Sheila Ramaswamy
Stephen Koo

Adaptive Spaced Repetition

I. Introduction

From weekly podcasts and active forums for polyglots to full language learning suites like Duolingo, different approaches and philosophies to language learning abound. However, little has been done to adapt language learning systems to the user and her learning patterns. We focused in particular on flash cards, a widely-used and powerful tool for memorization. A number of studies over the past century revealed that the productivity of memorization methods such as flash cards can be greatly augmented by spaced repetition: a technique that involves spacing reviews of previously learned material over successively longer time intervals, thus saving time by exploiting the fact that we take longer to forget something after each successful repetition. Moreover, computer programs now allow us to manage scheduling for an unlimited number of cards. Nevertheless even the best options available, such as SuperMemo and Anki[1], still employ simple algorithms that do little more than to increase the repetition intervals each time by a crudely derived coefficient. Meanwhile, the user's interactions with the system actually provide a large amount of potentially useful information, with which a smarter system could theoretically gain a better understanding of the user's mastery of the material. Through the use of machine learning techniques, we aim to build models of the relationship between user performance statistics and the card repetition intervals, in an effort to more accurately predict more optimal card display frequencies.

In this project, we use the published user logs of a flash card program called Mnemosyne as the basis for our preliminary data analysis and model training. We then build a working prototype of an online flash card system[2] that employs the trained model, while additionally updating its parameters with the new input from the online user, in order to progressively fit the model to the individual user's learning patterns. The algorithms we used in our system were largely provided by the scikit-learn machine learning Python package.

II. Data

Mnemosyne is a flash card application first developed in 2006 as free alternative to the proprietary SuperMemo program and simultaneously as a "research project into the nature of long-term memory."[3] Anonymized user logs are uploaded automatically—after initial voluntary consent—to a central database, which has then been published online periodically. We use a version released on January 14, 2014, which contains 48,965,391 log entries for flash card "repetition" (i.e. display) events[4]. Each log entry consists of information such as the user ID, timestamp, and the amount of time since the previous repetition of the particular card. For the purposes of this project, we evaluate the log entries two at a time: each training example consists of features extracted from a pair of consecutive log entries for the same card and user. The set of features are as follows (we will refer to the chronologically first entry as the "first repetition", and the second as the "second repetition"):

#	Feature	Description
1	previous grade	an integer score in the range [0, 5] provided by the user on each repetition as an assessment of his or her performance; taken from the first repetition
2	next grade	same as previous grade, but taken from the second repetition
3	easiness factor	a floating-point value computed from the cumulative history of grades on the given card, with higher values corresponding to higher overall grades[5], as computed prior to the second repetition
4	retention repetitions	the number of times the user has given herself a grade of 2 or higher the card on two consecutive repetitions, prior to the second repetition
5	acquisition repetitions	the number of times the user has given herself a grade of 0 or 1 on the card, prior to the second repetition
6	lapses	the number of times the user has transitioned from a retention repetition to an acquisition repetition, prior to the second repetition
7	retention repetitions since last lapse	the number of retention repetitions since the last occurrence of a lapse, prior to the second repetition
8	acquisition repetitions since last lapse	the number of acquisition repetitions since the last occurrence of a lapse, prior to the second repetition
9	thinking time	the amount of time in seconds that the user spent reading the card on the first repetition
10	interval	the interval of time in seconds between the two repetitions in the pair

Figure 1. Descriptions and enumerations of feature set.

[1] http://ankisrs.net/
[2] http://rocky-fortress-6883.herokuapp.com
[3] http://mnemosyne-proj.org/
[4] The logs can be found at: https://archive.org/details/20140127MnemosynelogsAll.db
[5] The exact formula is based on the SuperMemo 2 algorithm, which is described in detail at: http://www.supermemo.com/english/ol/sm2.htm

In some cases, we may discretize the interval on the scale enumerated in Figure 2, in order to smooth out nonlinearity and to reduce the amount of noise in the data. The interval ranges that we chose are based on the intuition that since forgetting curves appear to follow exponential decay, the repetition intervals should follow a roughly exponential growth pattern.

In addition, since using 48 million entries is excessive within our time constraints, we partition the data into individual datasets by user. We cap each dataset at 1000 training examples, and for each dataset, we hold out 200 examples as the test set, leaving 800 examples as the training set. We focus on one user at a time, since our goal is to train systems that can cater to the learning curve of each individual user.

III. Models and Analysis

Our main framework consists of predicting the interval given the other features described. We variously apply different prediction models, such as the Support Vector Machine (SVM), to generate predictors; to apply a predictor to scheduling the next repetition for a flash card, we compute all the features based on the user's history with the card so far, while artificially supplying a target grade as the "next grade." Thus the output interval should be the interval optimized toward helping the user obtain the given target grade on the next repetition. One may theoretically choose any target grade; in our online system, we arbitrarily chose a target grade of 4, which intuitively aims for comfortable recall without wasting time and making it too easy for the user.

Figure 2. Interval discretization scheme.

Rank	Features
1	next grade, easiness, retention repetitions, lapses, and retention repetition since last lapse
2	previous grade
3	acquisition repetitions since last lapse
4	acquisition repetitions
5	thinking time

Figure 3. Feature rankings.

Feature Analysis

As a preliminary analysis, we used a recursive feature elimination algorithm from scikit-learn, which uses recursive feature elimination, to determine an optimal set of features. Given an estimator, the procedure returns the optimal number of features along with a ranking of the given features. We chose the linear regression as the estimator for this analysis, because of its simplicity of interpretation: higher ranked features exhibit stronger (linear) correlations with the repetition interval. On our dataset of size 1000, we found the optimal number of features to be 5, with the rankings as listed in Figure 3. In all subsequent models, we thus only include the features in the top two rank categories in the feature set.

Linear Regression

As our first model, we again choose linear regression. We thus attempt to express the predicted interval $\hat{y}$ of the *i*th repetition pair as an affine function of the features f:

$$\hat{y}^{(i)} = w^T f^{(i)} + \beta$$

where w, β are parameters that minimize the least-squares residuals between the predicted intervals and the actual intervals in the training data. To predict an optimal interval for achieving a target grade g_{target}, we can pull the second grade out of the feature vector and rewrite the expression as:

$$\hat{y} = w'^T f' + w_{next\ grade} g_{target} + \beta$$

We computed the linear regression on a dataset for an arbitrarily chosen user. We recorded the mean square error (MSE) from after testing and the coefficient of determination (R^2) when testing the model on both the test and the training dataset.

Support vector regression

An alternative to simple linear regression that we tried is support vector regression (SVR). The SVR builds a model by solving the optimization problem:

$$\begin{gathered} \text{minimize } \tfrac{1}{2}\|w\|_2 \\ \text{subject to } y^{(i)} - w^T f^{(i)} \leq -\beta\varepsilon \\ \text{and } w^T f^{(i)} + \beta - y^{(i)} \leq \varepsilon \end{gathered}$$

where ε is a free parameter that serves as a threshold for true predictions. The resulting model, similar to SVMs, depends only on a subset of the training examples by ignoring any examples close to the model prediction.

Support vector machine

Finally, we train six different multi-class SVM classifier (SVC) models for each user, with each model trained only on data with the same value for the "next grade." This reflects the intuition that the distribution of repetition intervals may be conditionally independent given the value of the next grade. We also discretize the set of possible output intervals into 10 classes, as described previously. We use the "one-vs-one" variant of the multi-class SVM classifier, which optimizes the standard binary SVM objective for each pair k of the 10 classes:

$$\text{minimize } \tfrac{1}{2}\|w_k\|_2 + C\sum_i \zeta_k^{(i)}$$
$$\text{subject to } y^{(i)}(w_k{}^T f^{(i)} + \beta_k) \le 1 - \zeta_k^{(i)}$$
$$\zeta_k^{(i)} \ge 0,\ i = 1, \ldots, n$$

where we use a regularization coefficient $C = 1.0$. The following decision function for each *k*th binary SVM is then applied to the feature vector f:

$$\widehat{y_k} = sign(w_k{}^T f + \beta)$$

and the class that is predicted by the greatest proportion of the binary SVMs constitutes the overall prediction. Of course, the algorithms are actually implemented in terms of the dual form of the given primal objective in order to take advantage of the kernel trick. In our tests, we try the linear kernel, polynomial kernel, and the Gaussian (RBF) kernel with $\gamma = 1$ / # of features:

$$K_{rbf}(x, y) = exp\left(-\gamma\|x - y\|^2\right)$$

Clustering

Understanding that different people may have vastly differently learning curves, we speculated that training a single model as the prior for all new users on our system would not be very effective. We thus tried to cluster the users in our data based on their repetition logs—the resulting clusters could then potentially allow us to train multiple models, each conditioned on the user's closest cluster.

We computed three aggregate features for each user: retention rate, acquisition rate, and lapse rate. These are the ratios of their cumulative retention repetitions, acquisition repetitions, or lapses to their total number of repetitions, respectively. We ran *k*-means clustering on 500 users in the database using these features, with $k = 5$ clusters. However, we found no convincing clusters, and a visualization of the data corroborates the fact, as seen in Figure 4. We nonetheless train and test our models on data partitioned by the clusters, to see if accuracy could be improved by assuming independence of the parameters between clusters.

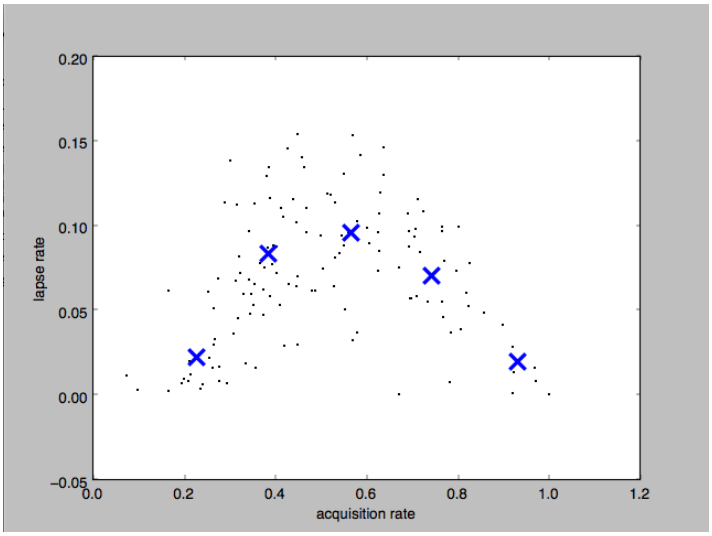

Figure 4. Plot of sample users along two dimensions in the space of aggregate features. There are no visibly discernible clusters.

IV. Results

Model	Data Description	Training Set Size	R^2 (train)	MSE (train)	Test Set Size	R^2 (test)	MSE (test)
Linear Regression	Data from user "c136315a"	800	0.76	18008435164 1	200	0.80	13375260536 6.86
SVR	Data from user "c136315a"	800	0.44	42286246360 5	200	0.70	20547305955 4.80
Linear Regression conditioned on lapse rate clusters[6]	Used data from 1 user in the cluster for training, another user from the same cluster for testing	~825[7]	0.356858	19586484614 71	~825	-1.1980369987 00247e+26	4.3373225003 02718e+37
Linear Regression conditioned on retention rate clusters	see above row	~825	0.348984	82876694741 69	~825	-9.1481441435 19614e+22	2.6460503827 832534e+32
Linear Regression conditioned on acquisition rate clusters	see above row	~825	0.376067	26623082966 9	~825	-4.0034716138 13629e+25	1.1422117452 866307e+35

[6] For this and all subsequent test, we only considered users that had at least 1000 samples (as they appear in the Mnemosyne logs) and had looked at at most 250 cards.

[7] Approximately 825 because we sampled random users from the logs for testing, ran the model 5 times, and took the average accuracy over all the clusters and runs.

Linear Regression conditioned on acquisition and lapse rate clusters	see above row	~825	0.417776	69478783762 3	~825	-9.7285644836 81107e+21	7.8693160929 70383e+32

Model	**Data Description**	**Training Size**	**Training Accuracy**	**Test Size**	**Testing Accuracy**
SVC Conditioned on lapse rate clusters	Used data from 1 user in the cluster for training, another user from the same cluster for testing	~825	0.762858	~825	0.432272
SVC conditioned on retention rate clusters	see above row	~825	0.768102	~825	0.480871
SVC conditioned on acquisition rate clusters	see above row	~825	0.793287	~825	0.531672
SVC conditioned on acquisition and lapse rate clusters	see above row	~825	0.738592	~825	0.528751
SVC conditioned on grade=0 (rbf kernel)	train with user “c6961489”; test with users “c136315a”, “c13786f5”, “c13b7ed6”, “c13e4eb3”, “c13fcf1b”	750	0.9545454545	2500	0.8026181484
SVC conditioned on grade=1 (rbf kernel)	see above row	750	0.9547325103	2500	0.747781175
SVC conditioned on grade=2 (rbf kernel)	see above row	750	0.6263157895	2500	0.2314695581
SVC conditioned on grade=3 (rbf kernel)	see above row	750	0.6161616162	2500	0.2904896581
SVC conditioned on grade=4 (rbf kernel)	see above row	750	0.8333333333	2500	0.2478075907
SVC conditioned on grade=5 (rbf kernel)	see above row	750	0.8214285714	2500	0.2209106099
SVC conditioned on grade=0 (linear kernel)	train with user “c6961489”; test with users “c136315a”, “c13786f5”, “c13b7ed6”, “c13e4eb3”, “c13fcf1b”	750	0.9431818182	2500	0.7887144051
SVC conditioned on grade=1 (linear kernel)	see above row	750	0.9547325103	2500	0.747781175
SVC conditioned on grade=2 (linear kernel)	see above row	750	0.5526315789	2500	0.3378802969
SVC conditioned on grade=3 (linear kernel)	see above row	750	0.6161616162	2500	0.3033893961
SVC conditioned on grade=4 (linear kernel)	see above row	750	0.8235294118	2500	0.2712045712
SVC conditioned on grade=5 (linear kernel)	see above row	750	0.8928571429	2500	0.262855215
SVC conditioned on grade=0 (poly kernel)	train with user “c6961489”; test with users “c136315a”, “c13786f5”, “c13b7ed6”, “c13e4eb3”, “c13fcf1b”	750	0.9659090909	2500	0.7889283089

SVC conditioned on grade=1 (poly kernel)	see above row	750	0.9588477366	2500	0.7047081302
SVC conditioned on grade=2 (poly kernel)	see above row	750	0.6421052632	2500	0.2373767905
SVC conditioned on grade=3 (poly kernel)	see above row	750	0.6464646465	2500	0.2841904955
SVC conditioned on grade=4 (poly kernel)	see above row	750	0.862745098	2500	0.2844512296
SVC conditioned on grade=5 (poly kernel)	see above row	750	0.9642857143	2500	0.2430891331

V. Discussion

The relatively decent coefficients of determination (0.80 on test set) from the linear regression indicate that there exists to some degree a correlation between the values of our features and the repetition interval. This provided the green light to continue work in this direction and extract as much worth as possible from the current feature set, and also gave a good baseline from which to start. However, the extremely high mean-squared errors also indicate that the predicted intervals are typically nowhere near the correct intervals. This may indicate that the relationship between the features and the repetition interval is not correctly characterized as linear (or affine). This may be addressed in a number of ways: add features that are nonlinear functions of the current set of feature to increase the expressivity of the model; or use a completely different model (perhaps a Generalized Linear Model) that more accurate captures the relationship between the features and the repetition interval. As mentioned previously, we directly "flattened" the nonlinearities in the repetition interval using a discretization scheme, transforming the problem into a classification problem instead of a regression problem. Moreover, the SVR model performed even worse than the naïve linear regression, thus we subsequently focused on undertaking the new classification problem.

The results of applying a multi-class SVM to the problem varied considerably depending on the setup. Our attempts to cluster users based on their lapse rate, etc., yielded few results. The SVC models trained on clustered data could not predict with an accuracy much better than half, which is likely because the clusters are not truly clusters to begin with. The lack of distinctive clusters is unsurprising, given that a feature space with three dimensions is typically hardly enough to cluster any significant set of samples. Moreover, a soft performance metric such as lapse rate is more likely to form a Gaussian distribution rather than separable clusters, as corroborated by our plots. This awareness, however, was helpful in selecting more "representative" users to select for training and testing our subsequent models.

Conditioning the models on the next grade provided great gains in the classification performance. We can see that depending on the given next grade and the chosen kernel, we can achieve up to 80% accuracy on the test set, while training accuracy is generally quite high regardless. The large discrepancy in performance between the test set and the training set indicates that the learned weights do not generalize well across users (since the datasets are partitioned by user). However, this is fine for our intended purpose, where we maintain independent models for each user, and thus "overfitting" to a single user is in fact necessary. In the future, it will then be important to use data points held-out from the same user as our test sets instead.

Overall, it is also understandable that our results, even on the training set, is at best only decent: we still only have a small number of features that don't necessarily have a linear relationship with the optimal repetition interval, and we may be able to do much better by building off of existing models of human memory in psychology literature. Nonetheless, we can safely conclude that it is possible to apply machine learning methods "out-of-the-box" to build a usable flash card scheduler.

VI. Future Work

If we have another 6 months to work on this project, we would first delve deeper into existing literature about human memory, language acquisition, and forgetting curves—we could then devise even more appropriate models for the quality of an individual's memory of a given item. We would also open up our application to more users, collecting more usage data and even collecting additional features (such as thinking time). We could also develop a way to assess a user's grade on a flash card that does not depend on their feedback, to make the calculation of the grade more deterministic. Finally, we need to develop better metrics for the quality of our models, e.g. at least incorporating recall, precision, and F1 scores into our classification evaluation.

VII. Additional References

P.A. Wozniak. (1998, May 10). *Application of a computer to improve the results obtained in working with the SuperMemo method* [Online]. Available: http://www.supermemo.com/english/ol/sm2.htm

Pavlik Jr., P. I., Presson, N., & Koedinger, K. R. (2007). *Optimizing knowledge component learning using a dynamic structural model of practice*. In R. Lewis & T. Polk (Eds.), Proceedings of the Eighth International Conference of Cognitive Modeling. Ann Arbor: University of Michigan.