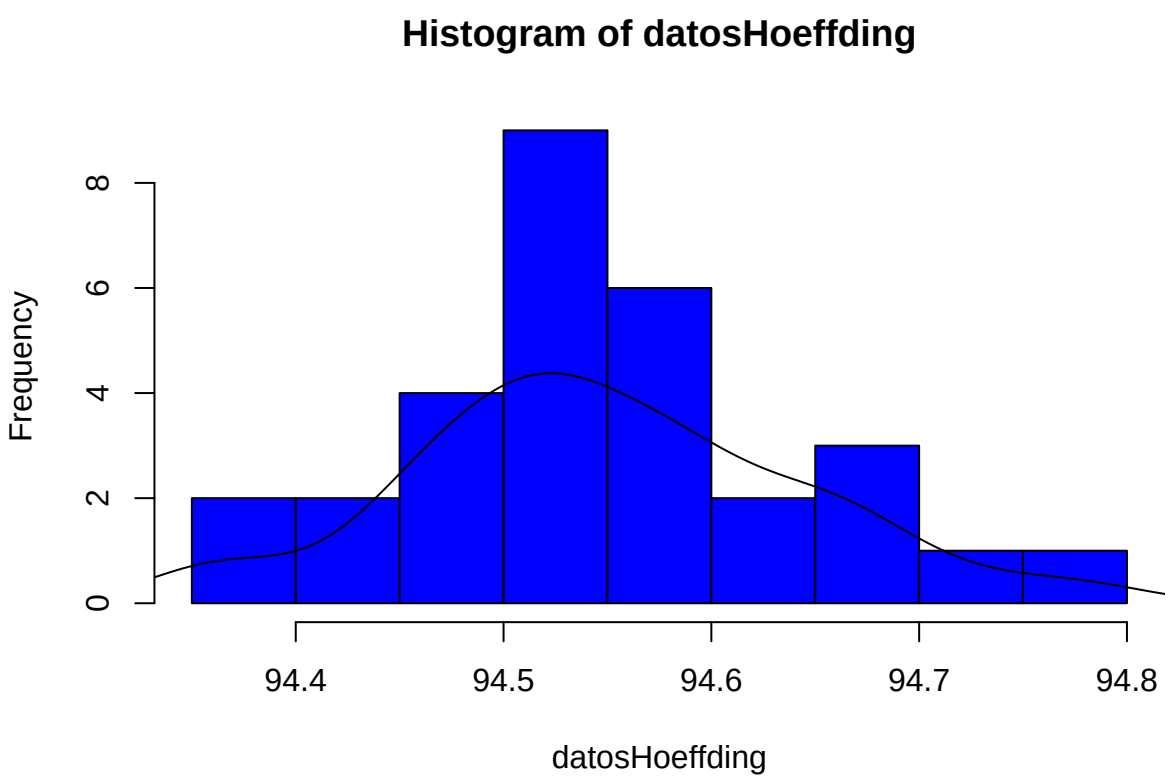


Comprobamos si los datos de Hoeffding siguen una distribución normal con el test de Saphiro Wilk y un histograma de los datos

```
shapiro.test(datosHoeffding)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  datosHoeffding
## W = 0.9824, p-value = 0.8852
```

```
hist(datosHoeffding, col="blue")
lines(density(datosHoeffding))
```



Como vemos ambos conjuntos de datos siguen una distribución normal. Debido a esto, vamos a utilizar un t-test con el fin de evidenciar si existen o no diferencias significativas entre las medias de ambos grupos.

```
t.test(datosNaives,datosHoeffding)
```

```
##
##  Welch Two Sample t-test
##
## data:  datosNaives and datosHoeffding
## t = -1055.4, df = 45.572, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -20.91581 -20.83615
## sample estimates:
## mean of x mean of y
##  73.67307  94.54905
```

Tal como observamos, al ser el p-value inferior a 0.05, podemos decir con alta certeza que las medias de las dos poblaciones de datos difiere de manera significativa. Por esta misma razón, el mejor de los algoritmos que han generado estos grupos de datos es el que mejor porcentaje de aciertos promedio ha proporcionado, es decir, el algoritmo Hoeffding Trees.

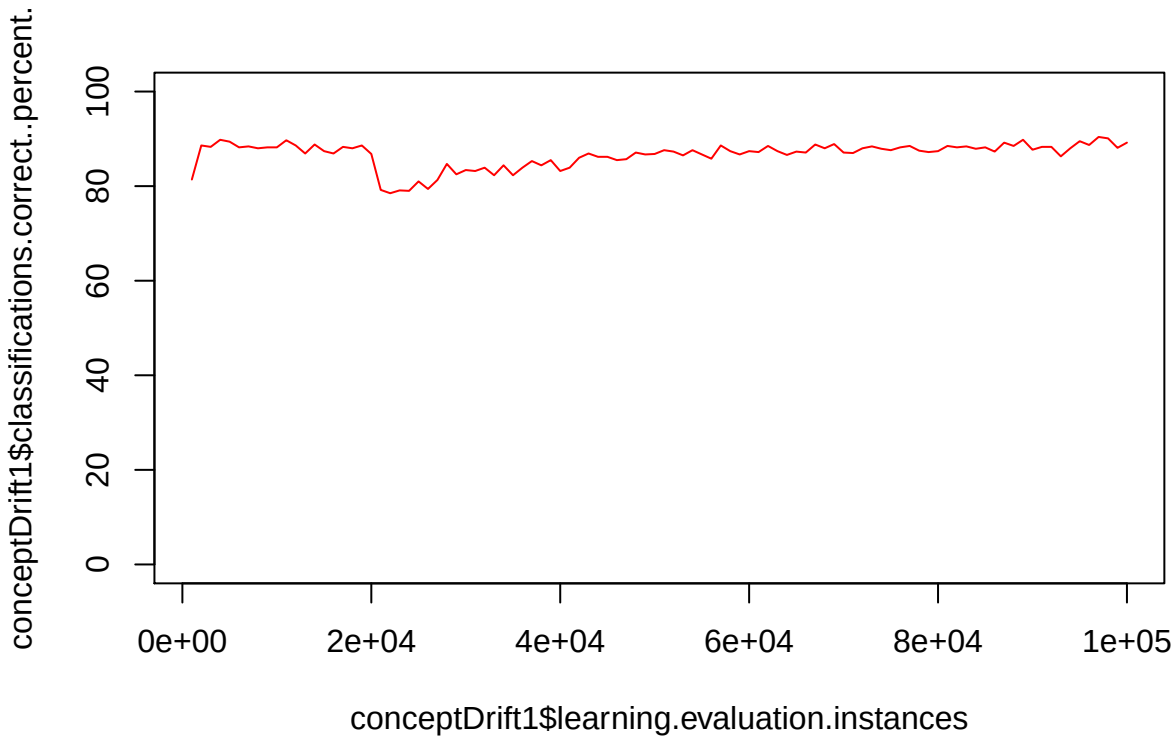
EJERCICIO 2 (Concept Drift).Se pide generar 100.000 instancias utilizando el generador de datos SEAGenerator, con un desvío de concepto centrado en la instancia 20.000 en una ventana de 100 instancias. Para simular el desvío de concepto, hacer que el simulador genere la función –f 2 al principio, y luego 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?

La tarea es evaluar en prequential, sobre el modelo Naïve Bayes, generando 100.000 instancias con frecuencia de muestreo de 1.000. **java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluatePrequential -l bayes.NaiveBayes -s (ConceptDriftStream -s (generators.SEAGenerator -f 2) -d (generators.SEAGenerator -f 3) -p 20000 -w 100) -i 100000 -f 1000" > conceptDrift1.csv**

La gráfica de la evolución de la tasa de clasificación de MOA:

```
conceptDrift1 = read.csv("./datosConceptDrift/conceptDrift1.csv")

# Mostrar el acierto del clasificador en el tiempo
plot(conceptDrift1$learning.evaluation.instances, conceptDrift1$classifications.correct..percent., "l",
```



Se observa que, tras el fallo por el desvio de concepto, posteriormente el sistema trata de recuperarse aprendiendo los nuevos datos. Con la precisión:

```
# mostramos los últimos valores de accuracy y de kappa
# junto con los valores medios.

# accuracy
conceptDrift1$classifications.correct..percent.[length(conceptDrift1$classifications.correct..percent.)]
```

```
## [1] 89.2
```

```
# accuracy mean
mean(conceptDrift1$classifications.correct..percent.)
```

```
## [1] 86.628
```

```
# kappa
conceptDrift1$Kappa.Statistic..percent.[length(conceptDrift1$Kappa.Statistic..percent.)]
```

```
## [1] 72.84481
```

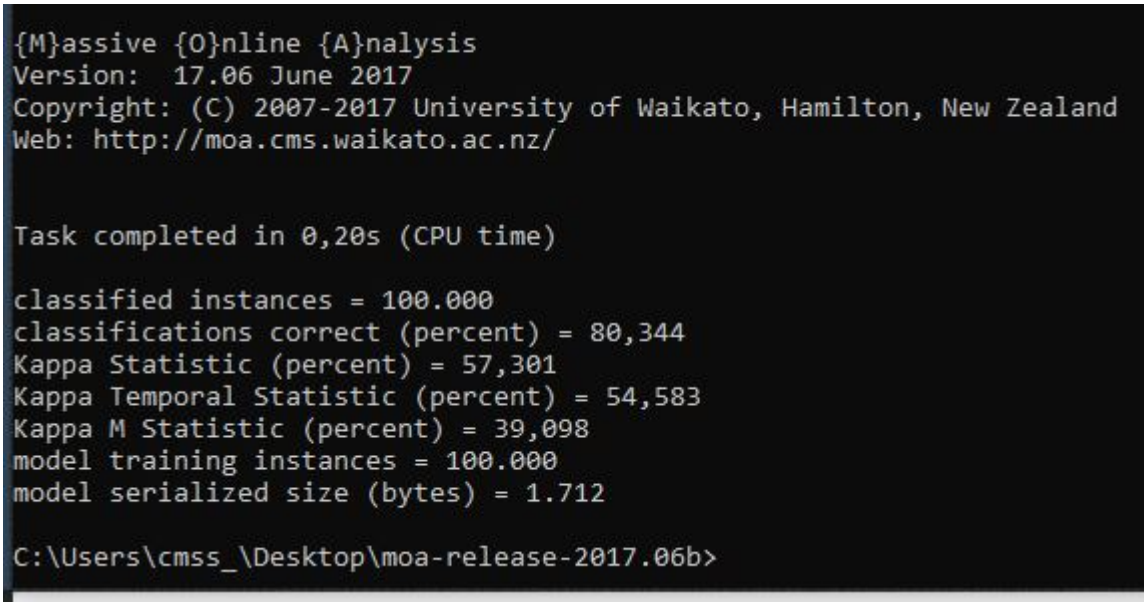
```
# kappa mean
mean(conceptDrift1$Kappa.Statistic..percent.)
```

```
## [1] 68.61815
```

EJERCICIO 3 (Concept Drift). 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.

A continuación el comando a utilizar para el cometido del enunciado:

java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "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"



¿Qué está pasando? Vemos que el porcentaje de clasificación, y también el estadístico Kappa, son inferiores en este segundo ejercicio en comparación con el primero.

Si hiciésemos múltiples ejecuciones y comparásemos las distribuciones de resultados, obtendríamos que hay diferencias significativas entre los dos métodos.

¿Qué ocurre?
El modelo estacionario se ha entrenado con la función f2 de SEA Generator. Falla con la función f3.
Al producirse el cambio de concepto, el modelo estacionario no se re-entrena. El modelo dinámico sí.
Esto provoca que el modelo dinámico vaya adaptándose con el tiempo a las nuevas condiciones de los datos y que, globalmente, al final proporcione una mejor tasa de aciertos.

EJERCICIO 4 (Concept Drift). ¿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.

A continuación el comando a utilizar para el cometido del enunciado:

java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask "EvaluateInterleavedTestThenTrain -l (moa.classifiers.drift.SingleClassifierDrift -l bayes.NaiveBayes -d DDM) -s (ConceptDriftStream -s (generators.SEAGenerator -f 2) -d (generators.SEAGenerator -f 3) -p 20000 -w 100) -i 100000" > conceptDrift2.csv

Leemos los datos generados...

```
conceptDrift2 = read.csv("./datosConceptDrift/conceptDrift2.csv")
```

```
# mostramos los últimos valores de accuracy y de kappa
# junto con los valores medios.

# accuracy
conceptDrift2$classifications.correct..percent.[length(conceptDrift2$classifications.correct..percent.)]
```

```
## [1] 88.086
```

```
# accuracy mean
mean(conceptDrift2$classifications.correct..percent.)
```

```
## [1] 88.086
```

```
# kappa
conceptDrift2$Kappa.Statistic..percent.[length(conceptDrift2$Kappa.Statistic..percent.)]
```

```
## [1] 71.23614
```

```
# kappa mean
mean(conceptDrift2$Kappa.Statistic..percent.)
```

```
## [1] 71.23614
```

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.