Lomb

Predicting the Memory Curve in Text-based Language Learning

Overview

Language learning has benefitted significantly from the Information Age. SRS (Spaced Repetition System) learning apps, such as Anki, Memrise or Duolingo, have become very popular and have been shown to improve the memorization rate of vocabulary in several studies. In the SRS paradigm, a word or concept is prompted (in the form of a question, flashcard or exercise) at progressively longer intervals; an approach which mirrors Ebbinghaus' well-known Forgetting Curve.

However this approach is not without problems. The main problems are: 1) Since the algorithms work by scheduling review times, students must review when the algorithm requires them to and for the amount of time it requires them to. This lack of control is generally seen as undesirable: often people don't study full time but must work or study and therefore have varying amounts of time through out the week to devote to doing reviews, or conversely they might wish to devote more review time before an upcoming exam. To illustrate, failing to do reviews on a weekend will result in having to devote three or four times more time to doing reviews next time. 3) Typically students, particularly intermediate and advanced students, will attend classes, read, and generally have much exposure in the target language. The SRS model has no way of accounting for this additional exposure, even when user activity data is available, and in this situation leads to mostly redundant and unnecessary reviews.

For the latter reason, intermediate learners often get bored with SRS apps and devote more time to reading, watching films or videos and generally increasing their exposure in the target language. They sometimes use assisted reading apps (like LWT or LingQ), dual language subtitles or popup dictionary extensions for their browsers. Due to the lack of intermediate-level material in many languages, or simply lack of material that is appealing to learners, this is often done using native-level media.

This approach is also problematic. Although intermediate learners might be able to understand 60% of the words in a given sentence, much of the meaning will be contained in new words which must be learnt. But which words to learn? Which words to revise?

For this research, we developed an app which can be used to both read books and review vocabulary. The data collected from user reading activity is used to predict how likely a user is to forget a word. Our model differs from SRS in several key ways: - Accounts for additional vocabulary exposure outside review sessions. - Adapts to learner's desired review time.

Data collected

On reading, the following data is logged for each word that is read : `EXPOSURE_AND_NO_LOOKUP`, `EXPOSURE_AND_SENTENCE_LOOKUP`.

On review we collect `REVIEW_CLICKED` and `REVIEW_NOT_CLICKED`.

Activity logged for each word can therefore be summarised into a sequence, which can be of arbitrary length. Each element of the sequence will contain the following variables:

REVIEW_NUMBER, SECONDS_SINCE_LAST_REVIEW, CLICKED_LAST_REVIEW, EXPOSURES_SINCE_LAST_REVIEW_WITH_LOOKUP, EXPOSURES_SINCE_LAST_REVIEW_WITHOUT_LOOKUP, SECONDS_SINCE_LAST_EXPOSURE_WITH_LOOKUP, SECONDS_SINCE_LAST_EXPOSURE_WITHOUT_LOOKUP, TIME_SPENT_LOOKING_AT_DEFINITIONS_LAST_REVIEW, CLICKED_LAST_REVIEW

The program keeps a database with these sequences. There is also a `next_review` entry which is created whenever a `REVIEW_CLICKED` or `REVIEW_NOT_CLICKED` event is received, and which is updated whenever the reading data of that word is received.

A LSTM is trained using this data. Hopefully, it can predict something!!

Keywords

- Personalized learning.
- Computer testing
- CSC Computer Supported Cooperative work
- CHI Computer Human Interaction
- Psychology journals.
- Transaction on learning technology
- Education technology
- Adaptive learning (that seems to be more of a ML term)
-

2020/05/08

I wrote the first service of the Lomb application. It is the tracking, which is the most important part. Initially it had terrible performance, because every time it received a request it would convert between the MongoDB and a dataclass. I rewrote the domain and db modules so that they are now together (yes, coupled). This allows me to execute the MongoDB queries, particularly the updates, directly. CPU usage went from ~30% to ~10% by doing this.

Stuff to do:

- Implement word lookups in the reader by dividing the window into three parts (similar to the words view).
- Look into why the word cumulative percentage counts are wrong (they clearly are).
- Implement a users service to handle registrations, logins and tokens.
- Implement a library. This should just keep texts as arrays of chunks (locations). It could also do all of the heavy NLP lifting and store that for each location. This would make a lot of sense for stuff like filtering by tag.
- Implement a word service. Among other things, this keeps the known words for users, so that the frontend can filter words sent to tracking.

- Implement a reader which grabs known words, text (the array of locations and tokens, etc.). I guess the most performant way to make this work is to shift a lot of logic to the frontend. Otherwise Python will have to do a lot of work...
- Implement a simple RNN or LSTM with the data so that I go through the process.
- Implement other algorithms to see how they work.

2020/05/09

Being *lazy*...

Word state:

- Known / ignored (can be an `ignore` collection).
- Learning / suspended (add a new field to the `pending_reviews` model)

To populate the examples, NLP, frequencies:

- Add as you go.
- Have an option somewhere to populate from text.

To search by NLP:

- Have a custom dictionary with the wiktionary and dict.cc databases. Do the NLP on the fly.

2020/05/10

Some research questions:

- **The (Vocabulary) Knowledge Tracing Problem**. It has been shown that isolated concepts can be learnt using stateless spaced repetition algorithms (all of them based on Ebbinghaus' *forgetting curve*, such as *Half Life Regression*). When concepts are learnt in context, and there is reasonably continuous exposure to new vocabulary, it is likely that words are learnt faster. It is possible that stateless protocols can actually work when additional data is taken into account (i.e. multivariable HLR). But this would seem to contradict our current memory model (working memory -> short-term memory -> long-term memory). My hypothesis is that if someone had a lot of exposure to some term during, say, a month of his/her life and then no exposure for a year, s/he might doubt the meaning of that term after a year, but after minimal exposure will likely remember most/all of what s/he had learnt. A stateless model will probably not be able to predict this. Stateful models (*Deep Knowledge Tracing*) have been proposed to model the Forgetting Curve and hence predict the performance after a time-gap, albeit with limitations. @whatdoestimetell, for example, showed in the analysis of their learning app that DKT outperforms *Bayesian Knowledge Tracing* (a stateless model), but their usage was still limited to isolated concepts, and considered only short gaps (up to a week). So a research question is to try to apply DKT, HLR and possibly BKT to the Vocabulary Knowledge Tracing problem.
- **The Vocabulary Collaborative Filtering Problem**. Which words are intrinsically more difficult to learn than others? For example, longer words might be more difficult to learn. Some words might look like other words and lead to confusion. Some words might be the same as in another known language and hence be 'free'.

- **The Vocabulary Recommendation Problem** Use CF to make a prediction of which vocabulary the user already knows, even though s/he hasn't explicitly marked the words as seen. Use that to then suggest vocabulary to learn before reading a novel.
- **The User Clustering Problem**. Different people have different types of intelligence and different types of memory. Can we find some variables to cluster users into different groups?
- **The Book Recommendation Problem**. Given a set of books $\mathcal{B}$, and a desired set of words to master $\mathcal{W}$, can we provide a learning path to learning all of the words by reading some of the books? This is actually a very complex question because it is hard to know what sort of variables we should consider. A very vague desiderata would be as follows:
  - We would like to propose the best sequence of books to master a certain vocabulary?
  - What does it mean to master certain vocabulary? it means to take the user from a starting point and progressively increase his or her vocabulary. In other words, each books should be a bit harder than the previous book, but not too hard.

It looks like an NP-hard problem. If we had the assurance that all books in the set were useful to learning the language, then we would still need to produce a longest path

2020/05/10 More things to improve

1. Definition lookups suck right now. I should implement my own dictionary that parses the whole sentence and finds separable verbs and looks up the infinitive. Can use `dict.cc` or `wiktionary`.
2. A related problem: saving sucks because I should be saving the lemma and then words inside it, or separable verbs. Separable verbs should probably be saved as a list. Also it is not clear how this could be generalized to phrasal verbs in English. Or how to deal with different tokens which 'look' the same but are actually different: `como` and `como`. I need to think about this. What is a vocabulary item and how to save it.
3. A more efficient way to associate chunks with word families. When a new token is added, then find all chunks which match. Do it as well when a new text is added. I am not sure that NLP'ing a whole text is a good idea and remain in favour of NLP'ing sentences on demand.

Cool idea: revise sentences The machine could assemble a bunch of chunks and then rate how likely you are to look them up. Then it presents a list of chunks instead of a list of words

Cool idea: revise many words at a time Basically pagination for the table. Click on the words you don't know, then click a key or a button to continue.

The word family question Should I do NLP of everything? Count known word families instead of known words? This would seem like a very good idea.

Lomb: A Development Roadmap

1. Establishing the Domain

No pretty frontend, no security. Minimize the amount of moving parts and focus on modeling the Domain. Much easier to develop new features. Goal is to develop core features of the app before developing a more complex multi-user version. Core features:

- What are the fundamental things about this app? What does this app need to get right from the beginning?

- **Data tracking**. Which pieces of interaction need to get tracked exactly and where are they stored?
- In order to know what to track, we need to know what sort of **research questions** the data will allow us to answer.
- The **data analysis** part is worked out at a basic level (there is some sort of framework and workflow for working with the data). There are simple, anecdotical answers to the research questions based on a limited (single-user dataset).
- **Usability**. The future UI will need to get people to use the app in a certain way. But in which ways should the app be used? There has to be a basic answer to what use the app will have in users' lives before further development can continue.

As these functionalities develop, it should be clearer where the bounded contexts lie.

2. From a single-user app to a 'close friends' app.

- At this point, a **users backend service** will be developed to allow routes to be secured.
- A **web UI** will have to be developed.
- It seems likely that users will be able to upload and share their own texts on the platform, so the **library** will need to have to be adapted to secure texts (probably by generating some sort of secret token which is shared).
- More languages will need to be added, which means more specific language parsing logic, new NLP modules to be researched and used, etc.
- Other, unforeseeable work, as a consequence of new user stories and user feedback

3. The multi-user, 'get people to use it' phase.

- At this point the app should be reasonably mature and ready to receive more users. At this point I can start collecting data and write a paper and a thesis...

Incidental Learning

From this interesting article: https://linguapath.com/learn-language-through-reading/

Over time, as we encounter the same token over and over again, we add extra information to our "mental dictionary" until we construct a pretty good description of this word's meaning. It adds +1 to our internal vocabulary. This process is called incidental learning.

As Paul Nation, a leading vocabulary acquisition researcher from Victoria University of Wellington, explains, a learner should know 98% of vocabulary in a book in order to read it with ease and learn the remaining 2% incidentally.

-> So there are some metrics for book recommendations. If we restrict that we should only have 2% new word families then we can just use a modified shortest path algorithm or something like that. Probably something $O(N^2)$

What is really crucial here is frequency because repetition is essential for vocabulary acquisition. However, there is no set "optimal" number of repetitions necessary for learning a word. Recommendations vary from study to study, but Dr. Paul Nation assures that 12 repetitions would be a safe bet in most cases.

2010/05/11

It occurred to me today that what I am doing right now is developing the domain. That is, the *language* that will be used to refer to the problem.

Also, I will need to change the way that the data is represented, again. I should probably add a new review type for words clicked while scanning a word list. How to model this?

I should probably start splitting all of those js functions into separate files which are then loaded from `static`. I need to find out how Flask handles static files. I am starting to get a lot of repeated code in my js.

It also occurred to me that we should probably analyse and rank the word families. How many word families exist in the American Corpus? It would be useful to have this dataset.

More on the book recommendation problem. The average adult English-speaker knows about 42,000 lemmas, or somewhere between 10K to 20K word families. We can easily calculate the frequencies of those lemmas from corpora. Then, the lemmas contained in a text can be represented by bitsets, which could be further compressed into 4-word Bloom filters per 1K of lemmas. So we would have a 160*64 bits Bloom filter. These could be used to quickly filter books down. Even if if is of radical complexity, bitset operations are very fast and would be acceptable for anything in the order of magnitude of 100's of books. We would then just calculate the differences between the books and possibly present them in a table. After that... maybe some dynamic programming? The idea is to prunne the search space at each step with the Bloom filters.

Actually, from an applications perspective, it would be great if a user could start reading a book, and after a few pages of looking things up the app could make a suggestion on whether to continue reading or change books to something easier (or more difficult).

Populate review with examples So I don't have to do it by hand every time.

Domain classes Application core logic.

Add tracking to the examples in the review and frequency lists pages

Start saving word families, not words Chunks could have an additional field with a mapping from tokens to lemmas and tags.

Keep the id of the last read sentence

Once this stuff is done, and everything is working properly, rebuild the dataset from the logs!

Translate ePubs and save them with chapters to json

Recommending Books

Two ways to go about the problem:

1. A 'taste-first' approach. We fill in a sparse matrix based on users' scores. We also calculate the proportion of new tokens (for the user) in each book (this could be done efficiently somehow, e.g. by scoring books for difficulty first). The final score is a weighted sum of both values. This is just a 'next-book to read' algorithm.
2. A 'learn-first' approach. This is close to the learning itinerary problem. A user has read some books and would like to read some other book but finds it too difficult. The problem is to recommend an itinerary of books (a reading list), to allow his reading skills to gradually improve and finally be able to read his/her target book with relative ease.

This is basically a graph problem. When a book is added, we first calculate the semantic distance to all other books (e.g. using word embeddings, `word2vec` or something like that). We can represent all of these distances between books as a complete undirected graph. We make the distance function more sophisticated by weighing the users CF scores table. Now what we have is basically a transportation problem (go from a

starting node to an end node), in the Capacitated Vehicle family. We could apply a modified Clark Wright Savings algorithm or something like that.

Stable Dependencies Principle

The dependencies between packages should be in the direction of the stability of the packages. A package should only depend upon packages that are more stable than it is.

Word status should appear next to words as a coloured dot

Those bloody separable verbs

Endpoints to add and remove words The callback would look for an example and use that to produce the lemma. Then it would remove that token from that lemma. Likewise for adding words. The that are now in store could be coverted to lemmas like that.

Celery for long-running NLP taks E.g. when a book is added.

Should be able to import functions from notebooks instead of copy pasting

- Review words in sentences instead of in
- Underline target word in examples
- Scrape Amazon
- Scrape DW and collect in a book
- Collections of Short Texts - e.g The DW Archives
  - Each archive could be a book