Stamina, a coffee monitoring system

Alexandra Rouhana [1], Lorenzo Scutigliani [2], and Quentin McGaw [3]

[1] Electrical and electronic engineering, Imperial College London
alexandra.rouhana12@imperial.ac.uk
[2] Electrical and electronic engineering, Imperial College London
lorenzo.scutigliani12@imperial.ac.uk
[3] Electrical and electronic engineering, Imperial College London
quentin.mcgaw12@imperial.ac.uk

28 March 2016

Abstract. Coffee drinking allows to reduce tiredness and is widely used in our society. In particular, the caffeine contained in coffee influences the quality of sleep of coffee drinkers. However, since each person reacts differently to caffeine, there is no rule of thumb to determine the optimal coffee intake schedule in order to maximise performance during the day while ensuring high quality sleep at night. Stamina is a solution providing this missing information to the coffee drinker, by monitoring and analysing the activity of the user over several days. It uses the sensors of both an Android smartphone and smartwatch to analyse the activity of the users, such as when they wake up, go to sleep or drink coffee. A remote server stores these data and analyses them with a clustering machine learning technique to provide suggestions to the users through their mobile devices about their daily coffee consumption.

Keywords: Caffeine, Sleep quality, Activity recognition, Smartwatch, Affinity propagation, Clustering

1 Introduction

Today's economy faces an increasing demand for products improving one's well-being and maximising physical and intellectual performance [6]. The aim of this project named Stamina is to improve users' health and performance by tackling a habit at the heart of their daily lives: coffee drinking. Stamina is a system which monitors and correlates coffee intakes, sleep patterns and daily activities to output to the end users whether or not they could drink a cup of coffee to optimise their sleep and activities. Stamina tracks the activities of its users through a mobile application analysing the data collected from the smartwatch's sensors and the smartphone's GPS localisation. The mobile application, called *Staminapp*, is compatible with smartwatches and smartphones with Android operating systems 5.0 or higher.

The design of Stamina is divided into three parts: the Android mobile application, the webserver and database, and the back-end machine learning program. This report explores the following topics:

- The selection process of a clustering algorithm adapted for the detection of patterns between multiple days, in order to determine various predictions according to the predicted *type* of day.
- The detection of daily activities with the mobile application *Staminapp* through the smartwatch sensors and the GPS capabilities of the smartphone. These activities can be sleeping, waking up, drinking coffee, performing a physical activity or entering *coffee zones* (a radius of 40 meters around the user's favourite coffee shop or place).
- The development of a secured webserver and database communicating with *Staminapp*.

This report first summarises the related work done in the coffee/sleep management area. Then, the facts and hypotheses Stamina is based on are stated. Afterwards the designed system is explored by describing its different components and justifying all design choices. Finally, the experimental procedure used to test Stamina is summarised and the obtained results are discussed together with their associated issues.

2 Related work

In the past decade, a lot of research focused on merging smartphone and smartwatch sensors data to detect the users' daily habits [1] [2] [3]. Classification algorithms such as the *Support Vector Machines* have been widely used to differentiate many real life activities from the smartwatch accelerometer and gyroscope sensors' data [4] [5]. Physical activities such as running, biking, walking, seating, and standing or even more complex activities such as drinking coffee can be detected through wearable device sensors [5]. Although the activity detection implemented in *Stamina* is relatively simple and does not include any classification algorithm (see part 4.3), it is definitely possible to enhance Stamina by implementing machine learning techniques for activity recognition. However, this project focused on correlating coffee intake with daily habits and sleeping patterns using machine learning. In particular, very little published research exists today in this domain.

Three similar products exist on the market: *UPCofee* by Jawbone, *Sen.se* [10] and *Caffeine Timezone 2* [8]. Only *UPCoffee* [9] uses analytics but no information on the algorithms used is currently available. *Caffeine Timezone 2* has been developed by Frank E. Ritter and Kuo-Chuan (Martin) Yeh from Pennsylvania State University [8]. This mobile application defines *sleep zones* in which the user will be able to sleep, and *active zones* where the user is stimulated by coffee. *Caffeine timezone 2* does not monitor sleeping patterns or activities automatically. The caffeine absorption and elimination models underlying the application are open source. The sleep zone and the active zone thresholds can be modified manually by the user. Caffeine levels are calculated each second and stored in a SQLite database. The evolution of vascular caffeine levels of the user is plotted each time the application is launched. According to the pharmacodynamics model used in *Caffeine Timezone 2*, caffeine elimination follows a nonlinear exponential equation shown below in equation (1) [8]. The caffeine calculations in Stamina is based on a similar equation.

$$(1): \ C(t) = C_0 \, exp(-\frac{t}{T}),$$

where C_0 and T corresponds respectively to the average caffeine quantity in a cup of coffee and to the caffeine elimination half-life.

The following table (Table 1) summarises the features of each of the previously stated similar existing products. Note also that *Sen.se* and *UP coffee* require extra devices, whereas Stamina focused on providing a solution using only an Android smartphone and smartwatch, hence being more affordable.

	Caffeine zone 2	Sen.se	UP coffee	Stamina
Automatic coffee drinking detection	NO	YES	NO	YES
Measure of sleep quality	NO	NO	YES	YES
Usage of machine learning	NO	NO	YES	YES
Automatic go to sleep detection	NO	NO	NO	YES
Automatic wake up detection	NO	NO	NO	YES
GPS coffee zones	NO	NO	NO	YES

Table 1: Features proposed by existing similar products

3 Facts and Hypotheses

Before describing Stamina any further, it is important to understand the underlying facts and hypotheses used to develop the project. There are mainly four medical facts Stamina is based on. First, each individual reacts differently to caffeine [18][22]. This highlights the need for a personalised coffee monitoring system such as Stamina. Secondly, the consumption of coffee (with caffeine) negatively influences the sleep of the user [18][19][20][21], suggesting that coffee should only be drunk after determining whether or not the sleep is compromised. Drinking coffee before a physical activity apparently enhances the physical performance of the person [18], which appears as a good thing. Drinking coffee is also considered healthy [23], as long as it does not deteriorate the sleep quality.

Concerning the assumptions, there are also four of them. First, it is assumed the sleep quality mainly depends on the consumption of coffee and not on other factors such as stress or drugs. Second, the users are assumed not to be sleeping or having poor sleep quality if they are moving while sleeping. This assumption allows to determine a sleeping quality percentage which is needed for this project. Days with similar wake up times and sleep durations are assumed to have similar physical activity times (if any) and sleep times. Finally, an obvious but necessary assumption is that the users are expected to wear their smartwatch most of the time and to always drink with the hand on which their smartwatch is attached. This

allows *Staminapp* to detect the user's' activities at all time, including the coffee drinking action.

4 System design

As explained in the introduction, *Stamina* is divided into three parts: the mobile front-end side, the remote server and database, and the machine learning program. Each part is described, justified and evaluated in the following sections.

4.1 Mobile part: Staminapp

The front-end consists of *Staminapp*, an Android application deployed on both smartphone and smartwatch. It is developed using Android Studio 2.0 with Java 7 because of the familiarity with Android development. It is compatible with Android operating system 5.0 and higher, which fully support Android smartwatches. The mobile part has two objectives, the first is to detect and confirm real life activities of the user, and the second is to exchange this information with the remote server to obtain useful information and suggestions related to the user's coffee consumption.

4.1.1 Detecting and uploading real life activities

The mobile application uses the GPS capabilities of the smartphone as well as the linear acceleration and gyroscope integrated sensors of the smartwatch to detect movements associated to the user's real life activities, according to the following conditions:

- **Go to sleep movement**
  The smartwatch's linear acceleration decreases significantly in the last 30 minutes; The current time must match a meaningful sleep time.
- **Wake up movement**
  The smartwatch's linear acceleration increases significantly in the last 30 minutes; The current time must match a meaningful wake up time.
- **Sleep quality**
  While sleeping, each time the smartwatch's linear acceleration changes from zero with a minimum increase/decrease of 0.2 $cm.s^{-2}$, the user is moving either because he is in a light sleep phase or is simply not sleeping. Each time this occurs, the sleep quality percentage is decreased by 1%. The current time has to be after the sleep movement and before the detection of the wake up movement.
- **Physical activity movement**
  The smartwatch's linear acceleration reaches a high threshold on any of the three axes more than four times in 10 minutes. The threshold was experimentally found to be $90cm.s^{-2}$.
- **Drinking movement**
  The smartwatch's linear acceleration and gyroscope data during the last 10 minutes contain a similar pattern within a margin of error according to a experimentally predefined drinking model with a trial and error method.
- **Coffee shop**
  The smartphone's GPS coordinates are contained in a circular area with radius 40m centered around the GPS location of the coffee place. This area is called a coffee zone; This event is triggered each 3 hours if the user is in the same coffee place and is not sleeping.

Regarding the physical activity and drinking movement detections, these are notified to the user who has to confirm them through interactive notifications, since these could simply be other unrelated events such as drinking water.

The linear acceleration is chosen to be used in most of the cases as it is straightforward to understand and to interpret its data. This type of sensor is similar to an accelerometer except that constant forces such as gravity are not present in the 3 dimensional data. On the other hand, the gyroscope is used to detect the drinking movement as the data obtained from the linear acceleration sensor are not sufficient to accurately detect a drinking movement.

After testing out the application for longer times (see part 5), the battery life was found to be terrible, especially on the smartwatch. To be battery efficient, *Staminapp* uses two techniques respectively for the smartphone and the smartwatch. On the smartphone, as the GPS is the power-hungry component, it is important to minimise its use with the lowest accuracy to reduce the battery usage. Staminapp acquires the GPS location of the user only each minute and only if the GPS detects that the user moves more than 20m. This makes the GPS active only each minute keeping the accuracy to the bare minimum, hence enhancing the battery life of the smartphone.

On the smartwatch, the Staminapp service runs in the background and drains the battery in just a few minutes

by constantly polling information from the sensors, using the main system on chip (SOC). In fact, the main SOC is extremely power hungry and is not designed to stay "awake" all the time. To fix this, the *Batching* technique was used. It basically stores the sensors' events in a shared hardware FIFO (First Input First Output). This technique is programmed such that the data generated by the integrated sensors are stored in this low-energy FIFO and the SOC wakes up from its sleep state only every 10 minutes to gather the 10 minutes long data and analyse it. Whilst being battery efficient, this last technique unfortunately brings a movement detection delay which can be up to 10 minutes. To mitigate the inconvenience of this delay, the notification asking the user to confirm an event also contains a time of when that event occurred, which could potentially help the user recall the last drunk coffee for example.

Concerning the user location feature of *Staminapp*, GPS coordinates and names of public *coffee zones* such as *Starbucks* are currently hard coded, but should ideally be automatically retrieved through the Google Maps API. Similarly, user-defined coffee zones are currently hardcoded in the application but should ideally be selectable on the map provided by *Staminapp*.

4.1.2 Server communication and notification

Once any of these events is detected and confirmed, *Staminapp* automatically uploads the event details together with a timestamp and a session cookie identifying the user to the remote server. This is done over a secure HTTPS link with a POST request. Depending on the event, different responses are obtained, each triggering different actions described in the table below (Table 2).

Event uploaded	Response from server	Triggered action on Staminapp
Go to sleep	Acknowledgement	None
Physical activity	Acknowledgement	None
Wake up	Expected time for your eventual physical activity of the current day. (see the machine learning part)	Sets an alarm trigger to suggest the user to drink a coffee 30 minutes before the expected physical activity time through a notification
Drinking coffee	Status of the coffee being drunk: is it the last one for today or not	Displays a temporary note on the bottom of the screen to inform the user
Entering a coffee zone	The user can still drink coffee OR can drink his last coffee today OR should stop drinking coffee	Displays a corresponding notification on the smartwatch and smartphone to inform the user

Table 2: Table of the events uploaded, responses received and corresponding actions triggered

Note that all these exchanges are explained in more details in the overview technical video of Stamina [31]. Since the main focus of the project was to design a background real life activity detector showing simple but useful information and suggestions to the user, not many resources were invested in developing the user interface. However, even though not necessarily appealing to the eye, the developed mobile application still provide all the functionalities described above.

4.2 Back end of Stamina: server

The back-end of *Stamina* is essentially an online centralised *nodeJS* webserver linked to a *mysql* database and to a *python2.7* interpreter used to execute the machine learning scripts. The following section describes and justifies all server related choices.

4.2.1 Server model

The back-end part of Stamina is built around a remote server, where the client is the mobile application Staminapp. Initially, processing and storing the data locally in the mobile application was considered as an option. In fact, this structure provides both offline access and privacy, but also eliminates the need for network communication. The service availability can't be compromised and the battery consumption is reduced thanks to the absence of internet connection. However, this idea was quickly discarded since offline data storage and authentication is generally recognised as insecure especially when storing data belonging to others (i.e. machine learning might need data from other users). Therefore, since security is the main concern in applications handling medical data [12], storing and processing data on the server while the machine learning results are only presented to the client was considered instead. In this way, securities threats can be reduced and the application becomes more portable and compatible. However, these improvements are obtained at the expense of offline access, bandwidth usage and battery life.

As a result, since this medical application requires secure authentication and data storage while minimising bandwidth usage and allowing offline access, a *balanced* or *hybrid* server-client model is preferred. To achieve this the client needs to filter and process the raw data obtained from the user input and hardware sensors into compressed packages only containing relevant data for machine learning purposes (as highlighted in the previous section). Then on the server, these packages can be stored and organised into a database where machine learning algorithms can readily access them to send relevant suggestions back to the client.

4.2.2 Database design

In order to store data in an organised and accessible way from the server, a database is required. In particular, there are two different typologies of data storage: relational (SQL) and non-relational (NoSQL) [15] [16]. For this project data stored in the database need to be regularly accessed by the machine learning algorithms. Therefore, to facilitate the insertion and selection of data in the system, relational databases are preferred since they can be easily accessed using SQL commands. Moreover, given the limited amount of time to design the system, the theoretically obtainable increase in storage efficiency from designing a personalised NoSQL database is not achievable in practice [16]. Hence, a relational database is used for storing users' data. Specifically, MySQL is chosen since being optimised for web server.

4.2.3 Web framework choice

There exist numerous web frameworks to develop a web server, which can be divided based on the programming language used. The following table summarises the most popular web frameworks for different programming languages (Table 3):

Language	Web frameworks
Python	web2py, Django, TurboGears
Java	JSF, Spring MVC, Vaadin
JavaScript	Node.js (Express.js)
Ruby	Ruby on Rails, Lotus
PHP	Symfony, Laravel, CodeIgniter

Table 3: Most popular web frameworks

Overall, these above listed frameworks are all well documented with a solid user base and exhaustive range of functionalities (i.e. request/response handler, https, cryptography, etc.). Therefore, the main discriminators are the programming language and performance.

Now, since the machine learning algorithms are decided to be developed in Python mainly because of specific libraries, the obvious choice would be a Python framework (i.e. *web2py*, *Django*, etc.). However, since learning a completely new framework would considerably extend the project's development time, the more familiar Node.js framework is chosen instead. *Node.js* together with the *Express.js* module provides all needed functionalities for the project including secure client-server communication with SSL (*https.js* module), data/credentials encryption with *AES-512* and *SHA-512* respectively [13] (*crypto.js* module [14]) and python script inclusion for machine learning algorithms (*python-shell.js* module). Furthermore, since running on the V8 JavaScript Engine and because of its non-blocking event driven structure, *Node.js* is currently one of the most performant server-side environment especially for high levels of concurrent connections.

4.2.4 Stress testing

In order to ensure the server's ability to successfully handle concurrent connections, an HTTP/HTTPS stress tester named *siege* is used [17]. Given a specific HTTP/HTTPS request, this tool sieges the target server with a specified number of concurrent users and reports the percentage of successful transactions. The performance of the server under stress is hence done with the following shell command:

siege –c*N* –t10s https://localhost:3000/insert POST action=coffee&time=123456789, where **-c** is the number of concurrent users N, and **–t** is the amount of time the test runs for.

This request was chosen since it uses nearly all available functionalities offered by the server, including database read/write operations and machine learning processing using the python module. Executing this command for different values of N, the threshold below which the percentage of successful transactions constantly stays at 100% is determined to be around 120. Further analysis showed any transaction failure above this threshold is caused by the python module. In fact, other requests not involving the Python machine learning program can withstand up to 10,000 concurrent users. The server is thus able to run with up to 120 active concurrent users. However, in order to accommodate a greater user-base, the machine learning integration (the python module specifically) will have to be redesigned. A possibility would be to compile the python code in JavaScript directly rather than interpreting it at run time.

4.3 Back end: Machine learning program

4.3.1 Design overview

In order to give a recommendation on the caffeine intake, the vascular caffeine level is estimated by the python module given the previous caffeine intakes of the current day, using the pharmacodynamics model described by the equation (1). The clustering

algorithms enable to tailor generic caffeine models to the user's daily habits. The python program detects similarities between different days and clusters them according to the occurrence of a physical activity, similarities in coffee intakes, wake up and sleep time. We assume that for each type of day, or cluster, the caffeine level at sleep time will have a different impact on sleep quality. The clustering algorithm is implemented in python due to its productivity and to the large variety of accessible machine learning libraries. Although the performance of Python is not as good as lower level languages such as C/C++, it is more than sufficient for our purpose. The *scikit-learn* library is used to implement the clustering algorithms because of its very detailed documentation and because of the wide range of clustering algorithms offered [11].

Due to the lack of research material about applying machine learning to detect patterns in daily habits, determining the best clustering algorithm for this specific purpose required testing different algorithms. In addition, a small sample data set containing entries for a month was built from scratch due to the absence of an open source database related to the effect of coffee consumption on sleep and activity. The dataset contains the following information for each day: the different times of coffee intakes, the sleep time and the wake up time, the time of a physical activity and the sleep quality. Note that all these times are recorded in epoch time standard.

4.3.2 Evaluation of clustering algorithms

Since clustering uses unsupervised algorithms, it is less straightforward to evaluate them than evaluating supervised algorithms. In order to compare algorithms, it is necessary to establish predictions on the similarities between members of the same cluster and verify which algorithm best satisfies these. The Python program includes a function that classifies days in the following categories: early or late wake up, early or late sleep time, long or short sleep duration and occurrence or not of a physical activity. The coherence of the clustering is measured by comparing the data set contained in each cluster to the predicted classification. Because we believe a type of day depends mainly on the wake up time and sleep duration of the user, a greater weighting was assigned to these two parameters whilst the sleep time and physical activity have lower weightings. This means the clustering algorithm will emphasize the differences in wake up time and sleep duration for its operation.

The three following clustering algorithms were tested: *K-means [26]*, *Mean Shift [7]* and *Affinity Propagation [25]*. Unlike the *K-Means* algorithm, the *Affinity Propagation* and *Mean Shift* algorithms do not require a predetermined number of clusters [25][11]. With our data, the *Affinity Propagation* converges to 5 clusters whilst the *Mean Shift* gives 6 clusters. In order to determine how many clusters were ideal to run the *KMeans*, the Silhouette coefficient was taken into account. The Silhouette coefficient is bounded between -1 and 1. A coefficient of -1 indicates that the clusters are incorrect, a coefficient of 1 indicates highly compact clusters, and a coefficient around 0 indicates overlapping clusters [25]. In our case, for less than 5 clusters, the Silhouette coefficient was below 10%, indicating overlapping clusters. Regarding the *K-Means*, more than 5 clusters showed overfitting as more than one cluster contained only one element. The ideal number of cluster for this algorithm is thus 5. A screenshot containing a Python console view of the clusters generated for each algorithm is available in the zip file provided with the submission of the project (in the Staminaputations folder).

When comparing the given clusters for the three algorithms, the most coherent algorithms regarding the classification we designed above are the *Affinity Propagatio*n and *Mean Shift* algorithms. For both algorithms, the data sets contained in each cluster belonged to the same category (for example "*early wake up, long sleep duration, no activity, early sleep time*"). The Silhouette coefficient for the *Affinity propagation* was higher than the one of the *Mean Shift*. This might indicate that the ideal number of clusters to classify days is 5. The Affinity propagation was therefore implemented since it gave the best results on our data set.

4.3.3 Evaluation of the back-end output

Once all the days are assigned to a cluster, the program assigns the most appropriate cluster to the current day according to its wake-up time and to the sleep duration of the previous night. In order to determine the ideal level of caffeine at the expected sleep time for the current day, the program does the following. It first finds the closest wake up time and the closest sleep duration in the previous days to the wake up time and sleep duration of the current day respectively. A first list of cluster label(s) is built where each label corresponds to a day with the closest wake up time, and another list is similarly created for the sleep duration. The cluster label appearing the most in

the concatenation of the two lists is the cluster temporarily assigned to the current day.

The expected sleep time is obtained by averaging the sleep times in the found cluster. The day with the best sleep quality in the cluster is used to determine the targeted level of caffeine when the expected sleep time occurs for the current day. A custom metric was designed to measure how accurately the selection of the most appropriate cluster is performed (refer to the function in the code *get_accuracy*).

5 Experimental setup and results

Stamina is currently working with the back-end running at *https://www.projectstamina-scutis.rhcloud.com/* and the Android mobile application installed on request. The first experiment to test the system as a whole was to replace the automatic detection of real life activity movements with explicit buttons in Staminapp. Each button triggers the rest of the system including the notifications and server requests. All the functionalities worked flawlessly and we switched back the real movement detection on.

The next step was to test it out for a few days. One of us hence used Stamina during three days in order to determine logical bugs, user interface issues and more practically to test out the battery life. Indeed, the main issue found was the battery life of the smartwatch which was drained in about 45 minutes. As previously explained, this was due to the fact that the sensors data were gathered by the main SoC (CPU) at a rate of 5 samples per second. The SoC could not go in its sleep state and was thus consuming a lot of power. The fix was to implement the batching technique (see part 4.1.1 for more details). Other than this, all the network communication, notifications and error handling worked as intended. A design flaw concerning the coffee zone notification was found. If the users stay in the same coffee place, Staminapp notifies them each three hours that they could drink a coffee. However, as the user-defined coffee place could be home, Staminapp should disable this feature whilst the user is asleep so the user is not woken up. As we did not have much time, we also added manually 30 days of data in the database in order to verify the machine learning was working as expected. The results obtained were quite coherent with the previously defined days.

Finally, the Android application Staminapp was provided to a civil engineering student with the moto 360 smartwatch. He used it during two days with some 30 days data we created with him according to his usual habits. We also hardcoded the places he usually goes to drink a coffee. Overall, he was satisfied with the experience, especially about the suggestion to stop drinking coffee and the GPS feature suggesting him that he could drink a coffee. However, the drinking movement detection was a bit frustrating as it was triggered even if he was simply touching his face with his left hand.

6 Conclusion

To conclude, a production-ready system was successfully developed to manage the coffee/sleep schedule of its users: *Staminapp*. The front-end consists of an *Android* mobile application deployed on both smartphone and smartwatch while the back-end relies on a centralised *nodeJS* webserver with a *MySQL* database and a *python2.7* interpreter for machine learning algorithms (the source codes are all on GitHub [28][29][30]. Overall, this application implements the core functionalities identified in the specifications including detection of sleep, coffee intake and physical activity, secure transfer and storage of these data to a remote database and also delivery of personalised coffee intake suggestions through machine learning.

However, *Staminapp* is far from being exhaustive in terms of functionalities and optimal regarding its suggestions. In particular, a number of future improvements are worth considering. First, the user interface is currently quite rigid and could be reworked into something more intuitive, dynamic and appealing to the users with additional features (graphs, statistics, fun facts, etc.). The coffee drinking action or physical activity recognition could also be improved in terms of reliability through the design of mobile local machine learning. Then, the coffee intake suggestions could be further personalised by adding to the machine learning algorithm additional parameters such as age, medical conditions (smoking, etc.) or cross-user patterns. Finally, to ensure customer lock-in a social aspect could be added to the application allowing users to share their position or coffee drinking habit.

7 Bibliography

[1] O. D. Incel, M. Kose, and C. Ersoy, “A review and taxonomy of activity recognition on mobile phones”, BioNanoScience, vol. 3, no. 2, pp. 145– 171, 2013.

[2] G. Bieber, N. Fernholz, and M. Gaerber, “Smart watches for home interaction services,” in HCI International 2013-Posters Extended Abstracts. Springer, 2013, pp. 293–297.

[3] S. G. Trost, Y. Zheng, and W.-K. Wong, “Machine learning for activity recognition: hip versus wrist data,” Physiological measurement, vol. 35, no. 11, p. 2183, 2014..

[4] S. Chernbumroong, A. S. Atkins, and H. Yu, “Activity classification using a single wrist-worn accelerometer,” in Software, Knowledge Information, Industrial Management and Applications (SKIMA), 2011 5th International Conference on. IEEE, 2011, pp. 1

[5] M. Shoaib, S. Bosch, H. Scholten, P. J. M. Havinga and O. D. Incel, "Towards detection of bad habits by fusing smartphone and smartwatch sensors," *Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on*, St. Louis, MO, 2015, pp. 591-596.

[6] Daniel Garrett,William H. Molloie**,**PWC,The global mHealth market opportunity and sustainable reimbursement models,2015

[7] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603-619, May 2002. doi: 10.1109/34.1000236

[8] Ritter, F. E.,Yeh, K.-C. M. (2011). Modeling pharmacokinetics and pharmacodynamics on a mobile device to help
caffeine users. In Augmented Cognition International Conference 2011, FAC 2011, HCII 2011, LNAI 6780, 528-535.
Springer-Verlag: Berlin Heidelberg.

[9] iTunes, UP Coffee from Jawbone, 2015, https://itunes.apple.com/fr/app/upcoffee/id828031130?mt=8

[10] Sen.se, The meaning of life, Manage Coffee Consumption, 2015, https://sen.se/store/apps/coffee/

[11] Scikit Learn, sklearn.cluster.AffinityPropagation, 2014, http://ww2.scikitlearn.org/stable/modules/generated/sklearn.cluster.AffinityPropagation.html

[12] Information Commissioner’s Office. Privacy in mobile apps: Guidance for app developers. (Dec 2013), https://ico.org.uk/media/for-organisations/documents/1596/privacy-in-mobile-apps-dp-guidance.pdf

[13] Christoph Hartmann. Encrypt and decrypt content with Nodejs. http://lollyrock.com/articles/nodejs-encryption/

[14] Node.js Foundation. Node.js v5.7.0 Crypto Documentation, https://nodejs.org/api/crypto.html

[15] O.S. Tezer. Understanding SQL and No SQL Databases and Different Database Models. (Feb 2014), https://www.digitalocean.com/community/tutorials/understanding-sql-and-nosql-databases-and-different-database-models

[16] Craig Buckler. SQL vs NoSQL: The Differences. (Sep 2015), http://www.sitepoint.com/sql-vs-nosql-differences/

[17] Jeffrey Fulmer, *siege - An HTTP/HTTPS stress tester*, http://linux.die.net/man/1/siege

[18] Ian Clark, H. P. Coffee, Caffeine, and Sleep: A Systematic Review of Epidemiological Studies and Randomized Controlled Trials, 2016

[19] Hindmarch, U. R. A naturalistic investigation of the effects of day-long consumption of tea, coffee and water on alertness, sleep onset and sleep quality, 2000.

[20] Hussam Sabba, L. S. The effects of coffee consumption on sleep and melatonin secretion, 2002.

[21] Micha Levy, E. Z.-K. Caffeine metabolism and coffee-attributed sleep disturbances, 1993.

[22] Denis M. Grant.Bing K. Tang, W. K. Variability in caffeine metabolism, 1983.

[23] Schilter, B, HolzhÄUser, D, Cavin, C. Health benefits of coffee. 19ème Colloque Scientifique International sur le Café, Trieste,May 2001 pp. 1-9.

[24] Jean-Louis G, von Gizycki H, Zizi F, Fookson J, Spielman A, Nunes J, Fullilove R, Taub H. Determination of sleep and wakefulness with the actigraph data analysis software (ADAS). 1996 Nov;19(9) 739-743. PMID: 9122562.

[25] Scikit Learn, 2014, sklearn.cluster, http://scikit-learn.org/stable/modules/clustering.html

[26] Brendan J. Frey, Delbert Dueck, Clustering by Passing Messages Between Data Points, Science 16 Feb 2007, Vol. 315, Issue 5814, pp. 972-976, DOI: 10.1126/science.1136800, http://science.sciencemag.org/content/315/5814/972

[27] Anil K. Jain, Data clustering: 50 years beyond K-means, Pattern Recognition Letters, Volume 31, Issue 8, 1 June 2010, Pages 651-666, ISSN 0167-8655, http://www.sciencedirect.com/science/article/pii/S0167865509002323

[28] Github link to Staminaputations (machine learning and data processing program), https://github.com/qdm12/Staminaputations

[29] Github link to Staminapp (Android mobile application), https://github.com/qdm12/Staminapp

[30] Github link to ProjectStamina (Webserver and database), https://github.com/scutis/projectStamina

[31] Dropbox link to Technical overview video home made, https://www.dropbox.com/s/essxtrspvrxs852/video.mp4?dl=0