Project Highlites: A simple yet useful open source solution for classification predictions, based solely on your data

Alain Descamps, Universiteit Antwerpen - Wander Departement Bibliotheek & Archief Dienst digitale oplossingen

Abstract

Subject classification of publications, projects and other entities is ubiquitous in CRIS systems. This project arose from practical considerations: how could we easily use AI to facilitate the cris administrator's work? This led to an application that can be deployed in various settings: given a brief textual description (e.g., a title and/or abstract of a publication), the application can generate one or more proposals for subject cataloguing. This can be implemented directly in various settings and for different types of classifications. Given sufficient training material, the application can be used with any classification system. We initially considered limited UDC coding for internal use, but also the retroactive assignment of so-called VODS codes, a scheme developed within the framework of the Flemish ECOOM project (ECOOM, n.d.; Vancauwenbergh & Poelmans, 2019).

The goal had to be realized with the resources we had available: our production server and the existing software used by our Wander software (https://www.wander.be). Under the hood, we found several open-source resources that we started working with. The first resource was our database engine, a no-SQL database with which we had over 25 years of experience and which had more than proven its worth: fast, scalable, and with flexible data storage — exactly what we needed.

The second tool was the programming language used to originally access the database: M, or Mumps, one of few programming languages supported by an ISO standard. Its purpose is again limited, namely controlling the database, but in that respect, it excels: all the requirements for a modern database, such as ACID atomicity, consistency, isolation, and durability, transaction processing, database recovery, mirroring, etc., are met. Moreover, it is a highly scalable product, functioning on multi-processor servers up to and including Raspberry Pis.

The third part involved expanding the M database to also be accessible from Lua: this MLua project, now known as YDBLua, was also sponsored by the University of Antwerp and made open-source (https://gitlab.com/YottaDB/Lang/YDBLua). This made it possible to develop precisely those things in Lua where Lua is stronger than M, such as processing files in XML format. The M database has since been integrated into other programming languages, including Python and Go (YottaDB, 2026).

The fourth part consisted of software pieces from our Wander repository, which had already proven their worth in our applications and each solved specific problems. For example, there is an algorithm that attempts to intelligently split a sentence into tokens. A nice twist: we also developed an algorithm that converts a sentence to lowercase, and does this for every known Unicode character, but also allows you to choose which Unicode version you want and compare the results.

The fifth part was a backoffice application, which allowed us to use certain test setups by our users and to collect initial experiences from them.

Developing the algorithm itself was quite an adventure: it had to be lightweight. For practical reasons, guessing a classification had to be done within a reasonable timeframe — say, a second — and it couldn't consume a lot of computer power, which were shared with all other applications. It quickly became clear that the current mainstream approach with neural networks required far more system resources.

Our algorithm became simple and modest: we assumed from the outset that sometimes *no* proposal would be calculated in any possible cases. The reason is that, in our opinion, a bad or pointless proposal is likely worse than no proposal.

When training the know-how database, the titles and abstracts are split into a series of tokens, and the resulting classification for each token is stored. *No* qualifiers are considered. The tokens are essentially the words from the titles and abstracts, split appropriately and converted to lowercase.

When guessing a classification for a given new title or abstract, the sentences are also split into tokens using the same process. These tokens are ranked according to the number of different classifications they are associated with. Words that lead to many classifications, such as 'international', 'the', etc., are less important; words that lead to few classifications - like 'sociodemographic' - are much more important. In the first round, the classifications to which the 'rarest' word leads are taken. In the second round, you take the intersection with the 'second rarest' word, and you continue this until you have no more classifications left: the classifications you had in the penultimate round become your answer. If this exceeds a certain number — for example, 10 — the algorithm does not generate any suggestions.

Interestingly, this algorithm is generic enough that it can even be used for other purposes than subject classification. A nice example is recognizing the language of a text. After all, one can also view a language as a kind of classification. All you have to do is fill the system with words of which you are 100% certain in what language they are written. We compared this approach with the algorithm we already had, which was based on 2- and 3-grams, and the results were generally even better.

The project is currently under development at https://gitlab.com/AlainDsc/highlites/. Is this project currently available as open source?

No, but we hope it will be by the euroCRIS Conference in Ponta Delgada.

Essentially, two things still need to be done.

The first one is to create a command-line interface that allows the necessary instructions to be executed via the prompt. These instructions consist of a 'training' component — to train the 'know-how' — and a 'guess' component, to calculate and propose a classification proposal. This appears in JSON format on standard output. The final product of this project is therefore not a ready-made application, but a piece of software that a developer can work with. Modules in Lua and maybe Python should be available.

The second part consists of converting the already programmed code into an open-source version. This process will definitely be completed by the euroCRIS Conference in Ponta Delgada.

References

ECOOM (n.d.). Research disciplines. https://www.ecoom.be/en/research/research-disciplines

Vancauwenbergh, S., and Poelmans, H. (2019). The creation of the Flemish research discipline list, an important step forward in harmonising research information. Procedia Computer Science, vol. 146, pp. 265–278.

YottaDB (2026). Multi-Language Programmer's Guide. https://docs.yottadb.com/MultiLangProgGuide/index.html