An approach to automated literature scanning

Bill Oxbury
v1.0 *March 2022*

BirdLife International maintains vulnerability assessments for about 11,000 bird species globally [2]. These assessments contribute to the IUCN red-list [1] and inform interventions to protect vulnerable-to-critically-endangered species.

These species assessments are, in turn, informed by numerous sources including journal publications in the biology and wider scientific literature. A natural challenge is to assist the process of harvesting information from the literature by means of web scraping and natural language processing (NLP). Indeed, NLP as an area of Artificial Intelligence (AI) has made rapid advances in recent years, and it seems ripe for the conservation community to take advantage of these capabilities.

This report describes an architectural design to approach this problem. The work is very much in progress, resulting in an α-level prototype after two months' work [3]. (This report should therefore be regarded as preliminary, to help inform the BirdLife team.)

The report will focus on the four main components: data acquisition and preparation, the core machine learning algorithm, the user interface and cloud deployment.

1. Data acquisition and preparation

This first part of the process builds and updates a database of URLs of relevant journal articles. Each record has fields:

- URL
- PDF link if available
- publication date
- title
- abstract
- domain
- file format (e.g. HTML, PDF, Microsoft Word etc)
- *plus some other fields which will be described below*.

The database is currently stored as a CSV file with around 3,000 rows, but in production would use a database application such as MySQL.

The domains that are scanned are currently five: www.nature.com, journals.plos.org, conbio.onlinelibrary.wiley.com, avianres.biomedcentral.com and www.ace-eco.org. These are all open-access journals and allow web-scraping.

This section (and this part of the process) focuses on which URLs (i.e. which journal articles) to acquire for the database; in the following section we will describe how to score articles for relevance to BirdLife analysts.

The end-to-end process for updating the database is implemented in Python, runs daily and consists of the following six stages.

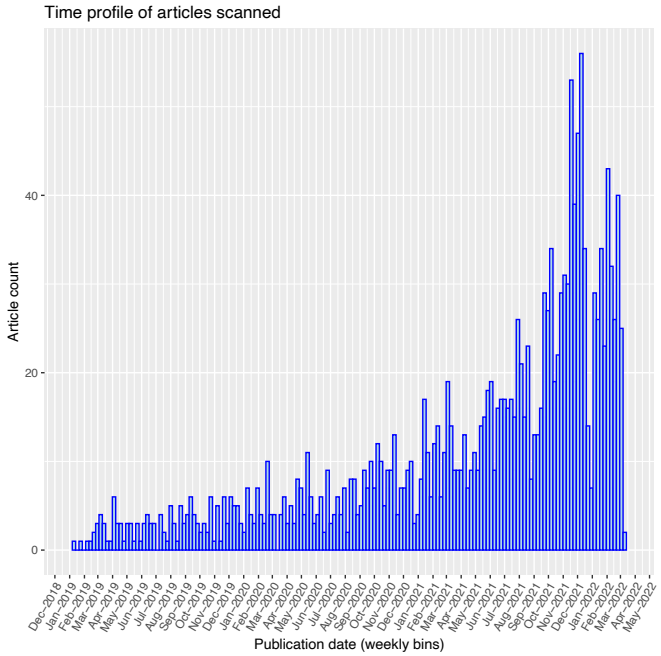
1.1 Google Custom Search API

The first step applies Google Custom Search [4] to a list of search terms. The search is confined to the specified domains (as above); the search terms are taken from a list of around 1,000 generic common names of bird species (from '*accentor*' to '*yuhina*'). The search is qualified as 'bird species' in order to disambiguate them (the list includes names such as '*bishop*' and '*oo*').

Each response is a JSON file containing metadata and URLs. The relevant parts of this are added to a master JSON file, recording the search date and search term used for each item.

1.2 Parsing the JSON

The JSON thus created contains much more information than is needed, and a script is run to extract fields such as URL, publication date and file format. An age threshold (currently 3 years) is applied to articles – the histogram below shows the profile of publication date of items in the database (as of 10-Mar-2022).



1.3 Extract text from HTML

Each URL for which file format is expected to be HTML is then followed to see if it's directly readable. Often it's not because the file format has been given incorrectly or the link is broken. In most cases for the domains of interest, title, abstract and PDF-link can be read directly from the HTML. These are often duplicated and at the end of the process the bad or repeated link records are removed.

1.4 Extract text from PDF

If text cannot be extracted directly from the HTML but a PDF-link is present – or if the URL already links to a PDF file, then that PDF is downloaded, read and then deleted. In 'reading' the PDF, we parse text blocks to attempt to find the title and abstract. The title is usually given either in the PDF metadata or in the Table of Contents. For the abstract, we look for the first block of text with multiple 'clean' sentences.

Often the PDF does not process correctly because it may be a scan of an archaic article, or a supplementary graphic only. These records are all removed at the end of the process.

1.5 Run text against a BirdLife model

At this point the database contains usable records containing URL, title, abstract etc. This stage then runs the title and abstract against a pre-computed model and assigns a score for relevance to BirdLife. The model will be described in the next section.

1.6 Detect species

The final preparation stage is to check for the presence of named bird species in each title and abstract. This uses the Python library *spaCy* [5] and a taxonomy constructed from a list of common names, Latin names, alternatives and synonyms provided by BirdLife.

Remarks:

(i) *Extending to more domains (journals)*: this is very easy at the custom search stage 1.1. Stage 1.3 then entails some bespoke work for each domain, since layouts and conventions vary between journals. However, this could be by-passed by focusing on the PDF, which is *not* treated differently between journals. This could offer a practical way to scale to large numbers of domains.

(ii) *Extending to more text within a document.* All of stages 1.3 to 1.6 can in principle be applied to text throughout the document, not just the title and abstract. This is desirable in future versions, but using the relevance score computed from title/abstract to direct one's search and reduce unproductive computation.

(iii) *Extending entity recognition beyond species.* It would be straightforward to extend process 1.6 to entity recognition beyond species, for example to location references, offering geo-views of the data.

2. Scoring for topic relevance

The output of the process described in the previous section is a database of scientific articles from trusted open-access journals focused on bird species. However, we still need to discriminate between titles such as '*Recurrent chromosome reshuffling and the evolution of neo-sex chromosomes in parrots*', which is unlikely to be of direct relevance to BirdLife, and titles such as '*Migration routes, population status and important sites used by the globally threatened Black-faced Spoonbill*' which are likely to be of interest.

And of course the title alone will not be as informative as the abstract (or ideally scanning the entire contents!). We combine the title and abstract for this purpose.

An unsatisfactory approach to assessing relevance would be to add additional qualifications to the original custom search, e.g. using query terms '*population status*', '*conservation*' etc. This would limit the pool that we have to search in, and would lose us articles that use vocabulary we might not have thought of.

A similar argument applies at the post-processing stage: if we seek to filter the database by using a list of search terms, we will be limited by our imaginations and won't know how many babies we're losing with the bath water.

Our approach, instead, is to learn (in the sense of machine learning) from data already generated by the BirdLife analysts themselves. The BirdLife DataZone [2] contains a rich supply of information, including text reports – see for example 'Text Overview' against any of the 544 species in the European Red List [6].

The data used to train our relevance algorithm consisted of the text overviews scraped from a sample of 4,633 species globally (not just European).

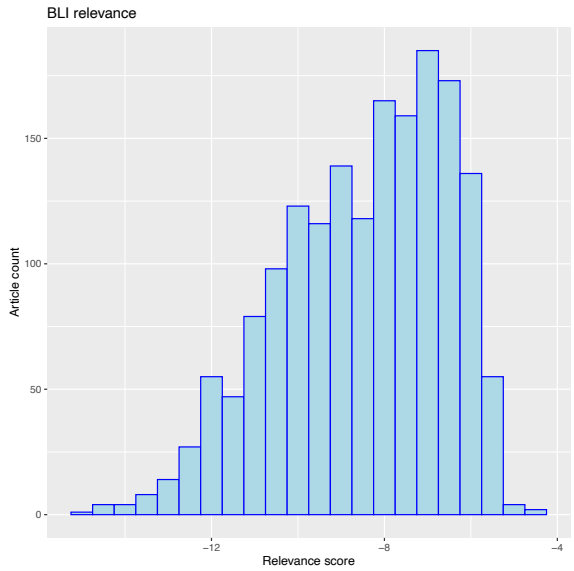
These text overviews were cleaned (headers and bibliographic references removed) and sentences de-duped (standard sentences often repeated across species). This resulted in a corpus of 37,788 distinct sentences – which collectively offer a proxy for 'relevance' to BirdLife.

Our approach to scoring titles/abstracts of journal articles is then to ask: *for each sentence in the title/abstract, what is the probability that this sentence could have been drawn from the BirdLife corpus?*

This probability, if it can be computed, is then a likelihood function for the sentence. Our score for the title/abstract is taken to be the average likelihood over all sentences. (We actually work with *log-likelihood* for the usual numerical reasons.)

To estimate the probability of being drawn from the BirdLife corpus we take a simple 'bag-of-words' approach. That is, we treat a sentence naively as a bag of words drawn independently from a probability distribution, and the sentence probability as the product (normalised for sentence length) of the word probabilities. The word probabilities are estimated simply by the frequency of BirdLife sentences in which the word occurs.

The histogram below shows the distribution of scores (average log-likelihoods) as of 10-Mar-2022. The skew to the right suggests that the initial custom search is doing a good job.



Remarks:

(i) Improving on bag-of-words. Clearly the naive bag-words approach could be significantly improved. Modern deep learning (neural network) methods are very good at modelling word order and semantic structure, and incorporating these methods for this problem is a clear next step.

(ii) How good is the score? Of course, use of more sophisticated methods should depend on (a) an assessment that current simpler methods are not 'good enough' (deep learning is computationally intensive and should be used only when it demonstrates a clear improvement in utility); and (b) objective methods for comparing performance of competing algorithms. For the latter, ideally one needs some benchmark analyst-assessed relevance rankings.

(iii) Extending the training signal. Training 'signal' for this problem encompasses both the text corpus that was scraped from the BirdLife DataZone (the 'training data' set) and information input directly by analysts (e.g. a benchmark ranking of articles). There's good scope for extending both sources of signal: by extending the training corpus to more species, by drawing on geolocation and other information in DataZone, by using journal articles that have been used and archived by BirdLife analysts. In terms of data input from analysts directly, we have an opportunity to use the User Interface (discussed in the next section) to harvest user feedback that can be used in an 'active

learning' loop. Once again, these extensions should depend on an objective assessment of existing performance.

3. User Interface (UI)

The output of the processing described in the previous sections is a database of journal articles, each with information allowing us to present title and abstract, retrieve and mine a PDF, and tagged by species present (in the title/abstract) and a BirdLife relevance score. As of 10-Mar-2022, this database contains 1,712 articles, with publication date at most 3 years old and 67% published in the past 12 months. (See the date profile in section 1.)

The LitScan application, currently in α version, allows users to search and browse this database. It is written in *R shiny* [7] and hosted in Google *Cloud Run* [8].

The app is self-explanatory, and the only remark to make here is how the relevance score is used. First, all results are returned in descending order of score, i.e. most relevant articles at the top of the page. Second, score is indicated visually with a red/amber/green traffic light: *green* indicates the upper quartile (top 25%), *red* the lower quartile (bottom 25%) and *amber* the middle 50% of the scored articles.

What is the purpose of the traffic light? A BirdLife user will know whether an article is relevant without needing to be told! But it serves two functions:

First, it provides a visual check on the performance of the scoring algorithm (which is being used to rank search returns, even without a traffic light). This may give confidence to the user, and is important to the developer, especially in early versions of the app.

Second, and perhaps more important, it provides a context for getting feedback from users: if a user disagrees with the traffic light, this can provide useful data for an active learning loop to improve the algorithm. (See remark (iii) in the previous section.) If the app is developed further, then a feedback button against the traffic light will be a natural addition to provide this data.

4. Cloud deployment

The daily workflow to maintain the LitScan app is currently

1. Run the processing stages 1.1 to 1.6 (about 30 minutes runtime)
2. Build and upload to Google Container Registry a Docker image of the R shiny app, including the updated database
3. Deploy the Docker image to Cloud Run.

To-do:

- Migrate the database to a MySQL instance hosted in Google Cloud
- Deploy processing 1.1 to 1.6 as a scheduled job in Cloud Run, updating the MySQL database.

When that is done, the process is self-sustaining and the Docker image only needs rebuilding when functional changes are made.

References

[1] IUCN Red List of Threatened Species https://www.iucnredlist.org/
[2] BirdLife Data Zone http://datazone.birdlife.org/home
[3] BirdLife **LitScan** α version https://bs-bli-5-domain-rgkbx46boq-nw.a.run.app/
[4] Google **Custom Search** API https://developers.google.com/custom-search/v1/overview
[5] **spaCy** Natural Language Processing in Python https://spacy.io/api
[6] European Red List http://datazone.birdlife.org/info/euroredlist2021
[7] **Shiny** from RStudio https://shiny.rstudio.com/
[8] Google **Cloud Run** https://cloud.google.com/run