Series temporales y minería de flujo de datos: Trabajo autónomo II-Minería de Flujo de Datos

Máster en ciencia de datos e ingeniería de computadores - UGR

26-4-2018

Mª Cristina Heredia Gómez

crstnheredia@correo.ugr.es

Índice

Parte teórica

El problema de la clasificación

La clasificación es el problema de determinar la clase a la que pertenece cada muestra de unos datos. Cuando los datos son estáticos, generalmente se dispone de un histórico de datos almacenados sobre los que se entrena y valida un modelo como procedimiento habitual.
En flujo de datos, no se dispone de datos almacenados estáticos, sino que éstos se van recibiendo continuamente (hay un flujo continuo de datos), que se puede suponer infinito y ordenado. Son, por lo general, grandes cantidades de datos que llegan con alta tasa, que no poseen una distribución fija y cambian a lo largo del tiempo. Por tanto, la clasificación en flujos de datos consiste en manejar toda esa información de forma que se extraiga el conocimiento de forma dinámica de esos datos conforme se reciben, determinando la clase a la que pertenecen mediante algoritmos capaces de reaccionar o adaptarse a cambios en los datos.

Clasificadores utilizados en los experimentos

Hoeffding Tree

Tienen diferentes variantes que constituyen una familia de algoritmos capaces de construir árboles de decisión a partir de datos que llegan incrementalmente de flujos de datos. Están basados en la desigualdad de Hoeffding:
Sean $z_1, ..., z_n$ variables aleatorias independientes fronterizas, tal que $z_i \in [a_i, b_i]$ con probabilidad 1. Sea su suma $S_N = \sum_{i=1}^{N} z_i$, entonces pasa cualquier $\varepsilon > 0$ se tiene que:

$$P\{|S_N - E[\mathbf{S}_N]| > \varepsilon\} \leq \exp\{-2\varepsilon^2 / \sum_{i=1}^{N}(b_i - a_i)^2\}$$

que proporciona una cota superior a la probabilidad de que la suma de variables aleatorias se desvíe una cierta cantidad de su valor esperado.
Los Hoeffding Tree asumen que la distribución de los datos no cambia con el tiempo, y van construyendo un árbol de decisión incrementalmente, según el procedimiento siguiente:

- cada nodo contiene una prueba que separa los ejemplos en varias ramas, según los valores de los atributos
- Un nodo será expandido cuando haya evidencia estadística de que existe una característica por la que dividir de forma óptima, por lo tanto, la decisión de cuando y cómo dividir el árbol es muy importante
- Un criterio común para discriminar es la ganancia de información o el ratio de ganancia, que minimizan la entropía de las probabilidades de las diferentes hojas generadas, según como se separe el nodo
- La ganancia de información estimada para datos incrementales se puede calcular usando la cota de Hoeffding:

  $$\varepsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}$$

  que mide con confianza $(1 - \delta)$ como la media estimada de una variable aleatoria de rango R, $\hat{\mu}$, estimada después de n observaciones independientes se acerca a la real.

Hoeffding Tree Adaptativo

A diferencia del Hoeffding tree no adaptativo, el hoeffding tree adaptativo no asume que la distribución de los datos no cambia con el tiempo, de hecho incluye mecanismos para detectar cambios de concepto, monitorizando las ramas del árbol y reemplazando aquellas en las que decaiga la precisión en clasificación por ramas nuevas que proporcionen mayor acierto. Ésto explica porqué en la parque práctica se obtenían mejores resultados cuando había cambios de concepto con un hoeffding tree adaptativo frente al hoeffding tree.
En MOA, el Hoeffding tree adaptativo usa el método ADWIN para detectar el cambio de concepto y adaptar la longitud de ventana al mismo. El funcionamiento de ADWIN es el siguiente:

- Se inicializa W
- Se añade la nueva muestra entrante a la cabeza de W
- Si no se produce ningún cambio de concepto en ninguna de las subventanas de W, eliminar los elementos últimos de la ventana W

donde para detectar si se ha producido o no un cambio de concepto, se comprueba si $|\mu_{\hat{W}_0} - \mu_{\hat{W}_1}| \geq \varepsilon_{cut}$, donde $\varepsilon_{cut} = \sqrt{\frac{2\lambda}{m} \cdot ln\frac{2}{\delta}}$, donde m es el tamaño de la ventana y λ es la media de los datos de la ventana. Es decir, dada una ventana W, si existen $W_0, W_1 \in W$ que sean subventanas sufientemente grandes y con medias suficientemente diferentes, podemos decir que los valores esperados son distintos y por tanto se pueden eliminar los elementos antiguos de la ventana.

Modos de evaluación/validación en flujos de datos

Los métodos de evaluación/validación usados en la práctica se describen en las siguientes subsecciones.

EvaluateModel

Con la opción **-m** se le indica el modelo que queremos evaluar. En concreto, evalúa un modelo estático ya entrenado sobre otro flujo de datos para predecir las clases. Un modelo estático se dice cuando ha sido aprendido sobre datos estáticos, es decir, sobre un flujo de datos que, una vez generado, no cambia con el tiempo.

EvaluateInterleavedTestThenTrain

Evalúa un clasificador sobre un flujo de datos primero testeando y luego entrenando con cada muestra del flujo de datos. A diferencia del **EvaluatePrequential**, suporta un evaluador sencillo que mide el acierto del clasificador desde que comienza la evaluación.

EvaluatePrequential

Evalúa un clasificador sobre un flujo de datos, haciendo test-then-train sobre cada instancia. A diferencia del **EvaluateInterleavedTestThenTrain**, en este método de evaluación podemos usar una ventana deslizante y/o un factor de olvido para calcular el acierto en clasificación basándose en instancias recientes.
El factor de olvido se usa de la siguiente forma:

$$E_i = \frac{S_i}{B_i}$$

con

$$S_i = L_i + \alpha \cdot S_{i-1}$$

$$B_i = n_i + \alpha \cdot B_{i-1}$$

donde n_i denota el número de ejemplos usados para calcular la función de pérdida L_i. $n_i = 1$ dado que L_i se calcula para cada muestra.

Concept drift

Definición del problema

El concept drift o cambio de concepto, es un problema común en flujo datos que consiste en que se produce un cambio en los datos, ya sea de forma abrupta (cambio muy rápido), gradual (cambio generado con una especie de transición, por mezcla de distribuciones de datos) o incremental (cambio muy lento, donde normalmente la diferencia entre los datos llegados y los inediatamente posteriores no es significativa), provocando que el modelo que explicaba los datos hasta ahora ya no sea válido para explicar los nuevos datos. Este cambio en los datos puede deberse a ruido, influencia del entorno, estacionalidad o variación de características. Las principales formas de abordarlo pasan por:

- reentrenar el modelo cada vez que llega una nueva instancia (coste demasiado elevado)
- detectar cuando se produce un cambio en los datos y reentrenar el modelo solamente cuando se detecte que ese cambio producido es significativo
- usar aprendizaje adaptativo, tal que el modelo sea capaz de detectar cuando hay un cambio de concepto y adaptarse a él

Técnicas para resolver el concept drift

En concreto, éste es un campo muy activo en investigación donde los algoritmos propuestos se dividen principalmente en:

1. Aprendizaje online: estos algoritmos actualizan de forma continua los parámetros del clasificador conforme llegan nuevos datos. Cumplen una serie de requisitos, como que cada elemento se procesa una única vez en la etapa de entrenamiento, tiene limitada la cantidad de recursos de memoria y procesamiento, el aprendizaje se puede pausar en cualquier momento y proporciona mejores resultados que un clasificador offline. Un ejemplo de algoritmo de este tipo es el Hoeffding tree adaptativo.

2. Uso de ventanas (soluciones basadas en instancias): al igual que los algoritmos que incluyen un factor de olvido, estos algoritmos incorporan un mecanismo para olvidar instancias viejas, en concreto, una ventana deslizante que da mayor peso a las instancias más recientes en el tiempo dado que contienen mayor información del contexto actual.

3. Ensembles (aprendizaje de múltiples modelos): estos algoritmos se basan en la combinación de multiples clasificadores elementales. Esto es especialmente útil si se dispone de clasificadores suficientes en número y diversidad, pues en ocasiones, su combinación da muy buenos resultados. SEA (Streaming Ensemble Algorithm) y AWE (Accuracy Weighted Ensemble) son los algoritmos más populares dentro de los ensembles para flujos de datos. La principal diferencia entre ambos consiste en que SEA usa voto por mayoría y AWE usa un voto ponderado. Entre las similitudes están que ambos mantienen un número prefijado de clasificadores, usan los datos de nueva llegada para entrenar nuevos clasificadores y los clasificadores que proporcionan peor acierto de clasificación son sustituidos por otros nuevos más precisos.

4. Algoritmos de detección del desvío: detecta cuando se produce el concept drift y se actúan en consecuencia, generalmente reseteando parte o todo el conocimiento del algoritmo hasta la fecha. Por lo general, se asume que un cambio en el redimiento del modelo clasificando datos es indciativo de que los datos están cambiando, y de que por lo tanto hay que actuar en concecuencia. En el ejercicio 5 de la parte práctica, usamos un clasificador simple (SingleDriftClassifier) que sustituía el actual, que era un Hoeffding tree cuando se detectaba un cambio de concepto y usaba DDM como detector de cambio de concepto.
   DDM (Drift detection method) controla la tasa de error del modelo cuando clasifica los datos, para lo que usa dos ventanas, una que contiene todos los datos y otra que contiene datos desde el principio hasta que decrece el acierto de clasificación (aumenta el error). Un incremento significativo en el error del modelo denota que la distribución de los datos está cambiando, por lo que hay que actuar en consecuencia con el modelo. Para establecer exactamente cuando está sucediendo un concept drift, se usa un nivel de warning y un nivel de drift. Como se asume que los errores vienen dados por una distribución binomial, para cada punto i se calcula su desviación estándar como:

$$s(i) = \sqrt{p_i(1-p_i)/i}$$

   donde p_i es la probabilidad de clasificar erróneamente i. Entonces se define el nivel de warning como:

$$p_i + s_i \geq p_{min} + 2 \cdot s_{min}$$

   donde p_{min} es el error mínimo y s_{min} la desviación mínima. Por debajo de este nivel, se empiezan a guardar las muestras como anticipación a un posible cambio de concepto.
   Por otro lado, se define el nivel de drift como:

$$p_i + s_i \geq p_{min} + 3 \cdot s_{min}$$

   donde, por debajo de este nivel, se entiende que está habiendo un cambio de concepto y entonces el modelo se resetea y se aprende un nuevo modelo usando las instancias guardadas cuando se activó el nivel de warning.

Parte práctica

Entrenamiento offline (estacionario) y evaluación posterior

Ejercicio 1

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.

Solución

Creamos un script en bash que nos ejecute la sentencia 20 veces con diferentes semillas (del 1 al 20), para generar el conjunto de entrenamiento, tal y como se pide:

Listing 1: Script Bash para ejecución de la sentencia 20 veces con diferentes semillas

```
#!/bin/bash

for i in `seq 1 20`;
do
   java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask \
   "EvaluateModel -m (LearnModel -l trees.HoeffdingTree -s \
   (generators.WaveformGenerator -i $i) -m 1000000) -s \
   (generators.WaveformGenerator -i 4)" > ./Ej1Res/Apartado1ejercicio1semilla$i.txt
done
```

Donde con **LearnModel -l trees.HoeffdingTree** estamos especificando que queremos entrenar un clasificador HoeffdingTree sin evaluación integrada, es decir, queremos entrenar el modelo en principio, únicamente, aunque lo evaluemos después. Con **-s (generators.WaveformGenerator -i $i) -m 1000000)** estamos indicando que queremos entrenar el modelo sobre un flujo de datos generado con el generador indicado (WaveformGenerator), con **-i** indicamos la semilla para dicho generador, y con **-m** indicamos el máximo de instancias generadas.
Tras el entrenamiento del modelo, especificamos que queremos hacer una evaluación del modelo con **EvaluateModel -m**, dejando el modelo de evaluación al que viene por defecto, que es el **BasicClassificationPerformanceEvaluator**. También dejamos el máximo de instancias generadas por defecto, que es 1.000.000. Por último, con la opción **-s (generators.WaveformGenerator -i 4)** indicamos que queremos generar los datos de test a partir de un flujo **WaveformGenerator** con semilla de valor 4. Tras esto, redirigimos la salida a un fichero nuevo de texto.

Con el script anterior, replicamos el proceso 20 veces considerando diferentes semillas para generar el conjunto de entrenamiento. Una vez generados los resultados, leemos los distintos ficheros y creamos una tabla que guarde los resultados de nuestro interés: la tasa de bien clasificados y el valor Kappa. Para ello, usamos el siguiente Script de R creado para esa tarea:

Listing 2: Script en R para generación de tabla con resultados de interés

```
setwd("/home/cris/Downloads/moa-release-2017.06b/moa-release-2017.06b/Ej1Res")

# creamos tabla a rellenar con sus nombres en las columnas
data<-cbind(seq(1:20),rep(0,20),rep(0,20))
colnames(data)<-c("seed","acc","kappa")

# rellenamos tabla a partir de los datos de los ficheros
for(i in seq(1:20)){
  dat<-read.csv(paste0("Apartado1ejercicio1semilla",i,".txt"),sep="=")
  data[i,2]<-as.numeric(levels(dat[,2])[dat[1,2]])
  data[i,3]<-as.numeric(levels(dat[,2])[dat[2,2]])
}
# escribimos los resultados a un fichero nuevo
write.table(data,"Apartado1Ej1ACCKappa_Res.txt")
```

Por lo que acabamos teniendo un fichero con tres columnas: el valor de la semilla, el porcentaje de acierto de clasificación y el valor Kappa. Los resultados se muestran en la siguiente tabla:

Semilla	Acc (%)	Kappa (%)
1	84.509	76.765
2	84.512	76.77
3	84.59	76.887
4	84.666	77.001
5	84.481	76.723
6	84.342	76.514
7	84.799	77.2
8	84.153	76.231
9	84.641	76.963
10	84.578	76.869
11	84.539	76.81
12	84.457	76.688
13	84.369	76.555
14	84.547	76.822
15	84.648	76.974
16	84.626	76.94
17	84.513	76.772
18	84.434	76.653
19	84.605	76.91
20	84.568	76.853

Ejercicio 2

Repetir el paso anterior, sustituyendo el clasificador por HoeffdingTree adaptativo.

Solución

Ejecutamos el script anterior, esta vez especificando que queremos entrenar un Hoeffding tree adaptativo, con la opción -l en **LearnModel -l trees.HoeffdingAdaptiveTree**. Los demás parámetros se mantienen, cambiando solo los nombres de los ficheros de salida:

Listing 3: Script Bash para ejecución de la sentencia 20 veces con diferentes semillas

```
#!/bin/bash

for i in `seq 1 20`;
do
   java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask \
   "EvaluateModel -m (LearnModel -l trees.HoeffdingAdaptiveTree -s \
   (generators.WaveformGenerator -i $i) -m 1000000) -s \
   (generators.WaveformGenerator -i 4)" > ./Ej1Res/Apartado1ejercicio2semilla$i.txt
done
```

Cuando se han generado los 20 resultados, los agrupamos usando el script de R descrito anteriormente, especificando que se lean los datos con nombre **Apartado1ejercicio2semilla[i].txt** y cree un fichero de nombre **Apartado1Ej2ACCKappa_Res.txt** obteniendo los siguientes resultados para cada semilla probada en la generación de datos de entrenamiento:

Semilla	Acc (%)	Kappa (%)
1	84.521	76.783
2	84.474	76.712
3	84.416	76.625
4	84.465	76.699
5	84.262	76.395
6	84.368	76.554
7	84.271	76.408
8	84.243	76.367
9	84.478	76.719
10	84.326	76.491
11	84.371	76.558
12	84.416	76.627
13	84.498	76.749
14	84.415	76.624
15	84.229	76.345
16	84.328	76.494
17	84.358	76.539
18	84.456	76.685
19	84.451	76.679
20	84.459	76.69

Ejercicio 3

Responda a la pregunta: ¿Cree que algún clasificador es significativamente mejor que el otro en este tipo de problemas? Razone su respuesta.

Solución

Aunque a simple vista no se ven diferencias significativas en el porcentaje de acierto en clasificación ni en el porcentaje Kappa, lo contrastamos ocn test estadísticos. Para ello, el primer paso será comprobar si los datos siguen o no una distribución normal:

Listing 4: Script R para comprobación de normalidad en los datos

```
setwd("/home/cris/Downloads/moa-release-2017.06b/moa-release-2017.06b/Ej1Res")
hoeff<-read.csv("Apartado1Ej1ACCKappa_Res.txt",sep = " ")
hoeffAdaptativo<-read.csv("Apartado1Ej2ACCKappa_Res.txt",sep = " ")

# test de normalidad
# shapiro
shapiro.test(hoeff$acc) # p-value = 0.3328 > 0.05
shapiro.test(hoeffAdaptativo$acc) # p-value = 0.1754 > 0.05
# jarque bera
library("tseries")
jarque.bera.test(hoeff$acc) # p-value = 0.1759  > 0.05
jarque.bera.test(hoeffAdaptativo$acc) # p-value = 0.4724  > 0.05

# test de normalidad valor Kappa
shapiro.test(hoeff$kappa) # p-value = 0.3323
shapiro.test(hoeffAdaptativo$kappa) #  p-value = 0.1766
```

Tras comprobar que tanto los datos de acierto como de Kappa de ambos algoritmos siguen una distribución normal, pasamos a comprobar si existen diferencias significativas usando un test estadístico paramétrico, en concreto, un *t-test*:

Listing 5: Script R para comprobación de normalidad y diferencias en los datos

```
# test de diferencia significativa
t.test(hoeff$acc,hoeffAdaptativo$acc) # p-value = 0.0006479
t.test(hoeff$kappa,hoeffAdaptativo$kappa) # p-value = 0.0006495
```

Obteniendo que sí hay diferencias significativas entre ambos algoritmos, pues obtenemos tanto para los valores de kappa, como para los de acierto en clasificación un p-valor < 0.05. Si calculamos la tasa de clasificación media y el valor Kappa medio por algoritmo:

Listing 6: Script R para comprobación de normalidad y diferencias en los datos

```
# valor medio de acierto en clasif
mean(hoeff$acc) #84.52885
mean(hoeffAdaptativo$acc) #84.39025
# kappa medio
mean(hoeff$kappa) #76.795
mean(hoeffAdaptativo$kappa) #76.58715
```

Obtenemos que Hoeffding Tree obtiene mejores resultados en media en clasificación que el Hoeffding Tree Adaptativo, y por tanto su valor Kappa promedio también es superior.

Entrenamiento online

Ejercicio 1

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.

Solución

Para resolverlo, tras probar con la semilla=2, generamos el siguiente script para automatizar el proceso de generar la población con diferentes resultados:

Listing 7: Script para generación de población de resultados de HoeffdingTree online

```
#!/bin/bash

for i in `seq 1 20`;
do
   java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask \
   "EvaluateInterleavedTestThenTrain -l trees.HoeffdingTree -s \
   (generators.WaveformGenerator -i $i) -i 1000000 -f 10000" \
   > ./Ej1Res/Apartado2ejercicio1semilla$i.txt
done
```

Donde:

- con **Interleaved Test-Then-Train** estamos especificando que queremos evaluar el clasificador Hoeffding Tree sobre el flujo espeficiado, mediante testeo primero y luego entrenamiento con cada muestra
- con **-l** indicamos el modelo que queremos entrenar, en este caso un árbol HoeffdingTree
- con **-s (generators.WaveformGenerator -i $i)** estamos indicando que queremos que las instancias sean generadas por un flujo WaveformGenerator con semilla aleatoria igual a la variable i, que toma valor según la iteracción del bucle
- con **-i 1000000** indicamos el total de instancias sobre las que queremos entrenar/evaluar
- con **-f 10000** indicamos que queremos que la frecuencia de muestreo sea 10.000

Tras la generación de la población de resultados, creamos otro Script en R que lee los 20 ficheros asociados a la ejecución y extrae los datos de nuestro interés: valor de la semilla, porcentaje de acierto y porcentaje del valor Kappa, escribiendolos en un nuevo fichero:

Listing 8: Script R que lee la población de resultados y crea fichero con la información de interés

```
setwd("/home/cris/Downloads/moa-release-2017.06b/moa-release-2017.06b/Ej1Res")

# creamos tabla a rellenar con sus nombres en las columnas
data<-cbind(seq(1:20),rep(0,20),rep(0,20))
colnames(data)<-c("seed","acc","kappa")

# rellenamos tabla a partir de los datos de los ficheros
for(i in seq(1:20)){
  dat<-read.csv(paste0("Apartado2ejercicio1semilla",i,".txt"))
  data[i,2]<-dat[nrow(dat),5]
  data[i,3]<-dat[nrow(dat),6]
}

write.table(data,"Apartado2Ej1ACCKappa_Res.txt")
```

Los resultados obtenidos para entrenamiento online del Hoeffding Tree se presentan en la tabla que sigue en la siguiente pagina:

Semilla	Acc (%)	Kappa (%)
1	83.8903	75.8362356949331
2	83.7851	75.677498015009
3	83.8876	75.829541470117
4	84.0451	76.0694603587848
5	83.8402	75.759994725058
6	83.9062	75.8590550656951
7	83.8867	75.829276988131
8	83.8687	75.8028419029038s
9	83.7875	75.6817212032629
10	83.8479	75.7714936861128
11	83.7456	75.6178349462264
12	83.8392	75.7591272121973
13	83.9761	75.9642551609865
14	83.8801	75.8192058978667
15	83.9843	75.9762001442894
16	83.8343	75.7518333609806
17	83.8963	75.8445017395676
18	83.8406	75.7617072631748
19	83.788	75.6828450081384
20	83.8152	75.723321424905

Ejercicio 2

Repetir el paso anterior, sustituyendo el clasificador por HoeffdingTree adaptativo.

Solución

Para replicar el proceso con el Hoeffding Tree Adaptativo, creamos y ejecutamos el siguiente script:

Listing 9: Script para generación de población de resultados de HoeffdingTree Adaptativo online

```
#!/bin/bash

for i in `seq 1 20`;
do
   java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask \
   "EvaluateInterleavedTestThenTrain -l trees.HoeffdingAdaptiveTree -s \
   (generators.WaveformGenerator -i $i) -i 1000000 -f 10000" \
   > ./Ej1Res/Apartado2ejercicio2semilla$i.txt
done
```

Dejando igual todos los parámetros y especificando que los ficheros de salida reciban el nombre de **Apartado2ejercicio2semilla[i].txt**.
Tras leer los resultados usando el mismo Script de R descrito en el apartado anterior, obtenemos los resultados que se describen en la siguiente tabla:

Semilla	Acc (%)	Kappa (%)
1	83.8042	75.7071683030803
2	83.7313	75.5967623000447
3	83.7875	75.6792025048753
4	83.7961	75.6960419534137
5	83.7144	75.5712869823438
6	83.8406	75.7607127313388
7	83.7784	75.6668833688617
8	83.8968	75.845067538964
9	83.8282	75.7427828950067
10	83.9	75.8495506621493
11	83.7407	75.6105013093504
12	83.7414	75.6123821191584
13	83.8943	75.8415733250357
14	83.8576	75.7855038464383
15	83.8748	75.8120830444433
16	83.8876	75.8315398544441
17	83.7386	75.607973527849
18	83.7614	75.6427618384802
19	19 83.75	75.6257100817952
20	83.8226	75.7343092244228

Ejercicio 3

Responda a la pregunta: ¿Cree que algún clasificador es mejor que el otro en este tipo de problemas? Razone su respuesta.

Solución

Listing 10: Script R para comparación de resultados mediante test estadísticos

```
setwd("/home/cris/Downloads/moa-release-2017.06b/moa-release-2017.06b/Ej1Res")
hoeff<-read.csv("Apartado2Ej1ACCKappa_Res.txt",sep = " ")
hoeffAdaptativo<-read.csv("Apartado2Ej2ACCKappa_Res.txt",sep = " ")

# test de normalidad
# shapiro
shapiro.test(hoeff$acc) # p-value = 0.3106 > 0.05
shapiro.test(hoeffAdaptativo$acc) # p-value = 0.112 > 0.05
# jarque bera
library("tseries")
jarque.bera.test(hoeff$acc) # p-value = 0.4348  > 0.05
jarque.bera.test(hoeffAdaptativo$acc) # p-value = 0.4432  > 0.05

# test de normalidad valor Kappa
shapiro.test(hoeff$kappa) # p-value = 0.3323
shapiro.test(hoeffAdaptativo$kappa) #  p-value = 0.1766

# test de diferencia significativa
t.test(hoeff$acc,hoeffAdaptativo$acc) # p-value = 0.008351
t.test(hoeff$kappa,hoeffAdaptativo$kappa) # p-value = 0.008368
```

Ejecutando el mismo script R descrito anteriormente para estos datos, comprobamos que ambas poblaciones de resultados son normales, y por tanto podemos usar un test paramétrico para compararlas, por lo que volvemos a aplicar un *t-test*, obteniendo tanto para el valor Kappa como para el acierto en clasificación un p-valor < 0.05 y por tanto, que sí hay diferencias significativas. Para comprobar qué algoritmo es mejor, calculamos su acierto medio en clasificación y su valor de Kappa medio:

Listing 11: Script R para comparación de resultados mediante test estadísticos

```
# valor medio de acierto en clasif
mean(hoeff$acc) #83.86725
mean(hoeffAdaptativo$acc) #83.80733
# kappa medio
mean(hoeff$kappa) #75.8009
mean(hoeffAdaptativo$kappa) #75.71099
```

Comprobando que, de nuevo para este tipo de problemas el Hoeffding Tree obtiene mejores resultados y por tanto un valor de Kappa más alto que el Hoeffding Tree Adaptativo.

Entrenamiento online en datos con concept drift

Ejercicio 1

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.

Solución

Listing 12: Script para entrenamiento de un Hoeffding Tree online mediante Interleaved Test-Then-Train con varias semillas

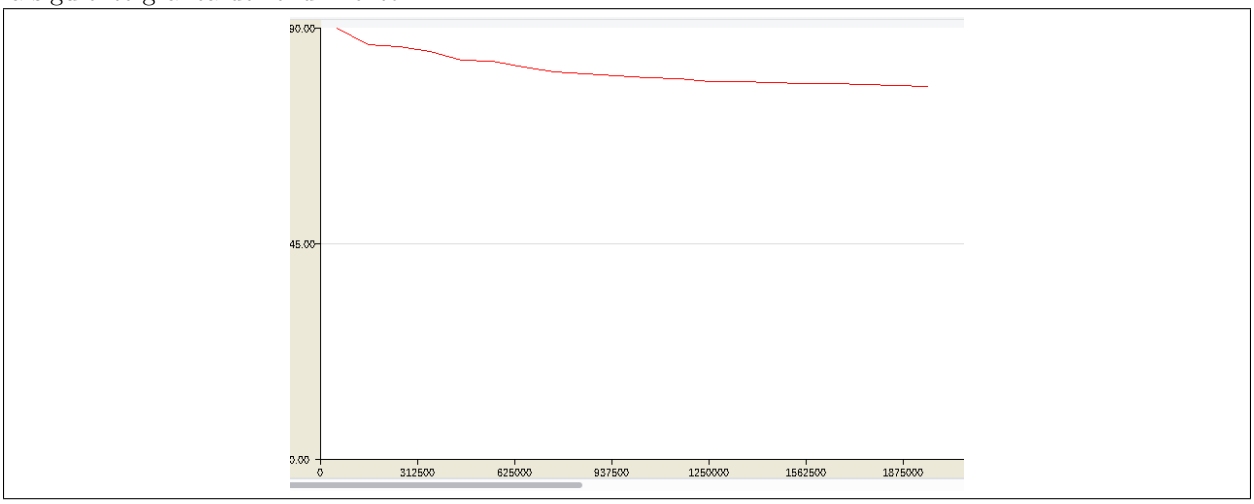
```
#!/bin/bash

for i in `seq 1 20`;
do
   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 $i -i $i -a 7 -n 3) \
   -i 2000000" > ./Ej1Res/Apartado3ejercicio1semilla$i.txt
done
```

De nuevo, especificamos el modelo con el parámetro **-l**, el Interleaved Test-then-train con **EvaluateInterleavedTestThenTrain** y el flujo generador de datos con la opción **-s**, donde indicamos que el generador será un RandomRBFGeneratorDrift. Con el parámetro **-s** indicamos que queremos que la velocidad de cambio sea 0.001, con **-k** indicamos el número de centroides con drift, con **-r** y **-i** indicamos las semillas aleatorias para el modelo y las instancias, respectivamente, con **-a** indicamos el número de atributos a generar y con

-n indicamos el número de centroides totales. Finalmente con el **-i** exterior al flujo indicamos las instancias con las que queremos entrenar / validar el modelo.

Si ejecutamos el modelo inicial en MOA con las semillas de modelo e instancias por defecto a 1, obtenemos la siguiente gráfica de rendimiento:



Donde vemos que el modelo al principio clasifica muy bien, pero después de que se produzca el drift decae su tasa de clasificación, siendo este en la última instancia de 77.45 %. El acierto medio en clasificación es de 80.86 % y el valor Kappa medio es de 61.65 %, siendo el valor Kappa obtenido en la última clasificación de 54.78 %, lo que denota un modelo poco robusto a los cambios de concepto.

Los resultados finales de las 20 pruebas con 20 semillas distintas se muestran en la siguiente tabla:

Semilla	Acc (%)	Kappa (%)
1	77.45455	54.7757723471886
2	83.95655	29.3917151648873
3	73.4734	43.0588295609063
4	73.0467	44.9855746952243
5	72.06165	43.1521883286061
6	83.37765	21.8729813463717
7	61.572	22.0496926464146
8	78.38975	12.7643684037922
9	98.1665	14.6081788421138
10	65.65775	29.1044298258018
11	74.93125	49.8076664335487
12	59.5718	14.515072275544
13	68.8432	22.3539725791203
14	76.17895	29.5535680557708
15	96.9101	5.02096922221682
16	80.4608	56.7645079268042
17	70.9067	25.4275073581716
18	97.5558	5.54894813389353
19	84.6607	15.1572570807118
20	76.8663	38.1591675664987

Ejercicio 2

Repetir el paso anterior, sustituyendo el clasificador por HoeffdingTree adaptativo.

Solución

Repetimos lo anterior sustituyendo el clasificador por un Hoeffding Tree adaptativo, guardando los resultados en ficheros generados con el nombre: **Apartado3ejercicio2semilla$i.txt**

Listing 13: Script para entrenamiento de un Hoeffding Tree Adaptativo online mediante Interleaved Test-Then-Train con varias semillas

```
#!/bin/bash

for i in `seq 1 20`;
do
    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 $i -i $i -a 7 -n 3) \
    -i 2000000" > ./Ej1Res/Apartado3ejercicio2semilla$i.txt
done
```

Los resultados obtenidos tras las 20 diferentes ejecuciones del algoritmo se muestran en la siguiente tabla:

Semilla	Acc (%)	Kappa (%)
1	96.24925	92.4913845097571
2	92.64715	73.9267030994111
3	96.3097	92.194394078399
4	95.9832	91.7716993068586
5	90.99615	81.6579176632214
6	94.9591	81.9302881617846
7	88.70825	77.2994290524042
8	90.8219	73.6628261896411
9	98.54465	44.7632656802649
10	87.76635	75.0545860395079
11	96.0234	91.9869427554932s
12	85.98955	71.4586413728736
13	94.35425	87.3096329277164
14	92.0137	79.1422140974413
15	97.3471	45.3605607698073
16	97.432	94.372406662667
17	93.33335	84.4778459429411
18	97.7868	48.4177298395338
19	92.18935	69.5571629489927
20	95.6582	89.4895479958483

Ejercicio 3

Responda a la pregunta: ¿Cree que algún clasificador es mejor que el otro en este tipo de problemas? Razone su respuesta.

Solución

Para responder a la pregunta, de nuevo cargamos los datos y aplicamos test estadísticos usando el siguiente Script:

Listing 14: Script R para comprobar diferencias entre en rendimiento de los algoritmos

```
setwd("/home/cris/Downloads/moa-release-2017.06b/moa-release-2017.06b/Ej1Res")
hoeff<-read.csv("Apartado3Ej1ACCKappa_Res.txt",sep = " ")
hoeffAdaptativo<-read.csv("Apartado3Ej2ACCKappa_Res.txt",sep = " ")

# test de normalidad
# shapiro
shapiro.test(hoeff$acc) # p-value = 0.2712 > 0.05
shapiro.test(hoeffAdaptativo$acc) # p-value = 0.2088 > 0.05
# jarque bera
library("tseries")
jarque.bera.test(hoeff$acc) # p-value = 0.6738  > 0.05
jarque.bera.test(hoeffAdaptativo$acc) # p-value = 0.4408  > 0.05

# test de normalidad valor Kappa
shapiro.test(hoeff$kappa) # p-value = 0.3623
shapiro.test(hoeffAdaptativo$kappa) #  p-value = 0.006073 < 0.05

# test de diferencia significativa
t.test(hoeff$acc,hoeffAdaptativo$acc) # p-value =2.127e-06
wilcox.test(hoeff$kappa,hoeffAdaptativo$kappa) # p-value = 2.017e-09

# valor medio de acierto en clasif
mean(hoeff$acc) #77.70211
mean(hoeffAdaptativo$acc) #93.75567
# kappa medio
median(hoeff$kappa) #27.26597
median(hoeffAdaptativo$kappa) #80.40007
```

Donde, en primer lugar, se comprueba la normalidad o no de las distribuciones de acierto en clasificación y de valor Kappa con un *shapiro test* y *test jarque bera*. Tras esto, concluimos que para ambos algoritmos los datos de acierto en clasificación sí siguen una distribución normal, por lo que se usa un test paramátrico *t-test*, para comprobar si hay diferencias significativas. Como para el Hoeffding adaptativo obtenemos que los valores Kappa no siguen una distribución normal, para comparar los valores Kappa de ambos algoritmos tenemos que aplicar un test no paramátrico, es este caso un *test de Wilcoxon*.

Tras aplicar los test, obtenemos que sí existen diferencias significativas entre ambos modelos, de hecho, el Hoeffding Tree obtiene un 77.70 % de acierto de clasificación medio, frente a un 93.75 % que obtiene el Hoeffding Tree Adaptativo, lo cual denota que el Hoeffding tree adaptativo se adapta mejor al concept drift que el Hoeffding Tree. Si comparamos los valores Kappa de ambos algoritmos, obtenemos que el valor mediano de el Hoeffding Tree adaptativo es superior al del Hoeffding Tree, para las 20 ejecuciones, lo cual reivindica que se adapta mejor a este problema.

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

Ejercicio 1

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.

Solución

Para el Hoeffding Tree creamos y ejecutamos el siguiente Script:

Listing 15: Script para entrenamiento de un Hoeffding Tree online con concept drift y olvido de instancias pasadas mediante Prequential con varias semillas

```
#!/bin/bash

for i in `seq 1 20`;
do
    java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask \
    "EvaluatePrequential -l trees.HoeffdingTree \
    -s (generators.RandomRBFGeneratorDrift -s 0.001 -k 3 -r $i -i $i -a 7 -n 3) \
    -i 2000000" > ./Ej1Res/Apartado4ejercicio1semilla$i.txt
done
```

Donde indicamos el nuevo método de evaluación con **EvaluatePrequential** y el tamaño de la ventana con el parámetro **-w**, sin embargo, este por defecto es 1000, así que no lo especificamos. Los resultados obtenidos de la ejecución del Hoeffding tree son los siguientes:

Semilla	Acc (%)	Kappa (%)
1	76.6	52.5587845317309
2	86.5	39.8256280421488
3	52.9	-0.10626992561104
4	65.3	29.6037523025862
5	63.3	25.2844078534841
6	82.1	16.9527697875104
7	56.7	11.3562292209175
8	80	18.8476364374113
9	98	8.91702340832458
10	64.5	25.3731343283582
11	73.6	47.5524475524476
12	57.6	11.0081268050238
13	69.8	17.9044429463062
14	78.1	33.4103624422282
15	97.2	-0.193229800329201
16	78	51.0578185134925
17	69	10.3823493700746
18	97.1	5.79521829521831
19	84.9	7.20027532633549
20	74.1	23.2583497285894

Para el Hoeffding tree adaptativo, repetimos lo mismo con el siguiente Script:

Listing 16: Script para entrenamiento de un Hoeffding Tree Adaptativo online con concept drift y olvido de instancias pasadas mediante Prequential con varias semillas

```
#!/bin/bash

for i in `seq 1 20`;
do
    java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask \
    "EvaluatePrequential -l trees.HoeffdingAdaptiveTree \
    -s (generators.RandomRBFGeneratorDrift -s 0.001 -k 3 -r $i -i $i -a 7 -n 3) \
    -i 2000000" > ./Ej1Res/Apartado4ejercicio2semilla$i.txt
done
```

Obteniendo los siguientes resultados:

Semilla	Acc (%)	Kappa (%)
1	96.6	93.1907121313472
2	95.3	82.198318309219
3	98	95.7559681697613
4	96.3	92.4689599023
5	91.9	83.4006869246596
6	95.7	84.938704028021
7	91.9	83.6938141175334
8	94.4	83.5269892631269
9	99.2	77.3819621147866s
10	91.6	83.0011130223616
11	98.1	96.1491065927295
12	87.5	74.5552253389244
13	98.4	96.3874137495033
14	94.1	84.1199776064769
15	96.3	12.1891019555725
16	98.8	97.3412858427902
17	94	85.506896301843
18	97.6	50.7773083391444
19	85.5	33.1328857079613
20	96.7	91.5002781727143

Ejercicio 2

¿Qué efecto se nota en ambos clasificadores? ¿A qué es debido? Justifique los cambios relevantes en los resultados de los clasificadores.

Solución

Sin necesidad de evaluar mediante test estadísticos y a simple vista, se observa que el acierto de clasificación del Hoeffding tree adaptativo es muy superior al de hoeffding tree para entrenamiento online con mecanismo de olvido. Esto también se observa en la diferencia de los valores kappa obtenidos por cada algoritmo, pues con el Hoeffding tree rara vez se llega a un Kappa del 50 %.

Con **Prequential** estamos indicando que queremos evaluar el clasificador testeando y luego entrenando con cada muestra del flujo. Observando las tablas de resultados, podemos observar que el Hoeffding Tree adaptativo se adapta bien al cambio de concepto y a el mecanismo de olvido mediante ventana deslizante de 1000 muestras, mientras que los resultados de Hoeffding Tree fluctúan según la semilla usada y no se observa un patrón claro antes o después del concept drift, cuando se ejecuta ventana deslizante como mecanismo de olvido de instancias.

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

Ejercicio 1

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.

Solución

Creamos y ejecutamos el siguiente Script Bash, que lanza la ejecución con 20 semillas diferentes:

Listing 17: Script para entrenamiento de un Hoeffding Tree online con concept drift y mecanismos de reinicialización de modelos con varias semillas

```
#!/bin/bash

for i in `seq 1 20`;
do
java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask \
"EvaluateInterleavedTestThenTrain -l (drift.SingleClassifierDrift -l
trees.HoeffdingTree) -s (generators.RandomRBFGeneratorDrift -s 0.001
 -k 3 -r $i -i $i -a 7 -n 3) -i 2000000" > ./Ej1Res/Apartado5ejercicio1semilla$i.txt
done
```

Dond con **-l** indicamos que queremos usar el clasificador simple basado en reemplazar el clasificador actual cuando se detecta un concept drift, con **-d** indicamos el método de detección de cambio de concepto, que por defecto es DDM y por eso no lo especificamos en el script, y dentro del **SingleClassifierDrift** con la opción **-l** indicamos el modelo Hoeffding tree a aprender.

Los resultados obtenidos para todas las ejecuciones se agrupan usando el Script de R mencionado anteriormente y se muestran en la siguiente tabla:

Semilla	Acc (%)	Kappa (%)
1	97.1995	94.3919003842531
2	90.50155	65.5687699828225
3	97.0895	93.8475436098632
4	96.5851	93.0011410650914
5	92.56285	84.8316978538835
6	96.5367	87.6157655741154
7	90.5222	80.9447173125267
8	91.61065	76.1416904118886
9	98.25575	20.8748697799271
10	89.46675	78.4907057869455
11	97.0355	94.028681807809
12	87.71595	74.94717395793
13	95.6094	90.1261819435396
14	92.465	80.3353841545105
15	97.00055	11.2706310080938
16	98.04425	95.7166728243708
17	94.15305	86.3684974724344
18	97.56775	7.17764184964568
19	88.046	43.9103702658066
20	96.9976	92.7041091480505

Ejercicio 2

Repita el paso anterior cambiando el clasificador HoeffdingTree por un clasificador HoeffdingTree adaptativo.

Solución

De nuevo, creamos y ejecutamos el siguiente Script Bash, que lanza la ejecución con 20 semillas diferentes:

Listing 18: Script para entrenamiento de un Hoeffding Tree online con concept drift y mecanismos de reinicialización de modelos con varias semillas

```
#!/bin/bash

for i in `seq 1 20`;
do
java -cp moa.jar -javaagent:sizeofag-1.0.0.jar moa.DoTask \
"EvaluateInterleavedTestThenTrain -l (drift.SingleClassifierDrift -l
trees.HoeffdingAdaptiveTree) -s (generators.RandomRBFGeneratorDrift -s 0.001
 -k 3 -r $i -i $i -a 7 -n 3) -i 2000000" > ./Ej1Res/Apartado5ejercicio2semilla$i.txt
done
```

Donde hemos sustituido el clasificador base por un Hoeffding Tree Adaptativo, obteniendo la siguiente población de resultados:

Semilla	Acc (%)	Kappa (%)
1	96.52275	93.038698221213
2	92.8668	74.7934774028823
3	96.8055	93.245829303353
4	96.24445	92.3027615257787
5	91.2725	82.2160413631178
6	95.6894	84.5533052061072
7	89.68885	79.2669897431198
8	91.24305	74.9384645608434
9	98.5662	45.2318199528825
10	88.25425	76.042006466411
11	96.60195	93.1545479357653
12	86.3948	72.2895948733506
13	95.00385	88.7730662968196
14	92.0956	79.3880397318764
15	97.5424	50.2471527443694
16	97.42345	94.3536399411526
17	93.4081	84.6568489515069
18	97.7865	47.64941851999
19	92.3333	70.1634269869136
20	96.2123	90.8124938512459

Ejercicio 3

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.

Solución

En la siguiente tabla se muestran el porcentaje de acierto en clasificación medio y mediano de cada método relativo a los apartados 2.3, 2.4 y 2.5, así como su valor Kappa medio y mediano.

Method	Acc mean	Kappa mean	Acc median	Kappa median
Interleaved Test-Then-Train + Hoeffding tree	77.70211	28.90362	76.52263	27.26597
Interleaved Test-Then-Train + Hoeffding tree Adapt	93.75567	77.31626	94.65668	80.40007
Prequential + Hoeffding tree	75.265	21.79946	75.35	18.37604
Prequential + Hoeffding tree Adap	94.895	79.06084	96	83.9069
SingleClassifierDrift + DDM + Hoeffding tree	75.265	21.79946	75.35	18.37604
SingleClassifierDrift + DDM + Hoeffding tree Adapt	94.895	79.06084	96	83.9069

Si comparamos el rendimiento de los 6 experimentos, comprobamos que en los tres casos (entrenamiento online con cambio de concepto, entrenamiento online con cambio de concepto y mecanismo de olvido y entrenamiento online con cambio de concepto y mecanismo de reinicialización) el uso del Hoeffding Tree adaptativo como modelo base proporciona mejores resultados en acierto de clasificación en media y mediano, y por tanto un mejor modelo que se ve reflejado en un valor Kappa más elevado.
También se puede comprobar que el uso de un clasificador simple que sustituye al clasificador actual cuando se produce el concept drift usando como detector del cambio DDM y el uso de una ventana de tamaño 1000 para olvidar instancias pasadas proporcionan identicos resultados tanto medios como medianos para las 20

ejecuciones, a pesar de que los valores por iteración son ligeramente distintos. Esto nos lleva a pensar que tanto incorporar un mecanismo de reinicialización cuando se produce un cambio de concepto como incorporar un mecanismo de olvido de instancias pasadas mediante ventana incrementan la potencia de clasificación de un hoeffding tree adaptativo cuando se quiere entrenar este clasificador online con datos que presentan cambios de concepto, mientras que la incorporación de estos mecanismos a un Hoeffding tree para este tipo de problema resulta en un peor rendimiento del modelo.