Beyond Location Check-ins: Exploring Physical and Soft Sensing to Augment Social Check-in Apps

Kiran K. Rachuri[1,2], Theus Hossmann[1,3], Cecilia Mascolo[1], Sean Holden[1]
[1]University of Cambridge, UK; [2]Samsung Research America, USA; [3]Swisscom, Switzerland
{kiran.rachuri, theus.hossmann, cecilia.mascolo, sean.holden}@cl.cam.ac.uk

***Abstract*—Smartphone sensing research has been advancing at a brisk pace. Yet, current social networking services often only take advantage of location sensing: applications like Foursquare use the phone's GPS and Wi-Fi radios to infer the user's location to simplify checking-in to a place. However, smartphone sensing could be exploited to considerably expand the spectrum of information a user can share with a few clicks with friends: not only the location of an event but activities such as "cooking dinner" or "waiting for a bus" can be predicted and suggested to the user to ease the check-in process. In this paper we show how mobile phone sensing can be used in this sense. For this prediction process to be accurate however, sensors need to be sampled often, with a considerable impact on the phone battery. To alleviate this issue, we explore streams of phone usage data (*soft sensors*), such as application usage, messages, and phone calls for predicting the user's activity in a more efficient fashion for augmenting mobile social check-in apps. We have deployed our application and collected a dataset of over 2700 check-ins to 48 activities from 20 users. Our analysis shows a prediction accuracy of 75% when offering 5 check-in suggestions to users. Furthermore, we show that when using only soft sensors we can achieve very similar performance to that obtained with real sensors, thereby significantly reducing the impact on the phone battery. This finding might have a potentially high impact on smartphone based activity check-in apps.**

I. Introduction

The unprecedented growth of Location Based Social Networks (LBSN) such as Foursquare and Facebook Places has fulfilled the desire of millions of users to share their current context with their social circles using smartphones. Motivations for context sharing range from *social*: signaling to friends and discovering new friends, over *personal*: life-logging and quantified-self, to *gaming*: points, mayorships and badges [1]. Yet, there is more to human activity than just location, indeed studies [2] have shown that users wish to share more about their daily context than just the restaurant or bar they visit. On a more fine-grained and less location-centric level, a user could signal that she is *at work drinking a coffee* so that her colleagues may join her or that she is *waiting for a bus* so that her family may want to call during this idle time. In spite of the enthusiasm of users to share context, a cumbersome check-in process (e.g., a long list of potential places to check-in) can discourage users from frequently sharing updates using these apps. Indeed, check-in fatigue [3] is a known problem in LBSN applications causing users to abandon the service and thereby adversely affecting the revenues of the service provider. Even if users check-in often, the service providers still need to build models to generate revenue from these check-ins. Given the advances in the mobile advertising space, one could envisage a lucrative business model utilizing activity check-ins of users. Although, a major obstacle in realizing this would be to gauge how trustworthy a user's check-in is, as the user might have already ended the activity or checked-in to a wrong activity.

The overarching goal of our work is to considerably expand check-in applications beyond just location, by addressing the check-in burden and fatigue problems while minimizing the impact on the phone's battery. To this end, we explore the automatic prediction and suggestion of activities to check-in to, by exploiting the phone's sensors, including a variety of "soft sensors" (i.e., phone usage logs). We design features based on smartphone sensor data and explore machine learning models to predict the user's activity, which can then be used by social check-in apps. To demonstrate this, we have developed *up2*, a social networking mobile application, which uses the concept of *check-in* to allow users to easily share their current activity on a fine-grained level. The activities we consider are not limited to physical activities unlike most existing work [4], but more fine-grained and diverse, e.g., eating, reading, watching TV. In fact, with *up2*, users can select among 47 pre-defined activities, *as well as define their own activities*.

In order to enable easy and quick check-ins, suggestions must be ready as soon as the user starts the check-in process (typically by clicking on a check-in button). As sensor sampling duration of typical activity recognition systems are in the order of seconds (e.g., GPS, accelerometer), initiating it after the user starts the check-in process will incur a high delay. The phone's sensors, therefore, must be sampled often in the background to have the suggestions ready when the user wants to check-in. Continuous sensing, however, leads to faster battery depletion. We, therefore, explore the feasibility of using low energy cost data streams such as phone calls, messages, app usage to suggest activities, which we refer to as *soft (software) sensors*. The intuition behind using soft sensors is that a user's phone usage in many cases has a correlation with her everyday activities. E.g., a user might typically call her parents when driving home from the office, or might read news, e.g., BBC or CNN news apps, while she waits for her bus, or use an activity tracker app while running. The idea is to exploit these correlations to infer activities, as the energy cost of using soft sensors is negligible compared to that of physical sensors. Further, we also explore validating check-ins using smartphone sensor data, which is useful in cases where check-ins are directly tied to commercial incentives, e.g., a user might earn a discount by frequently checking-in to a food place, or to support targeted advertising, which is an important consideration for the service providers. Our work is one of the very few systems [5], [6] to exploit phone usage patterns to predict user activity and validate the user's check-in, and to the best of our knowledge, our work is the first to quantitatively compare the energy-accuracy trade-offs of physical and soft sensors. Our main contributions are the following:

- We present a framework, which expands check-ins to fine-grained activities by exploiting smartphone sensing for check-in recommendation and validation.

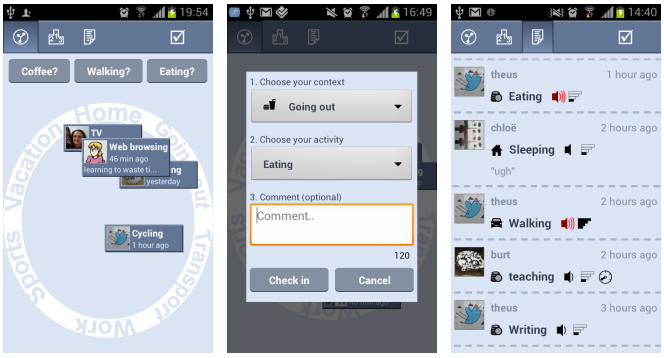
- We explore the feasibility of using low-cost data streams (*soft sensors*) for predicting the user's activity at negligible energy cost. Our results on the performance comparison of soft and real sensor features might have a potentially high impact on mobile social check-in applications.
- We report on a deployment of a context check-in application and analyze the collected dataset of about 2700 fine-grained activity check-ins. Our evaluation results show that we i) successfully suggest the activity a user will check-in to in about 75% of the cases (by offering as little as 5 suggestions out of a possible 48), and ii) identify 80% of fake check-ins at a false positive rate of less than 3%.

II. Related work and Goals

Activity and context recognition based on smartphone sensing has been widely explored. The phone's accelerometer has been proposed as a means to detect physical activity [4], [7] (e.g., walking, running). Other systems use the microphone [8], or combinations of different sensors [6]. Mun et al. [9] use accelerometer and location to detect transportation mode. Multimodal sensing, in particular the combination of GPS, accelerometer and mic, has been found effective in recognizing a variety of activities, beyond the relatively simple walking and running [10]. However, none of these works explored the feasibility of using smartphone sensing, especially soft sensors, to assist users in sharing their activities at a fine-grained level – we consider 48 activities in our evaluation. **Goal 1: Using the phone's sensor signals, infer the activity the user is about to check-in to**.

In order to suggest activities to check-in to, data needs to be captured from the phone's sensors and then processed. If these tasks are initiated after a user starts the check-in process, the high delays incurred might have a negative impact on the user experience. Therefore, like existing systems [4], [6], [9], a sensor assisted check-in application also requires background sensing as the application would not know when a user wants to check in. However, continuous background sensing leads to rapid depletion of the phone's battery thereby discouraging users from using these type of applications. Thus, researchers have devised mechanisms to limit the impact on the phone battery like duty cycling [4]. Our approach to achieve energy savings is very different from the existing work: instead of duty-cycling sensors, we use data streams that incur negligible energy cost such as phone calls or messages or app usage patterns to predict the user's activity. For example, using an application such as *runtastic* or *endomondo* might infer that the user might be running, or calling a specific friend (or a contact) might suggest that the user is at home. Although software data streams have been explored by some existing work [5], [11], none of them compared the energy-accuracy trade-offs of physical and soft sensors. Further, the problem we consider, i.e., predicting the user's check-in, is very different from these works. **Goal 2: To achieve energy efficiency, explore the feasibility of using the phone's soft sensor signals to infer the activity the user is about to check-in to**.

A pivotal business model of check-in applications is targeted advertising, e.g., advertising about "sports shoes" if the user is a frequent runner. More importantly, advertising at the right moment considerably enhances the chance that the user reads the advertisement, e.g., when the user is "waiting for bus". A related challenge, therefore, is verifying that the activity check-in indeed corresponds to what she is doing. "Invalid" or "Fake" check-ins may happen for various reasons, e.g., checking in after the activity has finished or a user may lie about the activity. Further, fake check-ins also impact the user-centric features of these apps such as friend recommendations and gaming. To mitigate the impact of fake check-ins, is thus our third goal. **Goal 3: Identify invalid or fake check-ins using the phone's physical and software sensor signals**.



(a) Friends' activities. (b) Check-in screen. (c) Past activities.

Fig. 1: *up2* screen shots.

III. Social Check-in Application

We built *up2*, a mobile social check-in app for the Android platform. The main goal of the app is not only to collect data from users but also to serve as a platform for implementing our check-in prediction scheme. Further, it uses many concepts from existing social networking services to motivate users to check-in more regularly so that it enables our research on check-in prediction. After installation, the user first sets up the profile consisting of user name, description and profile picture. To connect to their friends, users can search for user names and send friend requests. All the user's friends will be notified about her check-ins and comments. The main screens in *up2* are shown in Figure 1. To check in, the user either clicks on one of the suggested activities (the buttons shown in Figure 1a; for the data collection these are disabled or static) or, if none of the suggestions is correct, the user clicks the check-in symbol. If the check-in screen (Figure 1b) is accessed from the suggestion buttons, the context and activity are already filled in. Otherwise, the user first selects the context and then the respective activity. Additionally to the pre-defined activities in Table I, users can define their own activities and assign them to one of the default contexts.

The central part of the application for this work is the sensing service. We used the open source sensor libraries presented in [12] to implement the sensing service. The service captures data from various physical and soft sensors of the phone such as Accelerometer, Mic, GPS, Wi-Fi, Screen, Proximity, Application usage, Calls, SMS Messages, Battery, and Network information. The application collects data from the physical sensors only when the user checks in to reduce the impact on the phone's battery, whereas it continuously collects software sensor data. We store the sensor data temporarily on the phone in an SQLite database, which is uploaded asynchronously to a server for feature extraction. The server side of up2 is implemented on the Google App Engine.

Leisure	Home	Work	Transit	Sports	Vacation
Eating, Drinking	Cooking, Eating	Meeting, Reading	Walking, Cycling	Walking, Running	Sightseeing
Dancing, Date	Coffee, Housework	Writing, Telephone	Motorbike, Car, Bus	Cycling, Climbing	Shopping, Reading
Waiting, Coffee	Sleeping, TV, Gaming	Chatting, Coffee	Tram, Train, Plane	Football, Tennis	Hiking, Beach
	Shopping, Waiting	Waiting	Waiting	Swimming, Waiting	Waiting
	Hanging Out, Reading				

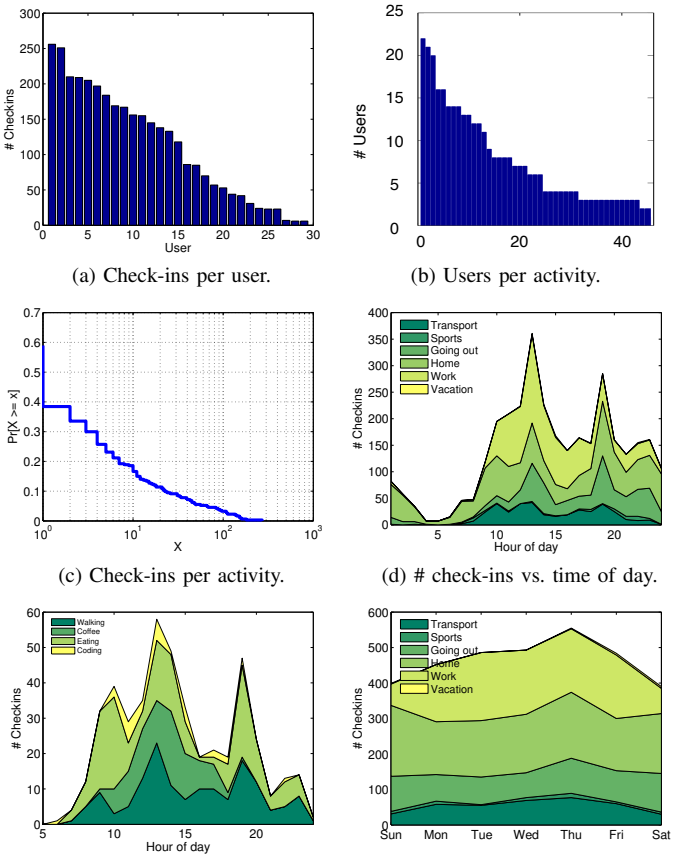
TABLE I: List of pre-defined activities.

IV. Application Deployment

In this section we describe the real-world deployment of the application described in the previous section. We recruited volunteers by advertising in social media such as Twitter and Facebook and circulating emails in different universities requesting participation. We also searched for volunteers by snowball sampling. We recruited a total of 29 users spread over 3 European countries and collected check-in data for about a month. We have disabled check-in suggestions for the data collection as it might introduce a bias in the check-ins.

Dataset. We collected over 3200 check-ins to 48 different activities. However, we observed a variance in the number of check-ins per user: Some users only provided around 20 check-ins, whereas some others contributed 200 and more check-ins. We removed check-ins with no associated sensor data and users with less than 50 check-ins, which resulted in over 2700 check-ins to 48 activities from 20 users. As described in the previous section, users can check in to predefined activities (Table I) or define their own. We have seen check-ins to 307 different activities, 260 of which are user-defined. User-defined activities range from "Studying" to "Web browsing" or "Procrastinating". Figure 2a shows the distribution of number of check-ins per user. Another observation is that there are differences in how users use the application. While some users report a wide range of different activities across different contexts, others only report a relatively small number of activities from few contexts (e.g., checking in only at home or at work). Further, we notice that not all users check-in to the same activities. In Figure 2b, showing the number of users per activity, we observe a long tail of activities with only few users. These observations on variance in user behaviour further emphasise the challenges in designing check-in predictors. Figure 2c shows complementary cumulative distribution function of the number of check-ins per activity. We see that relatively many activities have only a small number of check-ins. For 48 activities we have more than 10 check-ins. In the following we will focus on these 48 activities. Note that this set actually contains unique context-activity pairs, thus, for example, the activity "coffee" is contained three times, in contexts "leisure", "home", and "work" (Table I).

Temporal patterns. In human activity, daily and weekly patterns are often observed. We expect to see this reflected in the check-ins. Indeed, in Figure 2d, showing the context of check-ins depending on the time of day, we observe such patterns: in the morning and evening we have many check-ins at home, whereas work context is typical during business hours. Thus, the hour of day is a good predictor for the *context*. The question now is whether it is also a good predictor of the *activity*. In Figure 2e, we show the hour of day for the top 4 activities. We can indeed observe activity-specific patterns: for example "coffee" has a peak in the morning, whereas "eating" peaks in the early morning, around noon and in the evening. Looking at *weekly* patterns, in Figure 2f, as expected, we see more work check-ins during week days. Sports, transport and vacation contexts are rather independent of the day of the week. These observations are not surprising, but confirm our intuition that *hour of day and the day of the week are two strong predictors for activity*.



(a) Check-ins per user. (b) Users per activity.

(c) Check-ins per activity. (d) # check-ins vs. time of day.

(e) Time of day of top activities. (f) # check-ins vs. day of week.

Fig. 2: Check-in distributions and temporal patterns of users in the collected dataset.

V. Features and Models of Human Activity

In order to engage the user frequently, we aim to suggest a small set of activities in which the user quickly finds her current activity. In this section, we describe how we select the most meaningful features and use them in predictive models of human activity check-ins. Our goal is to keep the features relatively simple (to allow for feature extraction on the resource constrained mobile phone) and easily interpretable (to help us gain insights about mapping features to activities). We summarise all the features that we use in Table II.

Temporal Features. In the previous section we observed a strong link between the *time of day* and certain activities, which motivates the use of temporal features such as the hour of day (F^1) and the day of week (F^2) for check-in prediction.

Temporal features	
F^1 Time of day	$\{0, 1, \dots 23\}$
F^2 Day of week	$\{0, 1, \dots 6\}$
Physical sensor features	
F^3 Acceleration mean	$m_a = \frac{1}{N}\sum_i a_i$
F^4 Acceleration std. dev.	$s_a = \sqrt{\frac{1}{N}\sum_i (a_i - m_a)^2}$
F^5 Noise mean	$m_n = \frac{1}{N}\sum_i n_i$
F^6 Noise std. dev.	$s_n = \sqrt{\frac{1}{N}\sum_i (n_i - m_n)^2}$
F^7 Speed	$\frac{1}{t_N - t_1}\sum_{i=1}^{N-1} d(l_i, l_{i+1})$
F^8 Distance from home	$d(l_N, l_h)$
F^9 Distance from work	$d(l_N, l_w)$
Software sensor features	
F^{10} Previous check-in	{all activity IDs}
F^{11} Battery charging?	Boolean
F^{12} Battery level	$\{1, 2, \dots 100\}$
F^{13} Battery state	{low, medium, high}
F^{14} Network type	{Wi-Fi, cellular, none}
F^{15} Network name	String
F^{16} Last used app category	{app categories}
F^{17} # Proximity events	{1, 2, ...}
F^{18} # Screen events	{1, 2, ...}
F^{19} # SMS events	{1, 2, ...}
F^{20} # Phone calls	{1, 2, ...}
F^{21} Recent SMS/Calls?	Boolean
F^* Temporal features	F^1, F^2

TABLE II: List of prediction features.

Physical Sensor Features. From the accelerometer sensor, we obtain time sequences of vectors containing acceleration in the x, y and z directions, logged with a frequency of about $5Hz$. We first compute the undirected acceleration from these vectors $a_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$. From the sequence of accelerations we then compute the *mean* (F^3) and the *standard deviation* (F^4). Similarly, we compute the mean of the sequence of noise levels n_i (F^5) and their standard deviation (F^6). Using the phone's GPS and network location, we obtain a sequence of N timestamped latitude/longitude readings l_i. Since the contexts home and work are, for many users, bound to a single location, we extract the distance to home location l_h (F^8) and to work location l_w (F^9) using *the Haversine formula*. We extract the home and work locations from the latest home and, respectively, work check-in. Using the location of the last check-in has the advantage that this will adapt quickly if the user changes work location (e.g., during a conference) or home location (e.g., when moving) or even has several work/home locations. Further, in order to detect activities involving movement, we are interested in the speed with which the user travels. For this, we sum the distances between subsequent location readings, and divide this sum by the time difference between the time of the last location reading t_N and the time of the first t_1 (F^7).

Software Sensor Features. We take a broad definition of software sensors, i.e., data streams in the phone that do not incur any additional cost to capture them unlike physical sensors. For example, Screen on/off events are already logged by the phone and the incremental cost for an application to register for these events is negligible. Similarly, proximity sensor events, application usage, phone calls, SMS messages could be captured without much impact on the phone's battery as these are already captured by the phone's operating system. On the other hand, a physical sensor such as accelerometer or microphone needs to be actively queried by expending battery energy to capture data from it. Software sensors can provide us with information about the user's context, for example, using maps when driving or charging phone when at home or listening to music when travelling, i.e., there might be a correlation between the way the user uses her phone and her activity, and we aim to exploit this correlation.

We use several software sensor features to predict the user's check-in such as whether the phone is charging or not (F^{11}), battery level (F^{12}, F^{13}), its network state (F^{14} and F^{15}), number of recent screen, proximity and communication events (F^{17} to F^{21}). We define "recent" as occurring in the preceding one hour before a check-in. We also use the activity label of the *previous check-in* (F^{10}) of the same user, to account for typical sequences of activities. Thus, the previous check-in feature takes values from the set of all activity IDs. This feature is undefined for the user's first check-in. Further, we define a set of application categories such as music, news, social, games, transport, etc. and use the category of the last used application before check-in as one of the features (F^{16}). If an application does not fall into any of these then we categorise it as "other". The intuition behind this feature is that there might be a correlation between the application usage and the user's context, for example, playing "Angry Birds" during lunch or when at home. We also use temporal features as part of software sensors as they too are negligible cost data streams. In the evaluation, however, we also compare the performance of software sensor and temporal features to understand the performance gain of software sensor features over that of temporal. We normalise all numeric features to zero mean and unit variance.

Prediction Models. Given the features and activity labels from the past, we are facing the *supervised learning* problem of building a model that predicts the labels of new check-ins. What makes our problem particularly challenging is the large number of classes that we must be able to predict. In order to find a model which is able to exploit these patterns, we empirically evaluate ZeroR, Naive Bayes, Decision Table, Decision Tree, and Random Forest classifiers using the WEKA machine learning library [13].

VI. Predicting the user's check-in

In this section, we empirically evaluate the predictive power of different types of features using various models and measure the predictability of different activities. Considering that users have their own specific routines, we also create user-specific models to predict check-ins of users.

A. Prediction Accuracy

In this section, the questions of interest are: i) *What is the prediction accuracy that can be achieved with various feature sets (temporal, physical, and software)?*; ii) *How do the different prediction models compare to each other in terms of prediction accuracy?*; iii) *How does the predictive power of the types of features relate to their energy cost, especially physical sensors?* We define the prediction accuracy as percentage of correctly predicted check-ins out of all predictions that we make. Thus, the problem here is: given the set of N training

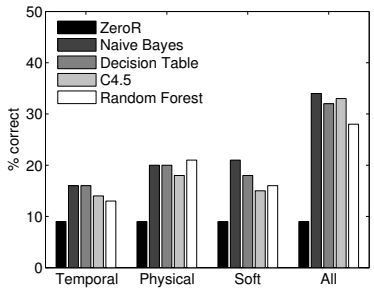


Fig. 3: Prediction accuracy (% of correctly predicted activities) of various feature sets.

feature vectors $f_1 = \{f_1^1 \dots f_1^n\} \dots f_N = \{f_N^1 \dots f_N^n\}$, learn a mapping of features to activity classes: $g : F \rightarrow C$. We evaluate each of the classifiers described in the previous section using the widely accepted *stratified 10-fold cross validation* on the entire deployment check-in data collected.

1) Prediction Power of Features: To get an idea of how well the different features predict activities, we first group the features into three types *temporal*, *physical sensors*, and *soft sensors* (as shown in Table II), and measure the prediction accuracy when only one recommendation could be given. The result of this evaluation is shown in Figure 3. First, we note that the trivial classifier (ZeroR) predicts 9% of the labels correctly since the top activity accounts for about 9% of all check-ins (see Table I). Note that this is independent of the selected features, since ZeroR just serves as a baseline simply predicting the most likely class.

For the temporal features, we see that the Naive Bayes classifier and decision table achieve the highest accuracy, with 16% of correctly predicted check-ins. Given the large number of classes we predict (48 activity classes), correctly inferring higher percentage of check-ins than the baseline is promising with relatively simple temporal features. The relatively good predictive power of the temporal features confirms the very strong temporal patterns that we have discussed in Section IV. We, however, note that this value is too low to be of much use in practice. The sensor features achieve better accuracy (20% with the Naive Bayes and 21% with the random forest). One of the most interesting results is the performance of software sensors, i.e., they achieve an accuracy of 21% with the Naive Bayes classifier. Considering that physical sensors have been widely used for predicting many activity types in the existing work [4], this is an interesting result and shows that there is a correlation between the user's context and her interaction with the phone. We achieve almost 21% accuracy in the best case by using only software sensor features. However, the intuition is that by combining the features, this increases further. All classifiers are able to improve on the trivial ZeroR by factors of 3 to 4. Further, we observe that the decision tree, the decision table, and the Naive Bayes model perform best, with slight benefits for the Naive Bayes model (34%). Given the very large set of sometimes similar activities, correctly predicting 34% of the check-ins is a good start, however, this should be further improved to use it on users' phones. We address this issue in the next sections. Based on the results presented so far, we have chosen *Naive Bayes model* for the rest of this evaluation. Naive Bayes has two main advantages over the competing decision tree: i) training its classifier is cheap ii) by design, it gives a probability distribution over all classes, which is particularly useful in suggesting *sets of activities* (i.e., activities with highest probability), as well as for verification, where we can exclude very low probability labels.

B. Check-in Suggestions

The goal of activity suggestions is to enable a fast user check-in. Suggesting only a single activity would lead to considerable error. Given that there is an increasing trend to move towards 4-5 inch screen sizes, (e.g, the Samsung Galaxy S5, the iPhone 6), 3-5 check-in suggestions would very easily fit a typical phone screen and it would be reasonable to let the user choose from such a small list. Thus, what is needed is a prediction of a small *set* of most likely activities to make the user choose from. In this case the learning problem is $g : F \rightarrow S$, where $S = \{s_1, s_2 \dots s_n\}$ is a set of n suggestions and s_i is a suggested activity from the set of all activities $\{c_1 \dots c_{48}\}$. Since the Naive Bayes classifier computes a class probability $p(c_i)$ *for each activity* $c_i \in \{c_1 \dots c_{48}\}$ for a given check-in, we choose to suggest the n activities with the highest class probability. The suggestion is *successful* in the case that the user-reported class is in the set of suggestions.

For evaluation, we measure the percentage of successful suggestions, depending on the number of suggestions n. Figure 4 shows the results of this evaluation. As a baseline, the figure shows the ZeroR classifier, which, for a given n, suggests the n most frequently reported activities. We see that ZeroR starts around 9% for one suggestion and then steadily improves up to 31% when suggesting 5 activities. Our Naive Bayes model outperforms this by far: the model that uses all features reaches 60% accuracy with 5 suggestions, twice the accuracy of the baseline model. One of the highlights of this result is that the software sensor features achieve an accuracy of 54%, which is close to that of the model with all the features. Given the battery limitations of smartphones and the high energy demand of phone applications, this result could pave the way for applications which currently use physical sensors for predicting the user's context to start using soft sensors too. Another important observation is that initially the performance steeply increases with the number of suggestions. However, already after 4 or 5 suggestions, the curve flattens. *Thus, a relatively small number of 4 or 5 suggestions is a good compromise offering high accuracy, yet not overloading the user interface as they can easily fit on a typical phone screen.* In the next section we show how to further improve the accuracy by exploiting user specific models.

C. Subject Specific Models

Human behaviour is different from one individual to the next. For example, some users might read news (on phones using news applications such as "BBC News") during lunch and some others might read them at home in the evening. Further, based on when and where they have "coffee", each user might produce a different physical sensor pattern like noise levels, physical activity patterns. Intuitively, a user specific model that is trained only with the user's data may perform better than the global model that is trained with data from all users. Yet, for user-specific models, a significant amount of training data per given user is needed. The questions we seek to answer here is i) *How much training data per user is necessary for*

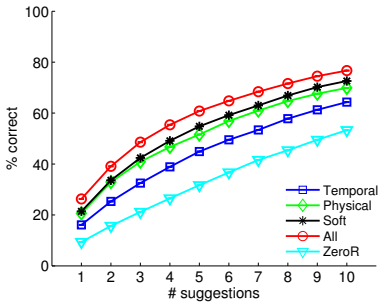


Fig. 4: Prediction accuracy of various feature sets vs. number of suggestions.

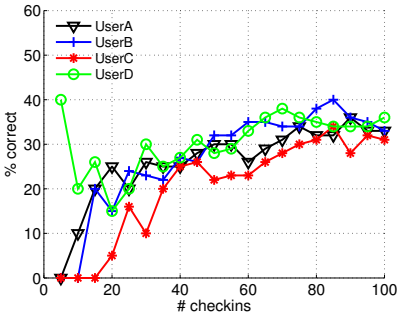


Fig. 5: Prediction accuracy for one suggestion vs # check-ins available for training.

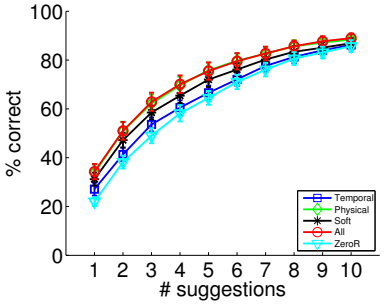


Fig. 6: User specific model: Prediction accuracy vs. # of suggestions.

successful user-specific models? ii) What is the performance gain of using user specific models over the generic model?

To evaluate this, we select the top four users from our deployment dataset and create models based only on their data. Figure 5 shows the prediction accuracy of the Naive Bayes classifier (trained only based on their individual check-ins) for one check-in suggestion, depending on the amount of training data (in number of check-ins). There are several important observations from this figure. First, there is a steep ramp up phase, during which the prediction accuracy dramatically improves with the number of check-ins used for training. After about $50 - 60$ check-ins, for all the users, the prediction accuracy stabilizes. Thus, we clearly do not need the entire history of a user's check-ins to make good predictions; a window of $\sim 60 - 70$ samples seems sufficient. Note that the exact value may vary, as different users check-in to different numbers of activities. Second, we observe that the accuracy does not steadily increase, but sometimes slightly decreases over time. We attribute this to changing user behavior, which is not yet reflected in the model when accounting for too much historical data. Finally, individual users are predictable to different degrees. This observation is related to the fact that some users check-in to a wider range of activities than others, however, after about 60 check-ins, the prediction accuracy using one suggestion is mostly stable. In general, global models are useful when a user joins the system (very little or no data is available for this user), while the accumulated history of the user should progressively allow some user profiling, after which, user-specific models could be used.

To understand the performance gain of subject specific classifiers, we evaluated their performance with respect to number of suggestions for each of the users. Figure 6 shows the result of this evaluation. We can observe that the performance of all the feature sets increases using subject specific models compared to that of their corresponding generic models (see Figure 4). The performance of the subject specific model that uses all features, for 5 recommendations, is 75%, which is 24% more than that of the corresponding generic model. This clearly shows that subject specific classifiers perform much better than a generic model. *Another important observation here is that, for 5 recommendations, the model using software sensor features achieves an accuracy of 72%, which is very close to that of the model with all features. This result is very promising considering the substantial energy gain of using software sensors over physical sensors.*

Activity	P	R	F
Walking (Transit)	0.82	0.86	0.84
Cycling (Transit)	0.81	0.87	0.84
Drinking (Leisure)	0.78	0.83	0.80
Coffee (Work)	0.71	0.88	0.78
Meeting (Work)	0.69	0.90	0.78

TABLE III: Most predictable activities (physical sensors).

Activity	P	R	F
Eating (Leisure)	0.64	0.71	0.67
Hanging out (Home)	0.71	0.64	0.67

TABLE IV: Least predictable activities (physical sensors).

D. Predictability of Activities

Intuitively, different activities are predictable to different degrees and based on different features. Some activities (e.g., eating, sleeping) have clear temporal or soft sensor patterns, whereas other activities, such as walking, have clear accelerometer patterns. We now investigate the predictability of the individual activities using physical and soft sensor features. This analysis provides important insights, for instance, about the limitations of various feature sets.

To be able to compare between the predictability of individual activities, we find it useful to adjust our learning problem. Instead of predicting class labels from the set of all classes $C = \{c_1 \dots c_{48}\}$, we now define a *per activity* binary classifier c_i for the top 15 activities as we have more check-ins for these activities (see Figure 2). Each binary classifier predicts whether a given feature vector belongs to the corresponding activity class c_i or whether it belongs to the set of all other activities, which we call $\bar{c_i}$. Thus, for activity c_i, our classifier learns the mapping $g_i : F \rightarrow \{c_i, \bar{c_i}\}$. The main challenge faced by these classifiers is that our training data is slightly *imbalanced*: for a large number of activities only a small percentage of training samples belong to c_i, while the vast majority belong to $\bar{c_i}$. A commonly used approach to address this problem is sampling [14]: we up-sample rare activities (i.e., sample the class of rare activities with replacement) and down-sample the over-represented class to have a balanced dataset. For our binary classifiers g_i, we take 100 samples from activity c_i and the same number of samples from activities $\bar{c_i}$. On these 200 samples, we then use leave-one-out cross-validation and measure the *precision (P)* and *recall (R)*. To compare the predictability of the classifiers, we combine these two metrics in the F-measure (ranging from 0 to 1, where 1 means a maximum precision and recall) and rank the classifiers g_i accordingly. Since the sampling introduces randomness in the data, we report average performance over 100 runs.

Activity	P	R	F
Coffee (Work)	0.74	0.87	0.80
Coffee (Home)	0.70	0.83	0.76
Meeting (Work)	0.68	0.83	0.75
Drinking (Leisure)	0.70	0.80	0.75
Reading (Home)	0.72	0.76	0.74

TABLE V: Most predictable activities (soft sensors).

Activity	P	R	F
Cycling (Transit)	0.63	0.73	0.68
Writing (Work)	0.64	0.70	0.67
Walking (Transit)	0.66	0.67	0.67
TV (Home)	0.62	0.71	0.66
Eating (Leisure)	0.64	0.65	0.64

TABLE VI: Least predictable activities (soft sensors).

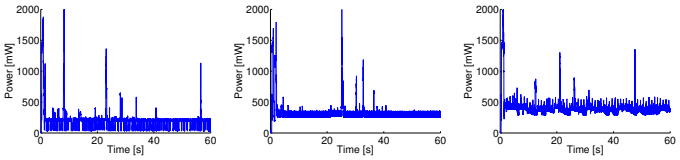
Tables III and IV list the precision, recall, and F-measure of the most and least predictable activities using only physical sensors. The first observation is that, in general, we reach quite high F-measures across *all* activities. With the Naive Bayes classifier, even the least predictable activity, "Hanging out" (at home), still reaches $F = 0.67$. A second observation is that the most predictable activities using physical sensors are physical activities from contexts "Transit" that have a strong correlation with physical sensors such as accelerometer and GPS, whereas the least predictable are in the contexts without much of correlation. Moving onto soft sensors, Tables V and VI show that activities in "Home" and "Work" contexts which have a correlation with phone interaction or time of day are more predictable whereas outdoor activities such as "Transit" (Cycling) or "Leisure" are less predictable.

E. Energy Cost of Features

As each feature comes with a different energy cost, it is important to investigate the trade-off between its energy consumption and predictive power. The aim of this section is to understand how much energy we invest when using each feature. On the one hand, we will show that we can improve the prediction accuracy by combining different features. On the other hand, invoking more features also requires more energy, since several sensors will be used simultaneously. To quantify the energy consumption of the sensed features, we use the Monsoon Power Monitor[1]. We measure the approximate energy consumed during a one minute sampling interval on a Samsung Galaxy S2 smartphone, running Android version 4.0.4 (in flight mode, in order to prevent external effects as much as possible).

The cost of the soft sensor features is assumed to be negligible since these merely require looking up the last check-in or capturing already logged events such as phone calls/SMS messages or active applications. For physical sensor features, however, the cost is non-negligible. Moreover, since we use quite basic features and low-intensive classifiers, we assume that the feature extraction and processing is negligible. If the feature extraction and classification tasks are computationally expensive, then the respective energy costs should be added too. Figure 7 shows a typical power measurement for a one minute sampling window of the accelerometer, microphone, and GPS. We observe that the individual sensors differ in energy costs with the accelerometer being the cheapest and the GPS the most expensive. The total power consumption for physical sensor features (see Table II) includes energy consumption of accelerometer, microphone, and GPS. Table VII summarises the energy-accuracy trade-offs: it shows the average power consumed by the feature sets in milliwatts (mW) and the prediction accuracy obtained. The table shows that physical sensors are clearly a very expensive option. For generic models, the accuracy achieved using purely soft sensors is about 6% less than that of using both physical and soft sensors. Whereas, for user specific models, the accuracy achieved using purely soft sensors is only about 3 to 4% less than that of using both physical and soft sensors. Therefore, the considerable energy gain of soft sensors would make them a better option than physical sensors.

[1] http://www.msoon.com/LabEquipment/PowerMonitor/



(a) Accelerometer. (b) Microphone. (c) GPS.

Fig. 7: Power consumption of various physical sensors.

Features	Power	Accuracy (# Suggestions, Model type)			
	(mW)	(3, GM)	(5, GM)	(3, USM)	(5, USM)
Soft	~ 0	42%	54%	58%	72%
Physical	561	40%	51%	62%	75%
All	~ 561	48%	60%	62%	75%

TABLE VII: Power consumption vs. prediction accuracy of features (GM: Generic Model; USM: User Specific Model).

VII. Verification

With the insights from the evaluation, we can now solve another challenge: *verification* of reported activities, either to enable accurate targeted advertising or to identify fake check-ins. In this section, we explain how to use the devised classifiers to solve the problem of verification.

A user can choose among a set of suggestions or can check-in to a completely different activity of her choice. So the issue now is to check if the chosen activity is the real user activity, or the activity is fake, e.g., due to checking in to a wrong activity or checking in after the activity has ended or false check-ins. This becomes important if activity check-ins are attributed some sort of importance i.e. for advertisement, user profiling, and offers. We can frame the problem of verification as another variation of the activity inference problem. This time, given a feature vector and a corresponding activity label (assigned by the user when checking in), we must decide if the label is trustworthy: $g : F, C \rightarrow \{fake, real\}$; where $fake$ means that we reject the label (e.g., we do not assign the user credit for the check-in, or, we do not consider it for targeted advertisements) and $real$ means that we accept it. We use the class probability, which we obtain from the Bayesian classifier, to decide whether we reject or accept the label. We use the global classifier to obtain class probabilities for all classes C. Given the user-reported class c_i, we record the probability $p(c_i)$ and apply a threshold p_{th}: if $p(c_i) > p_{th}$, we accept the label (we assign $real$), otherwise we reject the label (we assign $fake$). Note that, to detect fake check-ins, we must use the

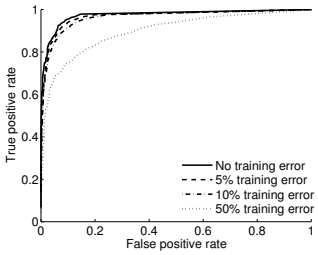


Fig. 8: ROC curve for verification performance.

global model, not the user-specific models. With user-specific models, a user could just systematically report wrong activities (e.g., always report "running" when in fact he is "walking") to fool the verification. The global model, however, consists of the behaviour of many users and cannot be easily fooled without some heavy collusion. Below, we also show that the global model is relatively robust, even if it is based on a large percentage of fake check-ins.

With the threshold p_{th}, we can now tune our system: A small threshold probability will guarantee that we do not wrongly invalidate a check-in (small number of false positives), but we might not detect all fake check-ins (high number of false negatives). Such a setting amounts to trusting the user. A high threshold, however, will cause a higher number of false positives but a smaller number of false negatives. This setting should be chosen in case it is essential to catch (almost) all fake check-ins. To evaluate this trade off, we use the class labels from our dataset as ground truth. We assign new (fake) class labels (randomly chosen among all classes) to 50% of the test check-ins. These are marked as fake and we measure how well our classification works to detect them using 10-fold cross validation. We note that random selection of fake label represents a beneficial case for detecting fakes. More severe attacks could for example try to systematically report fake activities that look similar to the ground truth from the sensor's perspective. In Figure 8 (solid line) we show the ROC curve. The ROC curve plots the true positive rate versus the false positive rate under varying threshold. As we aim for high true positive rate and low false negative rate, we want a steeply increasing curve. We observe that if we want to catch 80% of all fake check-ins (true positive rate of 0.8) the false positive rate is less than 3%. That means we will wrongly exclude a check-in in less than 3% of the cases. For a true positive rate of 0.7, the false positive rate is less than 1%.

Since we assume here that some users are cheating, we must also assume that fake check-ins could be used as training samples for the global model. In order to study the effect of fake training data on the verification performance, we artificially introduce fake labels: For 5%, 10% and 50% of the training samples for the global model, we randomly assign wrong labels. In Figure 8, we show how this affects the ROC curve. We observe that for 5% and even 10%, the effect is very small. Only when we introduce as much as 50%, the effect becomes severe. However, if we can assume that the system is used properly by the majority of users, the number of fake check-ins in the training data should be small. Summarising, by identifying check-ins where the user-reported activity class achieves only a small probability in our prediction model, we can successfully filter fake check-ins even if we have significant amounts of fake training data.

VIII. Conclusion

We have explored the use of smartphone sensing in applications where the user self-reports their current activity (e.g., to an online social network). In contrast to existing activity or context recognition systems, which continuously sense in the background to infer activity, we use the sensors to *support* the sharing of user-reported activity. To this end, we have presented an application, which exploits sensing in two ways: i) based on sensed features we provide suggestions that speed up the check-in process and thereby foster frequent user engagement, and ii) we use sensors to verify check-ins and prevent erroneous check-ins. We also use energy as a first order design consideration and showed that soft sensors are a viable option for predicting check-ins as they provide an accuracy close to that of physical sensors without impacting the phone's battery. This result has potential to pave the way to more diverse check-in applications beyond just location, as energy has been one of the biggest limiting factors for these applications. By collecting data from a real deployment, we found that by providing as few as 5 suggestions to the user, we correctly anticipate the reported activity in 75% of the cases using user-specific models, thus, simplifying the check-in process. With a modification of this model, we have also predicted unintended check-ins and found that we can identify most of them, with a small false positive rate.

References

[1] J. Lindqvist, J. Cranshaw, J. Wiese, J. Hong, and J. Zimmerman, "I'm the mayor of my house: examining why people use foursquare - a social-driven location sharing application," in *CHI '11*. ACM, 2011.

[2] F. R. Bentley and C. J. Metcalf, "Location and Activity Sharing in Everyday Mobile Communication," in *CHI EA '08*. ACM, 2008.

[3] "Check-in fatigue," http://www.techcrunch.com/2010/03/19/check-in-fatigue-location-war/.

[4] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell, "Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application," in *SenSys '08*. ACM, 2008.

[5] R. LiKamWa, Y. Liu, N. D. Lane, and L. Zhong, "MoodScope: Building a Mood Sensor from Smartphone Usage Patterns," in *MobiSys'13*. ACM, 2013.

[6] S. Nath, "ACE: exploiting correlation for energy-efficient and continuous context sensing," in *MobiSys '12*. ACM, 2012.

[7] N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman, "Activity recognition from accelerometer data," in *IAAI'05*, 2005.

[8] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell, "SoundSense: scalable sound sensing for people-centric applications on mobile phones," in *MobiSys '09*. ACM, 2009.

[9] M. Mun, P. Boda, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, and R. West, "PEIR, the personal environmental impact report, as a platform for participatory sensing systems research," in *MobiSys '09*. ACM, 2009.

[10] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh, "A framework of energy efficient mobile sensing for automatic user state recognition," in *MobiSys '09*. ACM, 2009.

[11] G. Chittaranjan, J. Blom, and D. Gatica-Perez, "Who's who with big-five: Analyzing and classifying personality traits with smartphones," in *ISWC '11*. IEEE, 2011.

[12] N. Lathia, K. K. Rachuri, C. Mascolo, and G. Roussos, "Open Source Smartphone Libraries for Computational Social Science," in *MCSS Workshop '13*. ACM, 2013.

[13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD Explorations Newsletter*, 2009.

[14] M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: one-sided selection," in *ICML '97*, 1997.