Modeling Temporal Evolution in the Endomondo Fitness Data Set

Yashodhan Hemant Karandikar

Advisor : Prof. Julian McAuley

ABSTRACT

The goal of this project is to model temporal evolution of runners' capacity across workouts as well as within each workout. We study this temporal evolution over 2 predictive tasks - prediction of duration of the next workout given previous workouts and prediction of average and instantaneous heart-rates. We show that accounting for temporal evolution improves results on these predictive tasks on the Endomondo fitness data set [5].

1. INTRODUCTION

People work out with the goal of becoming fitter. Regularly working out enables a person to take on more challenging work outs. For example, a runner training for long distance running can gradually attempt runs of longer durations. Thus, the running capacity or the fitness level of the person increases gradually with practice. Modeling this fitness level can be useful in order to be able to predict the duration of the next workout that the runner attempts. Such a prediction can be useful both as a goal and as a guide to the runner while planning the run.

Alternatively, runners often monitor their heart-rate while running and set goals of keeping the heart-rate below or above a certain value, either for improving their fitness level or for their own safety. The heart-rate at future instants in the run depends on how tired the runner is currently, among other factors. Further, this relationship between the heart-rate and how tired the runner is, varies from one runner to another. For example, an experienced runner might experience only a slight increase in heart-rate as he or she is more tired, while the heart-rate of an amateur runner might shoot up soon after the run starts.

The examples above highlight the temporal evolution of a runner's capacity, both across workouts and within each workout. We attempt to model both these forms of temporal evolution in this work.

2. PREDICTIVE TASKS

In order to study temporal evolution of a runner *across* several workouts, we choose the following 2 predictive tasks as follows:

1. Given the distance d_i and duration T_i for each workout i among first n workouts of a user u and given the distance d_{n+1} for the $(n+1)$'th workout, predict the duration T_{n+1} of the $(n+1)$'th workout.
2. Given the distance d_i and average heart-rate H_i for each workout i among first n workouts of a user u and given the distance d_{n+1} for the $(n+1)$'th workout, predict the average heart-rate H_{n+1} of the $(n+1)$'th workout.

In order to study temporal evolution of a runner *within each* workout, we choose the following 2 predictive tasks as follows:

1. Given the first n instantaneous heart-rates of a workout, predict the $(n+1)$'th instantaneous heart-rate.
2. Given the instantaneous heart-rates of the first f fraction of a workout, predict the heart-rates of the remaining $1-f$ fraction of the workout. This task is a generalization of the first.

3. RELATED WORK

[9] describes 4 different models which account for temporal evolution of user expertise through online reviews on websites such as Amazon, BeerAdvocate, RateBeer, CellarTracker. [9] describes these models along 2 orthogonal directions: individual/community evolution and evolution at uniform / learned intervals, for a total of 4 models. Of these, the model for individual evolution at learned intervals is the most general. Individuals improve their physical capacities for working out at very different rates and independently of other users, unlike general evolving trends in entire communities of users. Hence, the model for individual evolution at learned intervals makes most sense in the context of temporal evolution of users' physical capacities for working out. The work in this project is based on this model.

Auto-regressive models [1] for time-series prediction are useful for the task of predicting the instantaneous heart-rate values in the remaining workout.

4. ENDOMONDO FITNESS DATA SET

In this section, we introduce the Endomondo fitness data set [5]. Endomondo [5] is a free app and website that allows users to keep track of their workouts. The data extracted from the Endomondo website [5] is available in the form of a SQL dump of size 200 GB approximately after compression. It contains data for around 10 million workouts. Data for each workout is in the form of a HTML document string. This HTML document contains information about the workout such as the sport (such as running, walking, cycling etc.), user id, total distance, duration, wind, weather, hydration, date & time etc. The HTML document also contains embedded JSON data. This JSON data contains GPS co-ordinates (latitude, longitude and altitude), distance, duration, heart-rate, speed / pace sampled throughout the duration of the workout. The 'duration' trace indicates the time instants at which the remaining attributes were sampled.

4.1 Extracting relevant data

We developed a parser in Python to extract the relevant data from the HTML document. We use parallelization to speed-up the extraction of the data. Each line of the SQL dump contains data for multiple workouts in the form of HTML documents. A 'master' process reads each line, extracts the HTML documents and hands them over to 'worker' processes. Each 'worker' process takes 1 HTML document at a time, parses it to extract the user id, sport type, trace data and the summarized information and encodes it into a dictionary in JSON format. The worker queues this workout data into a global queue. A separate 'writer' process dequeues workout information and writes it to a single file in the form of 1 line per workout. Having a separate writer process to write the workouts extracted by worker processes eliminates disk contention among workers. It also allows data for all workouts to be written to a single large file instead of millions of small files, without any need for synchronization between processes since only a single process is performing the writes. All of the extracted data is written to a file of size around 30 GB after compression.

4.2 Data set statistics

We list some basic statistics of the data set. Table 1 lists the total number of workouts, the total number of users in the data set and the average number of workouts per user.

Total number of workouts	10320194
Total number of users	2083006
Avg. number of workouts per user	4.95

Table 1: Number of workouts and users in the data set

Although the average number of workouts per user is only 4.95, the distribution of workouts among users is skewed, so that there are many users with more than 10 workouts. Table 2 lists number of users with more than 10, 40, 70, 100 workouts.

Table 3 lists the number of workouts for some of the activities (such as running, walking, cycling etc). The number of workouts for running is the highest and it is significantly more than that for other activities. Hence, in this work we use data from workouts for running only.

n	**Number of users with more than n workouts**
10	271004
40	11679
70	2191
100	768

Table 2: Number of users having more than 10, 40, 70, 100 workouts

Activity	**Number of Workouts**
Running	4413262
Walking	2154563
Cycling, sport	938108
Cycling, transport	816808
Mountain biking	429906
Step counter	265403
Fitness walking	172440
Hiking	147061

Table 3: Number of workouts by activity

5. BASELINE MODEL

This section describes a baseline model to predict the duration of a workout given the distance. Given the total distance d_{ui} for the i'th workout of user u, the predicted duration T'_{ui} of the workout is given by:

$$T'_{ui} = (\alpha + \alpha_u)(\theta_0 + \theta_1 d_{ui}) \quad (1)$$

We have one parameter α_u per user u, which is learned during the training using user u's workouts. The parameter α is global to all users. Thus we have a total of $U+3$ parameters, where U is the number of distinct users in the data set.

Let $\Theta = (\alpha, \alpha_1, ..., \alpha_U, \theta_0, \theta_1)$. We need to find Θ which satisfies the following objective:

$$\hat{\Theta} = \arg\min_{\Theta} \frac{1}{|D|} \sum_{T_{ui} \in D} (T'_{ui} - T_{ui})^2 + \lambda \|\Theta\|_2^2$$

where D is the data set of all workouts for all users, T_{ui} is the true distance of workout i of user u. The term $\lambda\|\Theta\|_2^2$ is a regularizer. λ is the regularization hyper-parameter, which we select by performing a line search over values in the set $\{0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\}$ and choosing the λ which minimizes the error on a validation set. We find the Θ which satisfies the objective using the L-BFGS [10] implementation available in SciPy [7].

If the task is to predict the average heart-rate instead of the duration, we use the same model specification as equation 1 by replacing T'_{ui} by H'_{ui}. Thus, the predicted average heart-rate is given by

$$H'_{ui} = (\alpha + \alpha_u)(\theta_0 + \theta_1 d_{ui})$$

6. TEMPORAL EVOLUTION OF USERS

This section generalizes the baseline model described in section 5 and attempts to account for temporal evolution of users across workouts. We describe the model, the training algorithm and finally inference.

6.1 Model Specification

In order to account for evolution in the fitness level or capability of a user over time, we associate a *experience level* or *fitness level* e_{uw} with each workout w of user u. This can be seen as a way of encoding how fit the user is at the time of the workout w. The value e_{uw} is an integer in the interval $[0, E)$ where E is the number of experience levels.

Intuitively, we expect the experience level of a user to either stay the same or increase with each workout. We encode this intuition in the form of a monotonicity constraint on the experience levels of workouts for each user, as given below [9]:

$$\forall u, i, j \quad t_{ui} \geq t_{uj} \implies e_{ui} \geq e_{uj}$$

where t_{ui} is the time at which workout i of user u occurs.

Then, given the total distance d_{ui} for the i'th workout of user u, the predicted duration T'_{ui} of the workout is given by:

$$T'_{ui} = (\alpha_{e_{ui}} + \alpha_{ue_{ui}})(\theta_0 + \theta_1 d_{ui}) \tag{2}$$

where e_{ui} is the experience level of the user u at the i'th workout. Thus, we have one parameter α_{ue} per user u per experience level e. Further, we have an intercept term α_e for every experience level e, common to all users. The terms θ_0 and θ_1 are global to all users and experience levels. Thus, given U users and E experience levels, we have a total of $UE + E + 2$ parameters.

Note that setting $E = 1$ reduces the model to the baseline model described in section 5.

We can specify a similar model for the predicted average heart rate H'_{ui} of the user u during the workout i:

$$H'_{ui} = (\alpha_{e_{ui}} + \alpha_{ue_{ui}})(\theta_0 + \theta_1 d_{ui}) \tag{3}$$

6.2 Objective Function

As explained above, we have a different model for each of the E experience levels. Each of the E models has the following parameters:

$$\Theta_e = (\alpha_e; \alpha_{1e} ... \alpha_{Ue})$$

where $1 \leq e \leq E$ is the experience level and U is the number of distinct users.

Thus, all the parameters in all of the E models together can be written as:

$$\Theta = (\Theta_1, \Theta_2, ..., \Theta_E)$$

Let β denote the set of all experience parameters e_{ui}. Then, we need to choose the optimal parameters $(\hat{\Theta}, \hat{\beta})$ according to the objective

$$(\hat{\Theta}, \hat{\beta}) = \arg\min_{\Theta, \beta} \frac{1}{|D|} \sum_{T_{ui} \in D} (T'_{ui} - T_{ui})^2 + \lambda_1 \Omega_1(\Theta) + \lambda_2 \Omega_2(\Theta, \theta_1) \tag{4}$$

$$s.t. \quad t_{ui} \geq t_{uj} \implies e_{ui} \geq e_{uj}$$

where D is the data set of all workouts for all users. This objective is similar to the one described in [9]. Ω_1 and Ω_2 are regularizers defined as follows:

$$\Omega_1(\Theta) = \sum_{e=1}^{E-1} \|\Theta_e - \Theta_{e+1}\|_2^2$$

and

$$\Omega_2(\Theta, \theta_1) = \theta_1^2 + \sum_{e=1}^{E} \|\Theta_e\|_2^2$$

Ω_1 is a smoothness function which penalizes abrupt changes between successive experience levels [9], since in practice similar experience levels should have similar parameters [9]. Ω_2 is another regularizer which penalizes complex models i.e. it penalizes models which have higher magnitudes of parameters. These regularizers are necessary to avoid overfitting, since we have a large number ($UE+E+2$) of parameters. λ_1 and λ_2 are regularization hyperparameters, which trade-off the importance of regularization versus prediction accuracy at training time [9]. We select λ_1 and λ_2 through a grid-search over values in the set $\{0.0, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\}$ and select the values which yield the highest prediction accuracy on a validation set.

For the task of predicting the average heart rate instead of the duration of the workout, the objective function remains the same as in equation 4, except that the term $T'_{ui} - T_{ui}$ is replaced by $H'_{ui} - H_{ui}$.

6.3 Training the model

We optimize the parameters Θ and β using a co-ordinate descent [3] i.e. we alternately optimize equation 4 for Θ given β and β given Θ. We optimize the model parameters Θ given β using L-BFGS [10] available in the SciPy library [7]. Optimizing β given Θ means assigning a experience level to each workout so that the mean-squared-error is minimized, subject to the monotonicity constraint [9]. Since the experience levels are discrete, this problem can be solved efficiently using dynamic programming, as is done in [9].

We alternately optimize Θ given β and β given Θ until β does not change or a maximum number of iterations has been reached.

6.4 Inference

Once the model parameters and experience levels have been learned, we infer i.e. predict the duration or average heart rate for the next workout w using equation 2 or 3. Note that we need to estimate the experience level for workout w in order to be able to use equation 2 or 3 to predict the duration or average heart-rate. We estimate this experience level in 2 different modes of inference, namely *final* and *random*.

In the *final* mode of inference, we use the first $N-2$ workouts as the training set, the $N-1$'th workout as the validation set and the N'th workout as the test set, where N is the total number of workouts for a user u in the data set. We assume $e_{u(N)} = e_{u(N-1)} = e_{u(N-2)}$ and use these experience levels to predict the duration (using equation 2) or average heart rate (using equation 3).

In the *random* mode of inference, we randomly select 1 workout as the validation set and 1 workout as the test set. The experience levels for these workouts is assumed to be equal to those of the workouts closest in time in the training set.

We present results for both modes of inference in section 9.

7. TEMPORAL EVOLUTION OF WORKOUTS

We now consider temporal evolution of a runner's capacity *during* a workout and use it to predict the runner's instantaneous heart rate in the future.

7.1 Model Specification

Given the heart-rate at various instants in a workout, we associate a *tiredness level* e_{wi} with heart-rate sample i in the workout w. As before, the value e_{wt} is an integer in the interval $[0, E)$, where E is the number of tiredness levels. We expect the tiredness level of the user to either stay the same or increase with each sample in the workout. We encode this in the form of a monotonicity constraint as before:

$$\forall w, i, j \quad t_{wi} \geq t_{wj} \implies e_{wi} \geq e_{wj}$$

where t_{wi} is the time at which sample i in workout w was sampled.

Then, given the distance covered d_{wt} at sample t for the w'th workout, the predicted instantaneous heart-rate h'_{wt} of the user is given by:

$$h'_{wt} = (\alpha_{e_{wt}} + \alpha_{ue_{wt}})(\theta_0 + \theta_1 d_{wt}) \tag{5}$$

where e_{wt} is the tiredness level of the user at the t'th sample in workout w. Thus, as before, we have one parameter α_{we} per workout w per tiredness level e and an intercept term α_e for every tiredness level e, common to all workouts. The terms θ_0 and θ_1 are global to all workouts and tiredness levels. Thus, given W workouts and E tiredness levels, we have a total of $WE + E + 2$ parameters.

7.2 Objective Function

As before, we have a different model for each of the W tiredness levels. Each of the E models has the following parameters:

$$\Theta_e = (\alpha_e; \alpha_{1e} ... \alpha_{We})$$

where $1 \leq e \leq E$ is the experience level and W is the number of workouts in the data set.

Thus, all the parameters in all of the E models together can be written as:

$$\Theta = (\Theta_1, \Theta_2, ..., \Theta_E)$$

Let β denote the set of all experience parameters e_{wt}. Then, we need to choose the optimal parameters $(\hat{\Theta}, \hat{\beta})$ according to the objective

$$(\hat{\Theta}, \hat{\beta}) = \arg\min_{\Theta, \beta} \frac{1}{|D|} \sum_{h_{wt} \in D} (h'_{wt} - h_{wt})^2 + \lambda_1 \Omega_1(\Theta) + \lambda_2 \Omega_2(\Theta, \theta_1) \tag{6}$$

$$s.t. \quad t_{wi} \geq t_{wj} \implies e_{wi} \geq e_{wj}$$

where D is the data set of all workouts for all users. This objective is similar to the one described in [9].

As before, Ω_1 and Ω_2 are regularizers defined as follows:

$$\Omega_1(\Theta) = \sum_{e=1}^{E-1} \|\Theta_e - \Theta_{e+1}\|_2^2$$

$$\Omega_2(\Theta, \theta_1) = \theta_1^2 + \sum_{e=1}^{E} \|\Theta_e\|_2^2$$

Ω_1 is a smoothness function which penalizes abrupt changes between successive tiredness levels [9], since in practice similar tiredness levels should have similar parameters [9]. Ω_2 is another regularizer which penalizes complex models i.e. it penalizes models which have higher magnitudes of parameters. These regularizers are necessary to avoid overfitting, since we have a large number ($WE+E+2$) of parameters. λ_1 and λ_2 are regularization hyperparameters, which trade-off the importance of regularization versus prediction accuracy at training time [9]. We select λ_1 and λ_2 through a grid-search over values in the set $\{0.0, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\}$ and select the values which yield the highest prediction accuracy on a validation set.

7.3 Training the model

The training algorithm is identical to the training algorithm described in section 6.

7.4 Inference

Once the model parameters and tiredness levels have been learned, we infer i.e. predict the instantaneous heart rate at 1 instant in each workout w using equation 5. As before, we need to estimate the tiredness level at that instant in order to be able to use equation 5 to predict the instantaneous heart-rate. We estimate this tiredness levels in 2 different modes of inference, namely *final* and *random*.

In the *final* mode of inference, we use the first $N-2$ samples in each workout as the training set, the $N-1$'th sample from each workout as the validation set and the N'th sample as the test set, where N is the total number of samples in a workout. We assume $e_{wN} = e_{w(N-1)} = e_{w(N-2)}$ and use these tiredness levels to predict the instantaneous heart-rate using equation 5.

In the *random* mode of inference, we randomly select 1 sample from each workout as the validation set and 1 sample from each workout as the test set. The tiredness level for these samples is assumed to be equal to those of the samples closest in time in the training set.

We present results for both modes of inference in section 9.

7.5 Prediction of several instantaneous heart rate values

We now generalize the problem of predicting 1 instantaneous heart-rate value to predicting the heart-rate values for the last f fraction of the workout. We use the same model as in section 7. Let e_{wn} be the tiredness level of the last sample in workout w the training set. In order to predict the instantaneous heart-rate values in the remaining part of the workout, we assume $e_{wn} = e_{w(n+1)} = e_{w(n+2)} = ... = e_{w(N)}$ where N is the total number of samples in workout w. In other words, we assume that the last tiredness level of the workout fitted during training stays the same for the remaining part of the workout. In practice, this is not a valid assumption, but this provides us with a conservative estimate of how well the model performs on the remaining part of the workout i.e. on the validation and test sets.

Also note that the difficulty is not only that of estimating the tiredness levels accurately, but also not having learned model parameters for tiredness levels beyond the last tiredness level fitted during training. Thus, even if the true tiredness levels for the remaining part of the workout are known, these cannot be used for prediction of heart-rate values.

8. IMPLEMENTATION DETAILS

This section describes some of the techniques used to verify correctness of the implementation and achieve reasonably good performance to allow running multiple experiments.

We verify that our implementations of derivative functions are correct by comparing their outputs to those obtained by computing numerical derivatives. We use functions provided in [7] and [8] to compute numerical derivatives.

We use Python for most of the data processing tasks, with NumPy [6] and SciPy [7] libraries. We observe that the training is the computationally most intensive part of the execution. We first speed-up the training using Cython [4]. This gives considerable speed-ups and scales well for the tasks of predicting duration and average heart-rate for the next workout of a user. However, even with this optimization, training still remained prohibitively expensive for the large data sets involved in the task of predicting instantaneous heart-rate values (see 9 for details of the data sets). Hence, we port the training part of these tasks to C++. This gives much higher speed-ups and enables the training process for a single run to complete in a few minutes on a personal laptop machine.

9. EVALUATION

This section describes the results of the models described in sections 6 and 7 on the predictive tasks described in section 2.

9.1 Temporal evolution across workouts

This section discusses results of the model described in section 6 on the tasks of predicting the average heart-rate and total duration of a user's next workout.

9.1.1 Prediction of average heart-rate

For this task, we consider only workouts of users having more than 10 workouts. Further, we discard workouts with invalid date/time, extremely long or short workouts and workouts with extremely low or extremely high average heart-rate value since these indicate workouts with invalid data.

Tables 4 and 5 show results for prediction of average heart-rate, using the *final* and *random* modes of evaluation, as discussed in 6. As discussed in section 6, we have 1 workout in the validation set and 1 workout in the test set for each user in the data set, for both modes of evaluation. The results are shown in terms of the coefficient of determination, R^2 [2], which is a measure of the fraction of variance in the data explained by the model. Table 6 shows the optimal values of hyper-parameters found using grid search.

As seen from the results, the model which accounts for temporal evolution of users does not perform better than the baseline model ($E = 1$). A closer look at the hyper-parameters resulting from the grid search, as shown in table 6, gives us some insight into this result. As is seen from table 6, grid search yields high optimal values of λ_1. Recall from section 6 that λ_1 is a hyper-parameter which penalizes drastic changes in parameters of similar experience levels. A high value of λ_1 means that the experience levels are very similar to each other and this essentially reduces the models with $E = 2$ and $E = 3$ to the baseline model $E = 1$.

	Training R^2	Validation R^2	Test R^2
# Examples	39545	2384	2384
Baseline($E = 1$)	0.520887	0.422324	0.423512
$E = 2$	0.520928	0.422081	0.423318
$E = 3$	0.520927	0.422085	0.423312

Table 4: Prediction of avg. heart-rate - *Final* mode of evaluation

	Training R^2	Validation R^2	Test R^2
# Examples	39545	2384	2384
Baseline ($E = 1$)	0.517636	0.481049	0.430987
$E = 2$	0.517645	0.480964	0.430880
$E = 3$	0.517692	0.480968	0.430886

Table 5: Prediction of avg. heart-rate - *Random* mode of evaluation

	λ_1	λ_2
$E = 2$ (final)	10^4	0.0
$E = 3$ (final)	10^5	0.0
$E = 2$ (random)	10^5	0.0
$E = 3$ (random)	10^4	0.0

Table 6: Optimal values of hyper-parameters λ_1 and λ_2 for prediction of average heart-rate

9.1.2 Prediction of duration

For this task, we consider only workouts of users having more than 10 workouts. Further, we discard workouts with invalid date/time, extremely long or short workouts (in distance or duration) since these indicate workouts with invalid data.

Tables 7 and 8 show results for the prediction of the duration og the workout, using the *final* and *random* modes of evaluation, as discussed in 6. As discussed in section 6, we have 1 workout in the validation set and 1 workout in the test set for each user in the data set, for both modes of evaluation. The results are shown in terms of the coefficient of determination, R^2 [2]. Table 9 shows the optimal values of hyper-parameters found using grid search.

The model which accounts for temporal evolution of users performs marginally better than the baseline model in the *final* mode of evaluation, but this is not true in the *random* mode of evaluation. Further, as is seen from table 9, grid search yields relatively high optimal values of λ_1, indicating that the experience levels are very similar to each other and this essentially reduces the models with $E = 2$ and $E = 3$ to the baseline model $E = 1$.

	Training R^2	Validation R^2	Test R^2
# Examples	743987	52109	52109
Baseline ($E = 1$)	0.785361	0.677276	0.696565
$E = 2$	0.775247	0.677833	0.714295
$E = 3$	0.782736	0.677098	0.718808

Table 7: Prediction of duration - *Final* mode of evaluation

	Training R^2	Validation R^2	Test R^2
# Examples	743987	52109	52109
Baseline ($E = 1$)	0.736994	0.627867	0.596432
$E = 2$	0.691232	0.627357	0.578873
$E = 3$	0.705088	0.634563	0.581045

Table 8: Prediction of duration - *Random* mode of evaluation

	λ_1	λ_2
$E = 2$ (final)	10^4	0.001
$E = 3$ (final)	10^2	0.001
$E = 2$ (random)	10	0.01
$E = 3$ (random)	10	0.01

Table 9: Optimal values of hyper-parameters λ_1 and λ_2 for prediction of duration of the workout

The lack of improvement in the prediction accuracy after accounting for temporal evolution across workouts seem to suggest that we need data for many more workouts per user and over much longer periods of time in order to be able to notice a reasonably strong evolution of the user across workouts.

9.2 Temporal evolution in workouts

We now discuss results for temporal evolution *within* workouts. First, we discuss results for the task of predicting a single instantaneous heart-rate value in each workout, followed by prediction of several instantaneous heart-rate values.

9.2.1 Prediction of 1 heart-rate value

For this task, we consider only workouts with at least 200 samples of instantaneous heart-rate values and distance values. Tables 10 and 11 show results for the task of predicting a single instantaneous heart-rate value in each workout, using the *final* and *random* modes of evaluation respectively. Accounting for temporal evolution within each workout significantly improves the prediction accuracy over the baseline ($E = 1$), for both modes of evaluation.

	Training R^2	Validation R^2	Test R^2
# Examples	24347765	83423	83423
Baseline ($E = 1$)	0.608874	0.585329	0.560480
$E = 10$	0.887163	0.839256	0.799058

Table 10: Prediction of instantaneous heart-rate - *Final* mode of evaluation

	Training R^2	Validation R^2	Test R^2
# Examples	24347765	83423	83423
Baseline ($E = 1$)	0.613856	0.608207	0.615789
$E = 10$	0.907340	0.894736	0.897766

Table 11: Prediction of instantaneous heart-rate - *Random* mode of evaluation

9.2.2 Prediction of last several heart-rate values

For this task, we consider only workouts with at least 200 samples of instantaneous heart-rate values and distance values. Given a workout of $N \geq 200$ samples of instantaneous heart-rate and distance values, we split N (distance, heart-rate) samples as training, validation and test sets of sizes $0.4N$, $0.3N$, $0.3N$ respectively. Table 12 shows results for the task of predicting last several instantaneous heart-rate values in each workout. The results indicate that accounting for temporal evolution significantly improves results compared to the baseline. We observe that the R^2 scores on the validation set are much higher than those for the test

set. This is expected due to the fact that the samples in the test set are further in time than the samples in the validation set, which in turn are further in time than the samples in the training set. Recall from section 7 that we assume $e_{wn} = e_{w(n+1)} = e_{w(n+2)} = ... = e_{w(N)}$ where N is the total number of samples in workout w and e_{wn} is the last sample in workout w in the training set. This assumption is lesser likely to hold for samples further and further in time. Thus, this assumption is less likely to hold for the test set than for the validation set. Hence, the R^2 scores on the test set are lower than on the validation set, which in turn are lower than on the training set.

	Training R^2	Validation R^2	Test R^2
# Examples	9841317	7352229	7321065
Baseline ($E = 1$)	0.567424	0.451540	0.356603
$E = 10$	0.911437	0.632749	0.464435

Table 12: Prediction of several instantaneous heart-rates in each workout

10. CONCLUSION AND FUTURE WORK

We describe and evaluate 2 different ways of accounting for temporal evolution of users' physical capacities for running workouts - one *across* workouts and the other *within* workouts. Results indicate that accounting for temporal evolution across workouts does not improve prediction accuracy, perhaps due to the relatively small number of workouts per user in the data set. On the other hand, accounting for temporal evolution *within* each workout significantly improves results over the baseline model.

Future work includes training and evaluating these models for other types of workouts, such as cycling, walking etc. The results on these types of workouts might give some interesting insights about the nature of these activities and users evolve with time in each of these activities. It might be interesting to repeat this work for a much larger data set with many workouts per user, since temporal evolution across workouts might be more prominent in such a data set. Models for time series prediction such as auto-regression [1] can also be used to predict the instantaneous heart-rate in the remaining workout.

11. REFERENCES

[1] Autoregressive Model. `http://en.wikipedia.org/wiki/Autoregressive_model`.

[2] Coefficient of Determination. `http://en.wikipedia.org/wiki/Coefficient_of_determination`.

[3] Coordinate Descent. `http://en.wikipedia.org/wiki/Coordinate_descent`.

[4] Cython - Working with Numpy. `http://docs.cython.org/src/tutorial/numpy.html`.

[5] Endomondo. `https://www.endomondo.com/`.

[6] Numpy. `http://www.numpy.org/`.

[7] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[8] D. E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[9] J. J. McAuley and J. Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *World Wide Web*, 2013.

[10] J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35:773–782, 1980.