Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 34 (2014) 297 – 304

Procedia Computer Science

11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC-2014)

Smarter smartphones: understanding and predicting user habits from GPS sensor data

G. Maggiore, C. Santos, A. Plaat

Academy for Digital Entertainment
NHTV University of Applied Sciences
Breda, Netherlands

Abstract

Smartphones and similar devices allow access to a wealth of information. Navigating this wealth of information is problematic. Semantic locations, assigned to observed GPS user movements, can help in providing inforamtion that is useful to the user at a specific time or place. This paper shows how a stream of sensor data can be processed and interpreted to determine (i) the locations of interest for a user, such as home, work, etc, and (ii) to predict the expected future transitions between such locations. We have implemented our algorithms in a fully functional prototype smartphone app and backend, and we present results based on actual usage data gathered over the past few months. We conclude that inferred semantic location information allows a smart device to offer personalized, contextual, information without the need for the user to perform any explicit query. Our system is open source, and can be used to build context-aware recommender systems that suggest content which is at the right time and at the right place.

Selection and peer-review under responsibility of Conference Program Chairs

Keywords: recommender systems, mobile systems, clustering

1. Introduction

Smartphones and similarly "always on, always connected" devices allow access to a wealth of information [1]. Public transportation time-tables, shops with sales, new restaurants, public events, etc. can be accessed through these devices. This is one of the major contributing factors of the widespread adoption of mobile devices in modern society [2]. To make smartphone even more useful, two drawbacks must be solved: *(i)* the amount of available information is overwhelming, and is not all relevant for each user; and *(ii)* small devices

* Corresponding author. Tel.: +31-076-533-2203 ; fax: +31-076-533-2205.
E-mail address: maggiore.g@nhtv.nl

Selection and peer-review under responsibility of Conference Program Chairs
doi:10.1016/j.procs.2014.07.027

can be cumbersome to type on for extended periods, often making navigation of this wealth of information problematic.

In this paper we discuss a working implementation of how observation of user movements (via GPS), together with an Internet connection, allows inferencing semantic locations[3] such as home, favourite shops and restaurants, gyms, parks, workplace, etc. Such knowledge is used to offer personalized, contextual information without the need for the user to perform a query; for example, the device can vibrate when passing in front of a shop, displaying the message "this shop now has a sale, you may want to check it!". By anticipating the most likely immediate movement of a user it also becomes possible to offer information that is contextual with respect to likely future circumstances, such as "the 9.45 train from London to Paris is late!"

Problem statement: in this paper, we show how the stream of sensor data gathered from a smartphone can be automatically processed in order to determine: *(i)* the locations of interest of a user and their semantic label such as *home*, *work*, etc; and *(ii)* the predicted future transitions between such locations. We present a working implementation of our algorithms, and an evaluation.

We begin by proposing the overview of a system (Section 3) for storing the user history and inferring his next most likely activities. We make use (Section 4) of a layered, density-based clustering system to identify the locations of interest, and of a multi-resolution table of transitions to identify the typical patterns of travel between the clusters; the transition tables are also used to detect the *semantic* (home, work, etc.) of the clusters. Preliminary tests (Section 5) on human subjects yield promising results. In particular, our tests show good prediction accuracy, and low computational requirements.

Our system is open source. Our work will allow building context-aware recommender systems that suggest content which is at the right time and at the right place.

2. Related work

Mobile computing has seen, in recent years, a large body of research work. Various forms of location-based inference of activities have been the subject of related research.

One significant line of research uses multiple, cheap radio-frequency-identification (RFID) devices in order to track fine-grained activities such as using specific objects in a room or a gym[4,5,6,7,8]. As an alternative to RFID, other studies focus on using various other kinds of sensors in order to determine indoors location and offer contextual services through computing devices[9,10,11,12]. Of course, these RFID-enabled environments must first be set up manually, a slow and expensive process, although it is expected that deployment will proliferate, increaing the applicability of such techniques.

There exist various other research efforts in the direction of using mobile phone sensors to infer user information. These efforts are related to our approach[13,14,15,16,3,2]. Many of the proposed solutions rely on a single technique, such as clustering, Bayesian networks, or some other statistical learning technique. Our approach differs in that it offers a layered mechanism that combines multiple machine/statistical learning approaches such as clustering and probabilistic models. Given our results, we have come to believe that layered systems, possibly even capable of integrating multiple unreliable sensors, are the most promising direction of investigation.

3. The idea

The location inference system is based on *density-based clustering*. The shape (see Figure 1) of clusters is determined by the average velocity of the samples associated with the cluster. We distinguish samples in two groups: *(i)* static, the user was *not moving*; and *(ii)* dynamic, the user was *moving*. Clusters that contain static entries have a circular shape defined by the maximum radius allowed for clusters. Moving clusters are shaped like an ellipse, where the ellipse axis is parallel to the average velocity of the entries recorded in the cluster.

Fig. 1: Moving and static clusters

Fig. 2: Some entries from the transitions tables; color of arrow represents length of transition

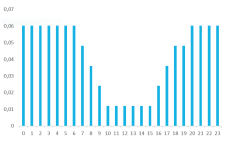

Fig. 3: Home hits by hour

Clusters "degrade" over time. If a cluster is not updated regularly, then it is removed from the system. This ensures that only fresh information is maintained.

We use the history of visited locations to infer a Markov process. The nodes of the process are the clusters found at the previous step of the algorithm, and the transitions are (intuitively enough) the transitions between clusters that were witnessed so far. For an example, see Figure 2. Transitions also have a *duration*. The duration of a transition measures how long it took to travel between the clusters that this transition connected. For each supported transition duration, we store a separate table. The supported transition durations grow exponentially.

The Markov process is also used for semantic label inference. We built, off-line, a series of probability distributions for each possible label. These distributions describe the expected patterns for assigning labels to a cluster. For example, we expect that the user will spend late afternoons and nights at home, and so the probability distribution of expected home usage will look like Figure 3. We compare the expected distributions with those we observed with the Kullback-Leibler (KL)[17] distance between probability distributions.

We can also build a reliability estimator that helps us improving correctness by avoiding potentially wrong predictions. By computing the KL distance between the complete distributions of all observations, and that of just a subset of the last observations, we can estimate how well the complete distribution predicts recent events, that is the current reliability of the system.

4. Technical details

We now illustrate the system in more detail and with extensive pseudo-code in the ML style. At the core of the system we have the `Analyzer`, which is responsible for storing all the information about clusters, transition tables, etc. The `Analyser` data type contains: the set of `Clusters`, which are all the clusters that have been detected so far in the system; the `PreviousHits`, which are the clusters where the latest observations fell (and *when* they happened); and the total number `NumHits` of observations seen so far:

```
type Analyser = { Clusters : [Cluster]; PreviousHits : [Cluster×DateTime]; NumHits : int }
```

A `Cluster` is defined as: a series of `Cells`; the last time `LastHit` when this cluster was hit by an observation; the transitions `Transitions`, indexed by the hour of the day, into other clusters; the distribution of stays `Stays` in this cluster, indexed by the hour of the day; and the optional user annotation of this cluster `UserAnnotation`:

```
type Cluster = { Cells : [Cell]; LastHit : DateTime; Transitions : Transitions
  Stays : Stays; UserAnnotation : Option<string> }
```

The `Stays` is, at its core, the histrogram `Stays` of how many times we have recorded observations at a given hour and minute. We also store a queue `LastHits` of the last hits that went into the `Stays` histogram. We also store a smaller version of the `Stays` histogram, which is called `RecentStays`, and its associated `RecentLastHits` queue. `RecentStays` will be used in order to determine how much the `Stays` histogram is a good approximation of the most recent user behaviour. If `Stays` and `RecentStays` differ too much, then it means that the user is behaving in an unexpected way, and thus predictions from `Stays` are less reliable:

```
type Stays = { Stays : [[int]]; LastHits : [Hour × Minute]; RecentStays : [[int]];
    RecentLastHits : [Hour × Minute] }
```

The transitions are stored similarly to the `Stays`, but of course with some additional data that represents the movements between clusters. Most importantly, we store the `Transitions` table, which for every hour and minute of the day, and for every possible transition duration (logarithmically indexed), maintains a map that for every reached cluster stores how many times this transition has happened. We also store the same queues as we did for `Stays`, plus the table of the latest observations:

```
type Transitions = { Transitions : (ΔT×Hour×Minute×Cluster)→int; LastHits : [Cluster×Δt×
    Hour×Minute]; RecentTransitions : (ΔT×Hour×Minute×Cluster)→int; RecentLastHits : [
    Cluster×Δt×Hour×Minute] }
```

A cell is a simple data structure. It stores a position, the average speed and movement direction of the hits recorded in the cell, and the average position of the hits recorded. The cell also stores the number of hits recorded here so far. An observation records the user position, heading, speed, and also when the observation was taken. We omit both cell and observation as they are trivially defined.

When we obtain a new observation, we must add it to an existing cluster or create a new cluster for it:

```
let AddObservation(cs:Analyser, o:Observation) =
```

We look for a cluster that is sufficiently[1] close to the current observation:

```
  if ∃ c ∈ cs.Clusters | AdjustedDistance(c, o) ≤ ε2) then
```

If we find such a cluster, then we add a hit to the cluster, store the current hit in the `PreviousHits`, and update the transitions and stays of the hit cluster:

```
    do cs.PreviousHits ← cs.PreviousHits + (c, o.Time)
    do cs.UpdateStays(c, o)
    do cs.UpdateTransitions(c, o)
```

We then look for a cell that contains the observation. If there is such a cell, then we add the observation to it, otherwise we just add a new cell to the cluster:

```
    if ∃ c ∈ cluster.Cells | Distance(c.Position, o.Position) ≤ ε1) then
      cluster.AddToCell(c,o)
    else
      cluster.AddNewCell(o)
```

If the appropriate cluster could not be found, then we simply create a new cluster, store the current hit in the `PreviousHits`, and update the transitions and stays of the hit cluster:

```
  else
    let new_cluster = Cluster.Create(o)
    do cs.Clusters ← cs.Clusters + new_cluster
    do cs.PreviousHits ← cs.PreviousHits + (new_cluster, o.Time)
    do cs.UpdateStays(new_cluster, o)
    do cs.UpdateTransitions(new_cluster, o)
```

Every time we have recorded a certain number of observations, we perform some clean up by removing all previous hits that are beyond the interesting event horizon. We then iterate all clusters in the system. If a cluster is too old, then we will remove it. Removing a cluster also requires to remove it from all the transition tables of connected clusters:

```
do cs.PreviousHits ← { c,t | (c,t) ∈ cs.PreviousHits ∧ not(too_old(t)))
do cs.Clusters ← { c | c ∈ cs.Clusters ∧ not(too_old(c)))
```

We update the transitions into a newly observed cluster with the `PreviousHits`. For each previous hit, we determine the logarithm of the time difference in minutes between that hit and the current hit. The logarithm of the difference in minutes allows us to maintain a variable resolution of our transitions:

[1] Controlled by one of the multiple customizable parameters, which are named ϵ_i.

transitions in the immediate future are allocated more indices, while transitions further away take less indices and are thus less precise:

```
let UpdateTransitions(cs:Analyser, c:Cluster, o:Observation) =
  for p,t_p in cs.PreviousHits do
    let Δm_2 = log_2 ((o.Time - t_p) / ε_5)
```

We then add a transition from the time (hour and minute) of the previous observation, with the transition length we have just computed:

```
p.Transitions.AddHit(c,Δm_2,t_p.Hour,t_p.Minute / ε_5)
```

The user may also choose to add an annotation for the clustering system. When this is done, we look for the closest cluster and then simply add the label to its `UserAnnotation` field.

We can now perform predictions about what clusters are the most likely to be visited in a given amount of time:

```
let PredictFutureClusters(cs:Analyser,Δt:TimeSpan) =
```

We find the last previous hit as the starting point of our prediction:

$$\texttt{let } c_0,t_0 = \max_{c,t \in cs.Clusters} t$$

We then find the indices of the prediction. The prediction will be taken from the `Transitions` table at the index given by the hour and minute of the last hit, plus the logarithm of the prediction delta time. For each transition found, we extract the destination cluster c_1. We then estimate the reliability of the prediction by combining the probability of the transition, and the overall confidence of the transitions table that we used:

$$\{\ c_0,c_1,p_{trans}\ |\ trans \in c_0.Transitions.[\log_2(\Delta t.TotalMinutes\ /\ \epsilon_5)].[t_0.Hour].[t_0.Minute\ /\ \epsilon_5] \wedge c_1 = trans.Cluster \wedge p_{trans} = |trans|\ /\ |c_0.Transitions|\ \}$$

We compute the distance between a cluster and an observation by adjusting the shape of the cluster depending on its average speed:

```
let AdjustedDistance(c:Cluster, o:Observation) =
```

We begin by computing the distance between the cluster and the observation:

```
  let distance = Distance(c, o)
```

If both cluster and observation are moving, then we adjust the distance so that it is reduced along the direction of movement and increased when perpendicular to it:

```
  if c.Moving then
    let axis_alignment = 1.0f - abs(Dot(c.AverageHeading.Normalized, Normalize(o.
        Position - c.AveragePosition)))
    if o.Speed > e3 then
      let vel_alignment = 1.0f - abs(Vector2<1>.Dot(c.AverageHeading.Normalized, o.
          Heading.Normalized))
      let elongation = smoothstep (c.AverageSpeed / 10.0f) 0.1f 5.0f
      distance × smoothstep axis_alignment 0.5f 10.0f ×
        smoothstep vel_alignment 0.5f 10.0f / elongation
```

If the cluster is moving but the observation is not (or the opposite), then we return a distance of $+\infty$ because a moving observation cannot go into a static cluster and vice-versa:

```
    else +∞
  elif o.Speed > ε_3 then +∞
```

If neither the cluster nor the observation are moving, then we return the linear distance:

```
  else distance
```

Clusters estimate their semantic location by comparing their stays and transitions with an expected pattern of the `Stays`. An `Activity` is simply a histogram of the expected distribution of hits with respect to time at a given location:

```
let SemanticLocationDistance(c:Cluster,a:Activity) =
```

If the cluster is already annotated with the name of the activity, then we return the lowest possible distance (which corresponds to the highest possible confidence) value:

```
if c.UserAnnotation = a.Name then log 0.0
```

We begin by extracting the stays, condensed by hour, and we compute the Kullback-Leibler divergence between these stays and those coming from the activity to match:

```
else
  let stays =
    ∑ {p × log(p / q) | h ∈ [0..23] ∧ p = ∑ { c.Stays.[h].[m] | m ∈ [0..59/ϵ5]} ∧ /
         |c.Stays| ∧ q = a.Stays.[h] ∧ q > 0.0 ∧ p > 0.0 }
```

We also use the transitions from the cluster into itself. This allows us to estimate how many of the stays are likely to be sequential, because if that is not the case then the number of transitions from the cluster into itself at that hour will be quite low:

```
  let transitions =
    ∑ { p × log(p / q) | h ∈ [0..23] ∧ p = c.Transitions.SelfTransitions(c, 1.0).[h]
         ∧ q = a.Stays.[h] ∧ q > 0.0 ∧ p > 0.0 }
```

The final estimate is the combined probability of the stays and transition estimates:

```
  log stays + log transitions
```

When we register a hit at a given time (hour and minute) then we add it to the `Stays` of a cluster:

```
let AddHit(t:Stays,h,m) =
```

We enqueue the hit in the `LastHits` and increase by one the `Stays` histogram at location h, m:

```
do t.LastHits.Enqueue(h,m)
do t.Stays.[h].[m] ← t.Stays.[h].[m] + 1
```

We then remove all the `LastHits` in excess, decreasing the appropriate entries in `Stays`. This ensure that `Stays` will only contain a certain number of the most recent observations:

```
while (|t.LastHits| ≥ ϵ8) do
  let h,m = t.LastHits.Dequeue()
  do t.Stays.[h].[m] ← t.Stays.[h].[m] - 1
```

We perform the same insertion and clean up operations on the `RecentStays` and `RecentLastHits`, in order to also track a smaller, more recent subset of observations:

```
do t.RecentLastHits.Enqueue(h,m)
do t.RecentStays.[h].[m] ← t.RecentStays.[h].[m] + 1
while (|t.RecentLastHits| ≥ ϵ9) do
  let h,m = t.RecentLastHits.Dequeue()
  do t.RecentStays.[h].[m] ← t.RecentStays.[h].[m] - 1
```

The confidence of a `Stays` histogram is then given by how much the long-term prediction `Stays` is reflected by the recently observed behaviour `RecentStays`. If the recent behaviour is not well predicted by the long-term observation, whether because of a temporary change in user behaviour or because of a deeper change in the user habits, then we will be able to measure a decrease in `Confidence`. The `Confidence` is computed as the Kullback-Leibler distance between `Stays` and `RecentStays`:

```
let Confidence(t.Stays) =
  ∑ { q × log(q / p) | h ∈ [0..t.Stays.Length-1] ∧ m ∈ [0..to t.Stays.[h].Length-1] ∧
       p = t.Stays.[h].[m] / |t.LastHits| ∧ q = t.RecentStays.[h].[m] / |t.
      RecentLastHits| ∧ q > 0.0 ∧ p > 0.0 }
```

The management of the transitions is extremely similar to that of the stays, with some added book-keeping in storing and separating transitions by destination cluster. When a new hit is recorded, we also update the transitions table. We only need to know the destination cluster `c`[2] where the transition ended,

[2] The source cluster is implicitly the one that stores the transition table itself

the length of the transition `e`, and the time (h, m) when the transition started. Finally, we add the new hit to the `RecentLastHits` and `RecentTransitions`, in order to track some of the most recent transitions. We compute the Kullback-Leibler distance between `Transitions` and `RecentTransitions` as a measure of the confidence in our ability to estimate the recent transitions through the full history of transitions. Some care must be taken because some of the clusters may be present in `Transitions` but not in `RecentTransitions`. We omit the pseudo-code as it is almost identical to the previous listings.

5. Evaluation

We measure the accuracy of activities and locations with a single experiment. We tested the system on data gathered from a small group of 10 students. We performed an analysis of the output of the system, which showed clusters and connectors on a map. As a result, we observe that the following locations of interest were consistently identified by the system:

a) school (as *workplace*); *b*) main home (the one where the student lives during the week); *c*) secondary home(s) (parents place visited during the weekend, or girlfriend home where many days are spent); *d*) pubs and similar entertainment locations; *e*) train and bus stations.

The following transitions of interest were consistently identified by the system between:

a) main and the secondary homes (and any bus/train stations in between); *b*) main home and school (and any bus/train stations in between); *c*) homes and pubs.

We believe that this preliminary evaluation clearly shows that our system is a promising conceptual and engineered tool for automatically identifying locations of interest in users and their transitions.

We also show that the system remains reasonably lean over time. This means that the number of clusters should not just grow indefinitely, and that old and less useful clusters which are just noise are cleaned up routinely. Figure 4 shows that the number of clusters stabilizes after a while.

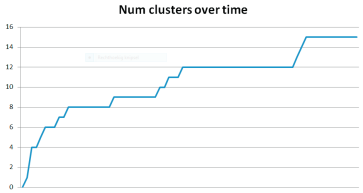

Fig. 4: Clusters over time

Finally, we measured the performance of the system, which is built with the Casanova[18] programming language for games and simulations, and run on a core i5 CPU. The average time it took to add an observation to the clustering system was 2 milliseconds, while the average time it took to perform predictions was 1 millisecond. Both operations are fast enough to be performed in real-time for a single user.

6. Conclusions and future work

Smartphones and similar "always on, always connected" devices offer large amounts of information that can be used in order to better understand some aspects of the user, such as frequently visited places and routines.

In this paper, we have shown how the stream of sensor data gathered from a smartphone can be automatically processed in order to determine: *a*) the **locations of interest** of a user and their semantic label such as *home*, *work*, etc. *b*) the **expected future transitions** between such locations

We have tested our prototype on a group of students, obtaining encouraging results. Our system iss able to correctly identify their locations of interest and their movements.

The work as presented offers possibilities for extension. On the one hand the accuracy of the activity inference system can be improved, using larger amounts of data from which to extract *staying patterns*. On the other hand, we will work on ways to *use* the inferred data for contextual, personalized recommendations. For example, we might connect locations and browsing preferences, so that when the user is at home then the appropriate content is offered.

Our system shows that a layered combination of smeantic location techniques allows powerful inference about users, and offers useful techniques for improved location-aware recommendations.

Acknowledgement

This work is part of the larger IMI project inside the Vivid initiative.
This research was co-funden by the European Union via the Interreg 2 seas program.

References

1. Wang, D., Sangwon, P., Daniel, F.. An examination of information services and smartphone applications 2011;.
2. Lane, N., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., Campbell, A.. A survey of mobile phone sensing. *Communications Magazine, IEEE* 2010;.
3. Pradhan, S.. Semantic location. *Personal Technologies* 2000;.
4. Philipose, M., Fishkin, K.P., Perkowitz, M., Patterson, D.J., Fox, D., Kautz, H., et al. Inferring activities from interactions with objects. *IEEE Pervasive Computing* 2004;.
5. Xiong, J., Seet, B.C., Symonds, J.. Human activity inference for ubiquitous rfid-based applications. In: *Ubiquitous, Autonomic and Trusted Computing, 2009. UIC-ATC '09. Symposia and Workshops on*. 2009, .
6. Consolvo, S., McDonald, D.W., Toscos, T., Chen, M.Y., Froehlich, J., Harrison, B., et al. Activity sensing in the wild: A field trial of ubifit garden. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2008, .
7. Denning, T., Andrew, A., Chaudhri, R., Hartung, C., Lester, J., Borriello, G., et al. Balance: Towards a usable pervasive wellness application with accurate activity inference. In: *Proceedings of the 10th Workshop on Mobile Computing Systems and Applications*. 2009, .
8. Patterson, D., Fox, D., Kautz, H., Philipose, M.. Fine-grained activity recognition by aggregating abstract object usage. In: *Wearable Computers, 2005. Proceedings. Ninth IEEE International Symposium on*. 2005, .
9. Coronato, A., Esposito, M., Pietro, G.. A multimodal semantic location service for intelligent environments: an application for smart hospitals. *Personal and Ubiquitous Computing* 2009;.
10. Kolomvatsos, K., Papataxiarhis, V., Tsetsos, V.. Semantic location based services for smart spaces. In: *Metadata and Semantics*. 2009, .
11. Li, D., Lee, D.L.. A topology-based semantic location model for indoor applications. In: *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2008, .
12. Hu, H., Lee, D.L.. Semantic location modeling for location navigation in mobile environment. In: *Mobile Data Management, 2004. Proceedings. 2004 IEEE International Conference on*. 2004, .
13. Partridge, K., Price, B.. Enhancing mobile recommender systems with activity inference. In: Houben, G.J., McCalla, G., Pianesi, F., Zancanaro, M., editors. *User Modeling, Adaptation, and Personalization*; Lecture Notes in Computer Science. 2009, .
14. Liu, J., Wolfson, O., Yin, H.. Extracting semantic location from outdoor positioning systems. In: *Mobile Data Management, 2006. MDM 2006. 7th International Conference on*. 2006, .
15. Welbourne, E., Lester, J., LaMarca, A., Borriello, G.. Mobile context inference using low-cost sensors. In: *Location- and Context-Awareness*; Lecture Notes in Computer Science. 2005, .
16. Lee, Y.S., Cho, S.B.. Human activity inference using hierarchical bayesian network in mobile contexts. In: *Neural Information Processing*; Lecture Notes in Computer Science. 2011, .
17. Kullback, S., Leibler, R.A.. On information and sufficiency. *Annals of Mathematical Statistics* 1951;.
18. Maggiore, G., Spanò, A., Orsini, R., Costantini, G., Bugliesi, M., Abbadi, M.. Designing casanova: A language for games. In: *ACG*; Lecture Notes in Computer Science. Springer; 2011, .