Universidade Federal do ABC
Programa de Pós-graduação em Ciência da Computação
Inteligência na Web e Big-Data

Classificação de Eventos em Logs de Interação Capturados por Eye Tracker

Leandro Marega Ferreira Otani
RA 131710240
leandro.otani@ufabc.edu.br

São Paulo, 08 de dezembro de 2017

1 Introdução

Softwares e demais sistemas interativos possuem desde seus primórdios modais[1] de interação como mouse e teclado. Estes modais, no entanto, não somente viabilizam a interação de pessoas com dispositivos como também são fontes bastante descritivas de dados: mouses e teclados geram registros (logs) de botões apertados e/ou movimentos realizados; celulares possuem informação como orientação de tela, tempo de atividade, nível de movimentação do dispositivo e local do toque na tela; Relógios inteligentes são capazes até mesmo de capturar dados fisiológicos como condutância da pele e batimento cardíaco.

Neste contexto de dispositivos e geração de logs, este trabalho trata de um caso especial de dados de interação: Séries temporais geradas por Eye Trackers (exemplo na figura 1), dispositivos que mapeiam a posição e movimento dos olhos ao mesmo tempo que, em alguns casos, fazem a classificação de eventos como fixações (olhos parados observando uma região de interesse) e sacadas (movimento dos olhos entre uma fixação e outra) [3, 1] a partir dos dados coletados no mapeamento.

Figura 1: Dispositivo Eye-tracker EyeX. Produzido por Tobii [6]

Como algumas das aplicações relevantes à área da computação, podem ser citados: (i) O estudo de usabilidade de sistemas interativos [2]; (ii) O desenvolvimento de sistemas acessíveis, com a introdução dos olhos como mecanismo de interação [4]; (iii) A recomendação de conteúdo com base na análise de logs gerados [7].

Apesar da diversidade de sensores Eye Tracker disponíveis no mercado, a estrutura mais básica de dados do Eye Tracker (e a considerada para esse trabalho) possui a anotação de tempo (timestamp padrão UNIX), e coordenadas (x,y) referentes ao ponto de observação do usuário.

[1] Termo utilizado na área de Interação Humano-Computador para descrever recursos que permitem interação entre pessoas e sistemas

2 Proposta

Uma vez considerado o contexto de análise de logs relacionados a movimentos do olho, este trabalho apresenta um modelo paralelizado do algoritmo "I-VT"que, como descrito na implementação iterativa em [3] e referenciado na figura 2, trata da classificação de pontos de fixação e sacada a partir do calculo de velocidade ponto-a-ponto que, dado um limiar definido, posiciona velocidades acima do valor estabelecido como sacadas e velocidades dentro do valor estabelecido como fixação. Naturalmente, fixações subsequentes são colapsadas pelo algoritmo e um centroide é calculado para representar o grupo colapsado.

Table 2: Pseudocode for the I-VT algorithm.

```
I-VT (protocol, velocity threshold)

Calculate point-to-point velocities for each
point in the protocol

Label each point below velocity threshold as
a fixation point, otherwise as a saccade
point

Collapse consecutive fixation points into
fixation groups, removing saccade points

Map each fixation group to a fixation at the
centroid of its points

Return fixations
```

Figura 2: Pseudocódigo - Algoritmo I-VT. Elaborado por Salvucci e Goldberg [3]

3 Implementação

A implementação da versão paralelizada do algoritmo I-VT foi realizada por meio da API Spark para a linguagem Python (PySpark [5]), conforme apresentado na figura 3.

```
13
14  def parallelizedVelocityThreshold(dataset, threshold):
15      firstSetRDD = sc.parallelize(dataset[:-1])
16      secondSetRDD = sc.parallelize(dataset[1:])
17      zippedRDD = firstSetRDD.zip(secondSetRDD)
18      mappedVelocities = zippedRDD.map(lambda x: (x[0],(euclideanDistance(x) / abs(x[0][0] - x[1][0]))))
19
20      classifiedVelocities =  (mappedVelocities
21                               .map(lambda x: (x[0], "Saccade" if x[1] > threshold else "Fixation"))
22                               .sortBy(lambda x: x[0][0])
23                               .collect())
24
25      count = -1
26      colapsedFixations = []
27      saccadeHappened = False
28
29      for elem in classifiedVelocities:
30          if elem[1] == "Saccade":
31              saccadeHappened = True
32          else:
33              if len(colapsedFixations) == 0 or saccadeHappened:
34                  count += 1
35                  saccadeHappened = False
36              print elem
37              colapsedFixations.append((count,elem[0]))
38
39      colapsedRDD = sc.parallelize(colapsedFixations, 4)
40      countRDD = sc.parallelize(colapsedFixations, 4).map(lambda x: (x[0], 1)).reduceByKey(add)
41
42      sumCentroids = colapsedRDD.reduceByKey(lambda (tsa, xa, ya),(tsb, xb, yb): ((tsa+tsb, xa+xb, ya+yb)))
43      zippedCentroids = sumCentroids.join(countRDD)
44      centroids = zippedCentroids.map(lambda (k,v): (k, v[0][0]/v[1], v[0][1]/v[1], v[0][2]/v[1] ))
45      return centroids.collect()
46
```

Figura 3: Método principal - parallelizedVelocityThreshold

Como pode ser observado, o algoritmo recebe como entrada um conjunto de dados (dataset) e um limiar de classificação (threshold) para então iniciar com a paralelização de duas instâncias do dataset

(linhas 15-16), uma excluindo a última posição do conjunto de dados original e outra excluindo a primeira, para que então pudessem ser juntadas (linha 17) e a velocidade ponto-a-ponto fosse computada (linha 18) com o apoio do cálculo da distância euclidiana (Anexo A - sub-rotina euclideanDistance).

Uma vez com as distâncias calculadas, um mapeamento ordenado foi realizado (linha 20-23) para classificar os pontos como "Saccade"(Sacada) e "Fixation"(Fixação) de acordo com o limiar definido em "threshold".

Uma implementação iterativa foi realizada (linhas 29-37) para agrupar as fixações. Optou-se por não paralelizar esta etapa com o objetivo de reduzir a complexidade de implementação (e possivelmente a complexidade computacional), uma vez que o correto agrupamento dos dados depende da observação da posição anterior do vetor. Foram realizadas algumas tentativas de implementação paralela dessa função mas não houve sucesso.

Por fim, a implementação resumiu-se a paralelização do vetor de fixações (linha 39), que permitiu a redução (linha 42) e mapeamento dos centroides (linha 44) com auxílio da junção da RDD reduzida com uma RDD que guardava a contagem de pontos em cada grupo de fixações (linha 43). O algoritmo retorna, conforme especificação, os pontos de fixação (linha 45).

4 Conclusões e próximos passos

Este trabalhou se mostrou uma excelente oportunidade de consolidar conhecimentos importantes sobre a paralelização de algoritmos por meio de técnicas de mapeamento e redução. Alguns desafios certamente foram encontrados na tentativa "tradução"do paradigma de caráter iterativo para uma implementação majoritariamente paralelizada, mas é entendido que isso é parte do processo de compreensão de um novo paradigma e que objetivo de paralelização de áreas críticas do algoritmo pode ser cumprido.

Com isso, espera-se avançar para uma análise comparativa desta implementação com outros algoritmos existentes (como os descritos em [3]), que também serão paralelizados seguindo o mesmo processo com o objetivo de auxiliar trabalhos futuros na escolha da implementação mais apropriada em termos de velocidade de execução e acurácia de classificação para alguns cenários de uso ainda a serem definidos.

Referências

[1] Gartner IT Glossary. Eye tracking, 2017. https://www.gartner.com/it-glossary/eye-tracking/.

[2] Joseph H. Goldberg, Mark J. Stimson, Marion Lewenstein, Neil Scott, and Anna M. Wichansky. Eye tracking in web search tasks: Design implications. In *Proceedings of the 2002 Symposium on Eye Tracking Research & Applications*, ETRA '02, pages 51–58, New York, NY, USA, 2002. ACM.

[3] Dario D. Salvucci and Joseph H. Goldberg. Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 Symposium on Eye Tracking Research & Applications*, ETRA '00, pages 71–78, New York, NY, USA, 2000. ACM.

[4] J. David Smith and T. C. Nicholas Graham. Use of eye movements for video game control. In *Proceedings of the 2006 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE '06, New York, NY, USA, 2006. ACM.

[5] Apache Spark. Spark python api docs, 2017. http://spark.apache.org/docs/latest/api/python/index.html.

[6] Tobii. Tobii eyex, 2017. https://tobiigaming.com/product/tobii-eyex/.

[7] Songhua Xu, Hao Jiang, and Francis C.M. Lau. Personalized online document, image and video recommendation via commodity eye-tracking. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, pages 83–90, New York, NY, USA, 2008. ACM.

Appendices

Anexo A - sub-rotina euclideanDistance
Abaixo é apresentada a implementação em Python do cálculo de distância euclidiana, dada pela fórmula

$$D(c1, c2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

```
7   def euclideanDistance(baseTuple):
8       coordinateA = baseTuple[0]
9       coordinateB = baseTuple[1]
10
11      return (math.sqrt(math.pow((coordinateB[1] - coordinateA[1]), 2) +
12          math.pow((coordinateB[2] - coordinateA[2]), 2)))
```

Figura 4: Algoritmo de cálculo de distância euclidiana