# Lodestar: Ontology-Based Annotation of Textbooks

Kyle Robbertze
RBBKYL001@myuct.ac.za
University of Cape Town
Cape Town, South Africa

## ABSTRACT

Textbooks are starting to use new techniques that digital book technology can offer. Currently available smart textbook platforms are limited to a single textbook and ontology pair and do not yet generalise to the use of any two related textbook and ontology.

This paper presents Lodestar — a smart digital textbook platform for ontology-backed textbooks. Lodestar provides a web-based textbook viewer with ontology-annotated term definitions and related concept listings. These definitions and related concepts are generated from the provided ontology. Lodestar is suitable for testing within organisations and, with some improvements, deployment on production systems.

## CCS CONCEPTS

• **Information systems** → **Enterprise applications**; *Digital libraries and archives.*

## KEYWORDS

smart textbooks, ontologies, knowledge representation, textbook use, digital textbooks

## 1 INTRODUCTION

Textbooks are used to convey knowledge. With the progression of the digital age, textbooks have frequently become digital. However, these digital textbooks often take the form of a scanned or typed copy of the physical book and thus, do not take advantage of new techniques that digital books can bring. These techniques include text annotation, context-based navigation and automatic question and answer generation [2]. The inclusion of new digital techniques in digital textbooks has been shown to improve students' performance in homework tasks by approximately 10% [2]. "Inquire Biology" is the only example of such a knowledge-driven digital textbook that currently exists. However, it is proprietary and is very specific to a single edition of a single textbook.

Thus, there is a lack of a generalised smart textbook platform that can be used to interact with a wide variety of textbooks. A generalised platform could help bring the academic improvements observed by Chaudhri et al. (2013) using "Inquire Biology" to other disciplines and textbooks without needing to re-engineer the combination of textbook and ontology for each textbook. This software development project aims to develop such a generalised smart textbook platform. Lodestar is a web-based platform for reading and managing knowledge-driven textbooks.

The knowledge base used in Lodestar takes the form of an ontology, which is a formal representation of concepts and their relationships [10]. In the context of smart textbooks, the ontology is a graph with nodes representing concepts and edges representing the relationships between concepts. A concept can also have properties, which are attributes specific to that concept. For example, a concept can have a definition property that contains the definition of the concept in relation to the ontology domain.

Lodestar uses an ontology to provide text annotation of textbooks. A text annotation is additional information about a word or phrase in the text that is displayed on-screen to supplement the text itself. In the case of Lodestar, these text annotations take the form of pop-up definitions of words and phrases with a list of related concepts. It achieves a precision of 89% and a recall of 70% when matching terms in the Data Mining for the Masses textbook [23] to items in the Data Mining Optimized (DMOP) Ontology [12]. This is an improvement on the current state of the art. Recall is the ratio of correct positive predictions to total positive elements, while precision is the ratio of correct positive predictions to total predictions made by the system.

Lodestar performs text annotation by matching words and phrases in a textbook to element names in the ontology. The matching process progresses through various known generalized matching patterns. Initially single words are matched (with the assumption that the element name is singular). Next phrases are matched. If, after these two matchers, there are still no matches, words in the text are converted to singular form and the single word matching occurs again against these singular words. Finally, synonyms for the element name are generated using WordNet [20], a lexical database for English, and these are matched against the text using the single word matchers.

This paper is laid out as follows: Section 2 investigates the existing literature on the subject of smart digital textbooks, Section 3 describes the software design of Lodestar and the development process that was followed, while Section 4 describes how that design was accomplished in the final software project. Section 5 describes how Lodestar compares to other annotation systems and Section 6 pulls conclusions about Lodestar from these results and contemplates additional work that could further improve Lodestar.

## 2 BACKGROUND

Textbooks have a long history of use by students and educators alike to cover material and plan lessons [9, 21]. They are considered an essential learning tool, no matter the manner in which it is used within the classroom [27]. They have also been shown to contribute positively to students' academic performance in developing countries. This is due to the fact that a textbook can provide a more comprehensive coverage of the content than a teacher may time to go through in the classroom [18].

In recent years, universities are increasingly using digital textbooks to offer cost-effective, efficient and accessible resources to students [28]. Studies have found that South African students read digital textbooks at the same speed as they read hard-copies, while maintaining the same comprehension levels as when they were

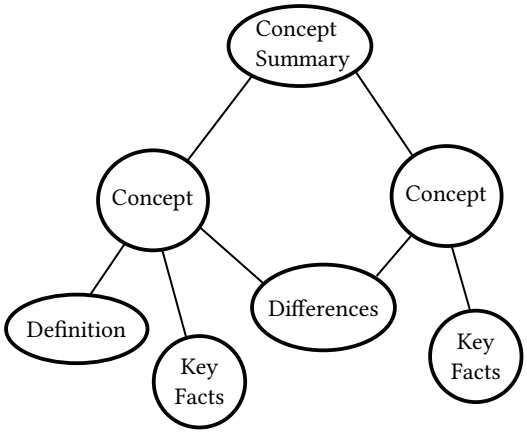


**Figure 1: A graphical outline of what a differential concept summary between two concepts would look like.**

using hard-copies [29]. However, Sackstein et al. (2015), hypothesized that reading speed could be influenced by students' prior exposure to the medium that they were reading on [29]. Thus, students' reading speed could solely be because they are familiar with the platform, and the study results could be hiding potential issues, such as students presenting lower comprehension levels and slower reading speeds that students unfamiliar with digital textbooks could encounter while using them.

It has been noted that digital textbooks are easier to update and distribute via the internet [3, 6]. Frequent updates are required for subjects such as taxation tables and case-law where new standards are often being set. This easy updating of digital textbooks also reduces the number of different editions of textbooks students need to purchase, minimising the cost incurred while studying [8]. A web-based platform is able to provide updates quickly to all its users as all textbooks and associated data are located in one location controlled by the maintainer of the platform, thus minimising the number of editions that students are required to use.

## 2.1 Digital Smart Textbooks

A digital smart textbook makes use of adaptive e-learning techniques to enhance a textbook and to encourage students to read the textbook actively. Active reading is the practice of summarizing, questioning, clarifying and predicting when learning a concept. Palinscar and Brown (1984) found that active learning, which active reading is a part, significantly improved students' comprehension of the content being studied. Students were also able to generalize these gains to class tests, thus improving their academic results [24]. Further, Chaudhri, et al. (2013) found that including adaptive e-learning into a digital textbook, "Inquire Biology", improved students' performance in homework tasks by approximately 10% [2]. Adaptive e-learning techniques include concept summaries, navigation history, asking and answering students' questions and in-line definitions of concepts [2].

Concept summaries are generated summaries of a single concept in the textbook. It includes the definition of the concept, key facts, and relationships with other concepts. Each linked concept then has its own concept summary. Concept summaries are independent of chapters and pull knowledge from different chapters into a single place, thus helping students to read actively. Active reading occurs through enabling students to clarify and summarize the content around a concept. These concept summaries are important as they clearly summarize the content without requiring the student to wade through irrelevant information as they would if they were searching through other resources such as Wikipedia [2]. Figure 1 shows a concept summary displaying a concept with the differences with a linked concept highlighted.

A potential limitation of a generated concept summary is that students lose any advantages gained by writing the summary themselves, as summarizing a piece of text is the first step of active reading [24]. While a generated concept summary does not prevent students from creating a summary of the content themselves, it may make students less likely to do so.

In-line definitions of concepts reduce students' cognitive load when working through the textbook by providing easy access to basic terminology [2]. Students experience high cognitive load when learning new concepts [32]. Reducing this load improves concept acquisition. This is a form of active reading and thus has been shown to improve academic results [24]. "Inquire Biology" provides these in-line definitions by underlining the term and providing a pop-up dialogue containing the definition when the student mouses over the term. This pop-up is useful as it allows students to skip over the definition if they already know it, while giving them the opportunity to test their knowledge of the definitions if they so choose [2]. Thus, further benefiting the learning process.

## 2.2 Text Annotation

Ontology-based text annotation is the process of mapping strings in text to ontology elements based on the provided subject domain ontology. This is done through detecting metadata and structuring the data for improved processing later [13]. Annotation is important, as this is how the textbook and ontology are combined to be presented to the user in a smart digital textbook platform: the ontology provides a semantic interpretation of the textbook [31].

The main link between an ontology and its text is a terminology, which maps concepts in the ontology to terms in the text. However, usually there is no simple one-to-one mapping between terms and concepts. According to Spasic et al. (2005), the probability of two experts to refer to the same concept with the same term is less than 20%. Another potential ambiguity is when the same term refers to multiple concepts depending on the context in which it is used [31].

There are several text annotation tools available, including Ontea [13], Textpresso [22] and GoPubMed [7]. These all focus on using ontologies to enhance written text and search results. While Ontea is generic to any piece of text and ontology, Textpresso and GoPubMed both focus specifically on the biomedical industry.

These tools are not suitable for use in Lodestar for several reasons. One reason is that Ontea is domain specific in the standard regular expressions provided [13]. Another reason is that Textpresso and GoPubMed are both highly specific to the biomedical field and would require an unjustifiable amount of effort to generalize

Ontea uses regular expressions to match text to concepts in the linked ontology [13]. Laclavik, et al. (2009) achieved an average

recall of 79% and an average precision of 53% using Ontea, but required the user to manually create the regular expressions to be used. Lodestar performed better, with a recall of 70% and a precision of 89%.

Ontea's performance was measured on text using an ontology specifically tailored to it. This would be expected to have a better precision and recall that using a separately developed ontology with the same text and annotation tool as the terms in the text and ontology are the same. Thus, improving the chance of matching terms correctly.

## 3 DESIGN

We aim to provide a digital smart textbook platform that enables educators and students to interact with textbooks in a context-aware, smart way. Lodestar allows administrators to upload textbooks and ontologies so that users can access the smart textbooks once they are processed. These smart textbooks include in-line definitions of concepts and related concept listing based on the ontology provided.

After meeting with the University of Cape Town's (UCT) Centre for Innovation in Learning and Teaching (CILT), we developed the following list of core requirements:

- Lodestar should annotate uploaded source files with a precision and recall which is, at minimum, equivalent to existing annotation systems such as Ontea [13],
- Administrators should be able to upload textbooks and ontologies,
- Users should be able to view textbooks after they are uploaded,
- Administrators should be able to added and remove users.

A list of desired additional features was also discussed, while these were not implemented, they form a reference for what is still needed before Lodestar can be used properly in a teaching environment. The list included:

- Integration with existing Learning Management Systems,
- User friendly interface following standard human-computer interaction principles,
- Track student usage to see which sections are used more frequently and students' progress through the textbook,
- Allow the user to highlight text and make comments in the margin

### 3.1 System Architecture

We selected a web-based system, as this enables administrators to update and maintain the textbooks provided easily. It further allows users to access the textbooks from various locations on a variety of devices without needing to develop specific applications for every anticipated platform. This reduces complexity and aids future maintainability. It also enables easy, cost-effective updating of textbooks by the administrators.

The platform is broken up into 4 parts: source parser, textbook annotation, textbook rendering and administration. This pipeline design ensures modularity and extendibility. Modularity allows the software to be altered with ease, as there are well-defined boundaries between parts and internal changes to one will not affect another. With this modularity, extending functionality is made possible by adding additional parts to perform the required duties. This extensibility helps iterative development and clean testing of changes. Figure 2 shows this pipeline process graphically.

The source parser module handles the upload of source textbooks and ontologies by the administrators. This is where supported formats are defined and metadata about the files are gathered. As administrators will be uploading content which is stored and processed on the server, there are security implications that need to be considered. Some of these are the responsibility of the server administrator, however some are considered within the software architecture chosen for Lodestar. Data security and server maintenance is the responsibility of the system administrator, while Lodestar must ensure that uploaded data do not present additional security risks to the system.

Uploaded textbooks are expected to be uploaded in one of the supported formats. If the file is not one of these, the upload fails. This approach reduces attack surface that uploading files provides. In order to reduce it further, files are converted to plain Hypertext Markup Language (HTML) files prior to further processing. This ensures that there is no malicious code that can be executed on the server. Using HTML as the intermediary file format ensures that the file is suitable for rendering on-screen even if no annotations are found.

Textbook annotation is the second step in the pipeline. This is where text annotations are added, which is done to reduce the processing power required to handle each client request that is sent to the renderer. This method increases the size of each textbook on disk, but the trade-off is that the time required to process requests is much smaller. The space-time trade-off is justified by the ratio of expected client requests to source file uploads. Each user will generate a request for every textbook viewed. Thus, requests are handled frequently, while the textbook should only be uploaded irregularly. Thus, processing the textbook each time a user wishes to view it would increase request handling time, making the site slow and unresponsive. In Lodestar, the textbook does not change on each request, rather on each upload of a new source file. This makes processing and storing the changes more efficient.

Once the textbook is processed, it can be displayed to the users in the rendering module. This module is responsible for the look and feel of the website as experienced by end users. The standard user view is a list of textbooks that they can access. Administrators have an additional link on the home screen that allows them to access the administration module.

The administration module is responsible for allowing administrations to maintain and update textbook and ontology details. This is also the entry point for adding new source textbooks and ontologies. Furthermore, the administration module allows administrators to manage users and user permissions on the website.

In every module, there is a clear distinction between the model, view and controller logic. In the Model/View/Controller (MVC) design pattern, the view displays information to the user, while the model is the internal representation of the data. The controller processes user input and interaction between the view and the model [17]. This separation makes maintaining multiple interfaces or changing the interface easy, without needing to change the internal representation of data. In iterative development this property

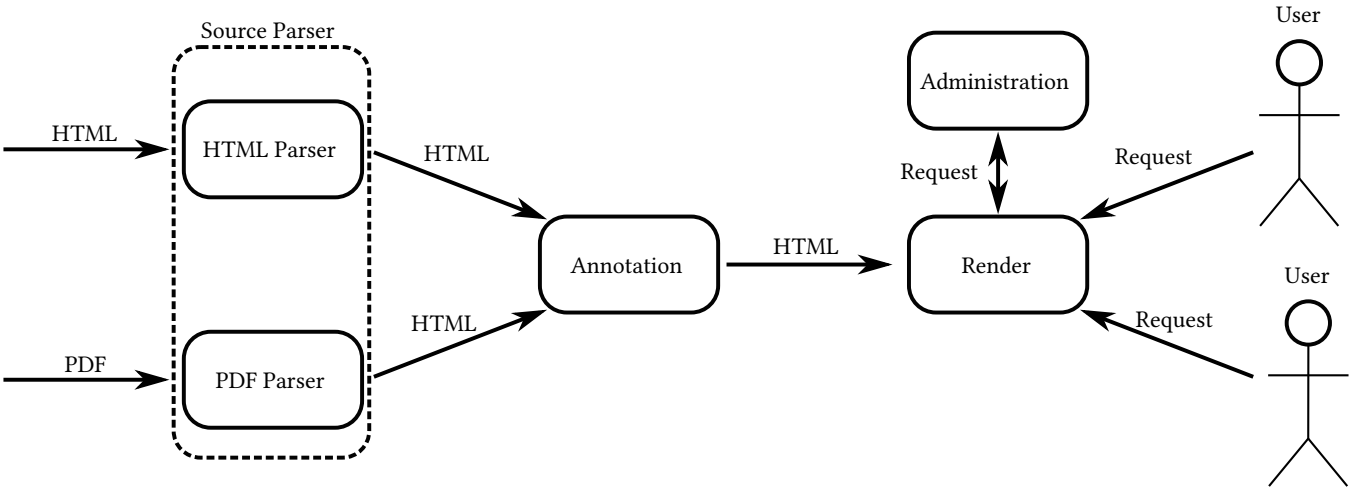


**Figure 2: Lodestar textbook pipeline**

is very useful, as it decouples the model, view and controller. This allows iterations to modify any one of the three easily without needing to change the others significantly.

### 3.2 Software Development

We chose to use an iterative development process. This process initially implements a skeletal sub-project. Additional features and enhancements are then added in future iterations [1]. Iterative development was chosen as it allows for flexibility in the implementation of features and does not need the team which is required in agile-based software development processes. The flexibility in implementation stems from the iterative nature of the process. After each implementation phase, the current implementation is analysed and this dictates on what the next implementation phase will focus [1].

The development cycle also followed Test Driven Development (TDD). TDD requires writing automated tests before developing the features that they test. Thus, initially, the tests are written and fail. The feature is then implemented and the tests run again. If the feature is implemented correctly, the tests should then pass [11]. TDD provides rapid, reproducible feedback on the state of the project. It also ensures that the test suite covers the entire project at a low level, thus ensuring external consistency when changes are made. However, as it tests features at a unit (method or class) level, it does not test for integration issues when multiple units interact in any complex way. These situations also need to be tested in an automated way to reduce the chance of regressions appearing.

Lodestar is licensed under the GNU General Public Licence (GPL) version 3 with the clause allowing redistribution under a later version of the licence at the distributor's discretion. This is an open source licence that allows users to modify and redistribute the modifications, provided that they release their modifications under compatible licensing terms [25]. Open source licensing such as the GPL ensures that, should the original authors stop developing the platform, interested stakeholders can continue on without needing to start from scratch. Users who require specific features for their environment can also freely make modifications. The source code is available at https://gitlab.com/paddatrapper/lodestar.

As the software needs to store user data, it must comply with the South African Protection of Personal Information (POPI) Act. This outlines that only data that are relevant be collected and that the user must consent to this data being collected before it can be done. Furthermore, data must only be used for the purposes that it was originally intended for and there must be reasonable safeguards to ensure the data's safety [5]. In Lodestar, this is done by never storing sensitive information, such as passwords, in plain-text. Instead, they are cryptographically encrypted before storing. The permissions structure also ensures that only registered and authorized users can access personal data. The system administrator can also ensure compliance by applying encryption between users and the server running Lodestar, and implementing security measures on the server to prevent unauthorized access.

## 4 IMPLEMENTATION

We chose to implement Lodestar in Python using the Django web framework[1]. This decision was made as Python is cross-platform and can be run on Windows, Mac and Linux [30]. This allows Lodestar to be developed and deployed on any platform, which ensures flexibility. Django also provides all the logic required to translate between web requests, responses and Python without us needing to re-implement it. TDD is supported through a structured testing framework implemented inside Django that allows tests to execute functions and construct classes in the same way as the Django engine does when serving web requests. It further allows testing of requests and responses at the web level. Thus, it makes performing integration testing of multiple components simpler. This improves efficiency and makes writing tests less of a burden.

Lodestar consists of four modules: the source parser, textbook annotation, rendering and administration. The source parser, textbook annotation and rendering module form a pipeline, with the

[1] https://www.djangoproject.com/

administration module providing user management, authentication and a permissions structure.

Progression through the processing pipeline is done using signals. When a source file is uploaded by an administrator and accepted by the server, it is saved into the database. This emits a signal to the source upload stage of the pipeline. Once this is complete, it emits a signal to schedule the textbook processing to occur in the background. The textbook processing takes place in the background so that the user does not have to wait for what can be a lengthy process, depending on the size of the ontology and length of the textbook. Their upload is accepted immediately, and scheduled for processing. The user can then continue using Lodestar or close their browser and the processing will still occur. The textbook will only be available once the processing is complete.

## 4.1 Source Upload

The source upload process needs to be tailored specifically for each supported file format. This is due to the fact that the different formats need to be processed into a common format that can then be annotated and rendered to users. Efficient processing of uploads was achieved through a hash table mapping file type to transformation function. A hash table provides easy extensibility by adding additional entries for new file formats as they are implemented. In order to improve maintainability, all source format parsers inherit from a single base class that is responsible for registering file types in the hash table and provides common code to reduce duplication.

Lodestar currently only supports two file types: Portable Document Format (PDF) and HTML. This limitation is due to a lack of feasibility with the time available for developing the initial version, and more formats can be added in the future. The PDF parsing function converts the source file into an HTML file and performs a variety of regular expression string replacements on it to remove illegal characters and ensure that it is formatted correctly for rendering. Conversion from PDF to HTML is performed by the pdftotext application from the Poppler[2] project. The HTML parsing function assumes that the file is valid HTML and ensures that it is copied into the correct directory with the correct name.

Once the source is uploaded and converted into a format that the processing pipeline can manipulate, it schedules the textbook for processing in the background.

Ontologies are uploaded as OWL/XML or RDF/XML files. When uploading, the administrator can add details about the ontology, including a list of the properties to use as definitions. A concept has a definition that can be added to an annotation if the concept has a value for at least one of these properties. The list of definition properties is a comma separated list. The last property in the list with a value found for each object in the ontology is used as the concept's definition. Once an ontology is uploaded, textbooks can be uploaded linking to it. This achieves our requirement of allowing administrators to upload textbooks and ontologies.

## 4.2 Textbook Annotation

Once the textbook annotation stage is started, it is responsible for annotating the HTML file so that the renderer can display in-line definitions and term relations. It processes the HTML line-by-line, finding words or phrases in the text that matches element names from the ontology and adds the required details as an HTML element to the text. The annotated line is then written to an HTML file that is later displayed to the user. This is explained in Algorithm 1. The algorithm uses owlready [14] to interact with the ontology.

**Algorithm 1** Annotation of a textbook

```
Require: ontology, sourceFile, htmlFile
1:  g ← loadOntology()
2:  predicates ← ontology.predicates
3:  definitionMap ← getDefinitions(g, predicates)
4:  while sourceFile has next line do
5:      line ← sourceFile.readLine()
6:      for all subject, definition in definitionMap do
7:          match, matchTerm ← getMatch(subject, line)
8:          if match then
9:              line ← line.replace(matchTerm, match)
10:         end if
11:     end for
12:     htmlFile.write(line)
13: end while
```

Algorithm 1 requires the file path to the source HTML file (*sourceFile*). This is the HTML file generated by the source upload module from the PDF or HTML file that the administrator uploaded. *htmlFile* is the file path that the annotated text will be written out to. getDefintions on line 3 creates a map of elements in the ontology to definitions. This is a hash map of element names to definitions, stored in *definitionMap*. If the concept does not have a definition, it is added, but the definition is blank. A concept has a definition if it has a definition predicate or property. These are the list of properties given by the administrator when they uploaded the ontology. The line test function then uses this filtered map of classes to definitions to detect terms.

getMatch on line 7 (Algorithm 1) calls the function described in Algorithm 2, which matches terms in the ontology to words in a line using a series of matching functions. These match terms, then single words and finally try variations on words such as plurals and synonyms using WordNet [20], an online lexical database. The plural and synonym matchers are described in Algorithms 3 and 4 respectively. This design allows the easy addition of new matchers by adding them to the list of matchers. getMatch (Algorithm 2) returns two data: the regular expression generated to match the term in the line and the HTML element to replace it with. This HTML element includes sections of the definition and related concepts so that they can be displayed when the textbook is rendered and is generated by getItem on line 4 (Algorithm 2). The regular expression is used on line 9 (Algorithm 1) to replace the term with the new HTML element. This regular expression is generated to ensure that the term is matched in the correct location in the line. For example if a previous term contains the current term in the definition, the regular expression must not match the term in the definition, rather the one in the original text.

In Algorithm 3, WordNet is used to split the line up into words (line 1) and then to generate the singular versions of each word

[2] https://poppler.freedesktop.org/

**Algorithm 2** Matching a term in a line of text

**Require:** *matchers*, *term*, *line*, *predicates*
1: **for all** *matcher* in *matchers* **do**
2: *stop*, *regex*, *match* ← *matcher*(*term*, *line*)
3: **if** "stop" in *result* **then**
4: *item* ← formatItem(*match*, *predicates*)
5: **return** *regex*, *item*
6: **end if**
7: **end for**
8: **return** None

(line 2). These singular versions are then matched against the term, ignoring letter case.

**Algorithm 3** Matching a plural of a term

**Require:** *term*, *line*
1: *lineList* ← *wordnet*.tokenize(*line*)
2: *lineListSingle* ← *wordnet*.getSingleEach(*lineList*)
3: **for** $i = 0$ **to** length(*lineListSingle*) **do**
4: **if** *lineListSingle*[$i$] = *term* **then** {Case insensitive match}
5: **return** "stop",
"\b" + *lineListSingle* + "\b(?![\s\w]*)",
*lineList*[$i$]
6: **end if**
7: **end for**
8: **return** None

WordNet is used again in Algorithm 4 to retrieve a list of synonyms for the term. A limitation of this approach is that WordNet is unable to determine which part of speech the term is, often resulting in many synonyms with the incorrect part of speech being returned and subsequently matched. WordNet is also limited in its breadth around scientific subject, thus often resulting in generic or incorrect (in context) synonyms being returned [19].

**Algorithm 4** Matching a synonym of a term

**Require:** *term*, *line*
1: *synonyms* ← *wordnet*.synonyms(*term*)
2: **for all** *synonym* in *synonyms* **do**
3: **if** *synonym* = *term* **then** {Case insensitive match}
4: **return** "stop",
"\b" + *term* + "\b(?![\s\w]*)", *synonym*
5: **end if**
6: **end for**
7: **return** None

As an example of the matching process, the following HTML paragraph and an ontology with the elements given in Table 1 can be combined by running Algorithm 1. The concepts in Table 1 come from the Data Mining Optimization (DMOP) ontology [12].

```
<p>
 A data set is required to perform analysis of data in order
  to make predictions
</p>
```

This produces the following output (additional formatting added for readability):

```
<p>
 A <span class="popup" onclick="showPopUp('definition-0')">
    <span id="class-0">data set</span>
    <span class="popup-text" id="definition-0">
      DataSet: in data mining, the term data set is
      defined as a set of examples or instances
      represented according to a common schema.
    </span>
    <span class="popup-relations" id="relations-0">
      <span class="popup-relation" id="relation-0">
        <span class="relation-name">type</span>
        <span class="relation-obj">Data</span>
      </span>
    </span>
  </span> is required to perform analysis of
  <span class="popup" onclick="showPopUp('definition-1')">
    <span id="class-1">data</span>
    <span class="popup-text" id="definition-1">
      Data is the generic term that englobes
      different levels of granularity: data can be a
      whole dataset (one main table and possibly
      other tables), or only a table, or only a
      feature (column of a table), or only an
      instance (row of a table), or even a single
      feature-value pair.
    </span>
    <span class="popup-relations" id="relations-1">
      <span class="popup-relation" id="relation-0">
        <span class="relation-name">type</span>
        <span class="relation-obj">
          non-physical-endurant
        </span>
      </span>
    </span>
  </span> in order to make predictions
</p>
```

## 4.3 Textbook Rendering

The processing pipeline produces an annotated HTML file for each textbook uploaded. The rendering module embeds this into the website to ensure a consistent look and feel. Consistency is used to ensure that the user feels comfortable while using the website. The user is presented with a list of textbooks and when one is selected, the renderer initially displays just the textbook's content on the screen. Terms are underlined and the user can click on them to access the definitions and related concepts. This information is presented in a side panel that pops up next to the text on the same page, which is consistent with other websites, such as GitLab and Facebook, where extra information is presented to the side of the main content. It also ensures that users do not have to switch between screens to view definitions and the textbook, making the website more user-friendly. By displaying textbooks to the user, we ensure that our requirement for users being able to view textbooks

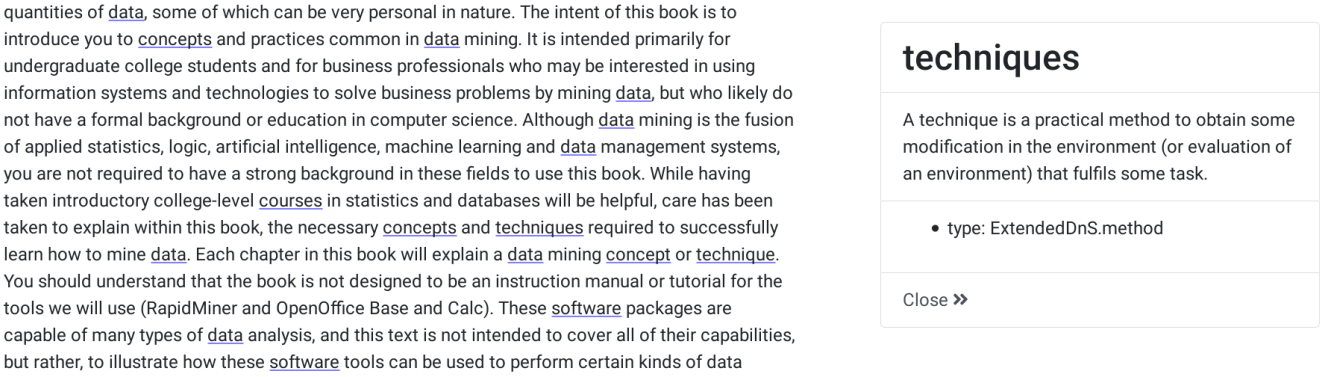


**Figure 3: The user interface of Lodestar showing the underlined terms and the information pop-up for techniques, which is shown when the term is clicked on.**

| Concept | Definition | Related Elements |
|---|---|---|
| DataSet | DataSet: in data mining, the term data set is defined as a set of examples or instances represented according to a common schema. | type: Data |
| Data | Data is the generic term that englobes different levels of granularity: data can be a whole dataset (one main table and possibly other tables), or only a table, or only a feature (column of a table), or only an instance (row of a table), or even a single feature-value pair. | type: non-physical-endurant |

**Table 1: A list of example concepts in an ontology. The related elements are elements in the ontology, and are not necessarily listed here.**

is met. Figure 3 is what the user sees when viewing a textbook on Lodestar.

All styling and manipulation of the web-page is done using standard web technologies, namely CSS, JavaScript, jQuery[3] and HTML [16]. This ensures that the renderer can be maintained easily in the future as these technologies already drive the majority of websites [16].

The administration section of the rendering module allows administrators to add, delete and modify textbooks and ontologies. Unnecessary re-processing of textbooks is prevented by only executing the processing pipeline when the textbook source file changes. This is done by storing the MD5 message digest hash and comparing this to the MD5 hash of new file uploads. The MD5 hash is a 128 bit hash of a file that is typically unique [26]. While it is not cryptographically secure [34], it is sufficient to minimise unnecessary re-processing of textbook source files. MD5 is not cryptographically secure because two files can occasionally produce the same MD5 hash even if their contents are different. In Lodestar, this will only cause the updated textbook not to process the new source, but this can be started manually by an administrator from the detail view of a textbook. The advantage of MD5 is that it is fast and not computationally taxing to compute [26]. Reducing the number of times a textbook is processed reduces the load on the server running Lodestar and thus, improves the user experience.

The administration section further enables the administrator to add and remove users and manage their respective permissions. This ensures that our user administration requirement is met.

[3] https://jquery.com/

## 5 EVALUATION

### 5.1 Materials and Methods

We evaluated Lodestar's annotation by comparing standard $F_1$, precision and recall measures [4, 33] between four variations of Lodestar's matching algorithm. The four variations were:

Base case — matching words and phrases,
Plurals — the base case and plurals retrieved from WordNet,
Synonyms — the base case and synonyms retrieved from WordNet, and
All — all three matchers three combined.

Precision is a measure of how many annotated elements are relevant

$$p = \frac{|\text{relevant elements} \cap \text{total matched}|}{|\text{total annotated}|}$$

Recall is a measure of how many relevant elements are annotated

$$r = \frac{|\text{relevant elements} \cap \text{total matched}|}{|\text{relevant elements}|}$$

$F_1$ is the harmonic mean of the precision and recall, giving a measure of the annotation's overall accuracy.

$$F_1 = \frac{2}{p^{-1} + r^{-1}}$$

Lodestar's annotation system was tested with 7 pages selected at random from the content sections of the "Data Mining for the Masses" textbook [23]. The content sections are the chapters in

| Base | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Page | Total Relevant | True Positive | False Positive | Total Matched | False Negative | Precision % | Recall % | $f_1$% |
| 15 | 31 | 16 | 5 | 21 | 15 | 76 | 52 | 62 |
| 65 | 25 | 9 | 4 | 13 | 16 | 69 | 36 | 47 |
| 66 | 24 | 10 | 2 | 12 | 14 | 83 | 42 | 56 |
| 81 | 30 | 19 | 0 | 19 | 11 | 100 | 63 | 78 |
| 107 | 5 | 0 | 3 | 3 | 5 | 0 | 0 | 0 |
| 227 | 28 | 23 | 0 | 23 | 5 | 100 | 82 | 90 |
| 239 | 36 | 22 | 1 | 23 | 14 | 96 | 61 | 75 |
| Plural | | | | | | | | |
| Page | Total Relevant | True Positive | False Positive | Total Matched | False Negative | Precision % | Recall % | $f_1$% |
| 15 | 31 | 27 | 5 | 32 | 4 | 84 | 87 | 86 |
| 65 | 25 | 9 | 4 | 13 | 16 | 69 | 36 | 47 |
| 66 | 24 | 11 | 2 | 13 | 13 | 85 | 46 | 59 |
| 81 | 30 | 28 | 1 | 29 | 2 | 97 | 93 | 95 |
| 107 | 5 | 2 | 3 | 5 | 3 | 40 | 40 | 40 |
| 227 | 28 | 24 | 0 | 24 | 4 | 100 | 86 | 92 |
| 239 | 36 | 25 | 1 | 26 | 11 | 96 | 69 | 81 |
| Synonym | | | | | | | | |
| Page | Total Relevant | True Positive | False Positive | Total Matched | False Negative | Precision % | Recall % | $f_1$% |
| 15 | 31 | 27 | 26 | 53 | 4 | 51 | 87 | 64 |
| 65 | 25 | 7 | 26 | 33 | 18 | 21 | 28 | 24 |
| 66 | 24 | 7 | 36 | 43 | 17 | 16 | 29 | 21 |
| 81 | 30 | 27 | 33 | 60 | 3 | 45 | 90 | 60 |
| 107 | 5 | 1 | 21 | 22 | 4 | 5 | 20 | 7 |
| 227 | 28 | 22 | 43 | 65 | 6 | 34 | 79 | 47 |
| 239 | 36 | 23 | 16 | 39 | 13 | 59 | 64 | 61 |
| All | | | | | | | | |
| Page | Total Relevant | True Positive | False Positive | Total Matched | False Negative | Precision % | Recall % | $f_1$% |
| 15 | 31 | 28 | 24 | 52 | 3 | 54 | 90 | 67 |
| 65 | 25 | 7 | 26 | 33 | 18 | 21 | 28 | 24 |
| 66 | 24 | 8 | 37 | 45 | 16 | 18 | 33 | 23 |
| 81 | 30 | 26 | 35 | 61 | 4 | 43 | 87 | 57 |
| 107 | 5 | 1 | 20 | 21 | 4 | 5 | 20 | 8 |
| 227 | 28 | 22 | 41 | 63 | 6 | 35 | 79 | 48 |
| 239 | 36 | 23 | 16 | 39 | 13 | 59 | 64 | 61 |

**Table 2: Full results for annotation using Lodestar's annotation system. Base refers to the base set of matchers, matching single words and phrases. Plural refers to the base set of matchers and the plural matcher. Synonym refers to the base set of matchers and the synonym matcher. Finally, All refers to all matchers: base, plural and synonym combined.**

the textbook with actual content. This excludes pages such as the title page and acknowledgements, ensuring that the pages contain content relevant to the subject matter. This was done by noting the first page of the first chapter and the last page of the final chapter and using a random number generator to select 7 pages within these bounds. The selected pages were 15, 65, 66, 81, 10, 227, 239. These page numbers are the numbers listed on the pages in the physical copy, or the bottom of each page in the PDF. The ontology used was the Data Mining Optimization (DMOP) Ontology [12].

Each of the seven pages was converted into HTML using the Lodestar PDF parser. They were then run through the four analyzers: base, plural, synonym and all. The total relevant elements in the text were curated by the researchers. We manually annotated the pages, selecting terms that are relevant to data mining, such as data set and annotation. We further selected terms that existed in the ontology, such as data. After Lodestar annotated the text, we compared the elements it annotated with the elements we selected to remove any false negatives in our manual annotation. This was done to produce the total relevant term counts listed in Table 2. Lodestar's annotation was then tallied to produce the results in Table 2.

## 5.2 Results and Discussion

Lodestar experimental results using precision, recall and $F_1$ are given in Figure 4. Table 2 shows the full results.

From the results in Figure 4, the base and plural matcher performs better than purely the base alone, achieving a precision of

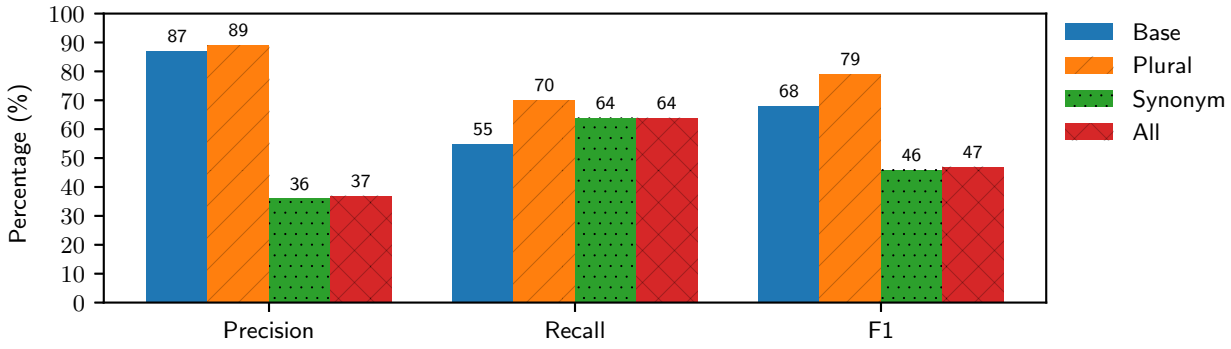


**Figure 4: The average precision, recall and $F_1$ results over all 7 pages for each of the four cases.**

89% and a recall of 70%. These results are encouraging, when considered in relation to those achieved by other annotation systems such as Ontea. This is important considering that Ontea's results were produced from an ontology generated from the text it was annotating, while Lodestar used a separately created ontology and text. Further study into the performance of Lodestar compared to other annotation systems is needed to determine exactly how it compares. This is due to the fact that the data set used by Laclavik, et al. (2009) is no longer available and neither is the source code. Thus, making it impossible to annotate the pages using Ontea.

The less effective recall performance in Lodestar could be an artefact of the automatic generation of regular expressions for matching. The automatic nature of the generation reduces Lodestar's ability to handle corner cases in the text and ontology. An example is the class "CorrelationSimilarity" in the DMOP ontology. This does not match "correlation" in the text as it does not know to match only half the element name. It would match "correlation similarity", however, as it is not capable of breaking the element name up and matching it against a multi-word term in the text. It is likely that comparing only part of the element name would result in a high number of false positives, reducing the precision.

Some false positives include words like "project" and "course", which the ontology uses as nouns, while in the text they are used as verbs (project) or colloquially (course). Further, while still the same part of speech, some matches are used in different ways in the text and ontology. For example, the word "series" in the text: "This is the participant's performance on a series of responsiveness tests." (page 107 of Data Mining for the Masses). The DMOP ontology uses the word "Series" in the statistical sense, to mean a list of numbers, while the textbook means a list of tests.

The synonym matcher reduces the accuracy of the base-case matches by 21%. This could be due to synonyms not taking into account what part of speech the term is or synonyms being too generalised or common to be of use. This can be seen by comparing the precision of the base-case and that of the synonym matcher. There are numerous false positive matches found by using synonyms, such as "perform" and "very". This is further confirmed in the synonym matcher results in Table 2, where the high number of false positives on each page are recorded. The increase in recall shows that there is a benefit to matching synonyms if the false positives can be identified and removed, as it does match more of the positive elements. This could be achieved by integrating a part of speech tagger into the synonym matcher, but due to time limitations, this is yet to be implemented in Lodestar.

There was also some synonyms that were missed, such as "attribute", used in the textbook, as a synonym of "feature" in the ontology. This missed synonym was due to WordNet not including "attribute" as a synonym for "feature", which could be a limitation of WordNet's breadth around scientific subjects [19].

Matching plurals improves the annotation for both precision and recall. This is expected, as plurals do not suffer the same generalisation issue that synonyms do. Plurals continue to use the same base word as the singular version, thus ensuring that common words are not also matched.

## 6 CONCLUSIONS

We developed a digital smart textbook platform, Lodestar, to annotate source textbooks using an ontology and render it to users. We have evaluated Lodestar's annotation system, which has shown promising results. While the synonym matcher reduced Lodestar's annotation ability, the best results were achieved with the base and plural matchers combined.

While Lodestar currently presents as a solid foundation, with better annotation and generalized textbook support, there are more features that require implementing if Lodestar is going to be used more effectively in schools and universities. For example, adding study notation features such as highlighting and margin notes [15]. These combined with smart features such as question and answer generation, as seen in "Inquire Biology" [2], would make Lodestar much more useful in educational institutions.

## REFERENCES

[1] Victor R Basil and Albert J Turner. 1975. Iterative enhancement: A practical technique for software development. *IEEE transactions on software engineering* SE-1, 4 (1975), 390–396.
[2] Vinay K Chaudhri, Britte Cheng, Adam Overtholtzer, Jeremy Roschelle, Aaron Spaulding, Peter Clark, Mark Greaves, and Dave Gunning. 2013. Inquire Biology: A textbook that answers questions. *AI Magazine* 34, 3 (2013), 55–72.
[3] Gobinda Chowdhury. 2012. How digital information services can reduce greenhouse gas emissions. *Online Information Review* 36, 4 (2012), 489–506.
[4] Cyril W Cleverdon, Jack Mills, and E Michael Keen. 1966. Factors determining the performance of indexing systems,(Volume 1: Design). *Cranfield: College of*

*Aeronautics* (1966), 28.
[5] Michelle De Bruyn. 2014. The protection of personal information (POPI) act: impact on South Africa. *International Business & Economics Research Journal* 13, 6 (2014), 1315–1340.
[6] Nomusa Dlodlo and JP Tolmay. 2012. An architecture for eBook provision to South African schools. In *14th Annual Conference on World Wide Web Applications, Durban, 7-9 November 2012*. ZA-WWW, 4–15.
[7] Andreas Doms and Michael Schroeder. 2005. GoPubMed: exploring PubMed with the gene ontology. *Nucleic acids research* 33, suppl_2 (2005), W783–W786.
[8] Giselle Du Plessis and Melanie Wiese. 2014. The battle of the e-textbook: libraries' role in facilitating student acceptance and use of e-textbooks. *South African Journal of Libraries and Information Science* 80, 2 (2014), 17–26.
[9] Bruce Fuller, Donald B Holsinger, David Baker, Rosemary Bellew, Richard Bennett, Prema Clarke, Brunhilda Forlemu, Ben H Fred-Mensah, Rosalind Michahelles, Pablo Stansbery, et al. 1993. *Secondary education in developing countries*. Education and Social Policy Department, World Bank.
[10] Thomas R Gruber. 1993. A translation approach to portable ontology specifications. *Knowledge acquisition* 5, 2 (1993), 199–220.
[11] David Janzen and Hossein Saiedian. 2005. Test-driven development concepts, taxonomy, and future direction. *Computer* 38, 9 (2005), 43–50.
[12] C Maria Keet, Agnieszka Ławrynowicz, Claudia d'Amato, Alexandros Kalousis, Phong Nguyen, Raul Palma, Robert Stevens, and Melanie Hilario. 2015. The data mining optimization ontology. *Journal of web semantics* 32 (2015), 43–53.
[13] Michal Laclavik, Martin Šeleng, Marek Ciglan, and Ladislav Hluchy. 2009. Ontea: Platform for pattern based automated semantic annotation. *Computing and informatics* 28, 4 (2009), 555–579.
[14] Jean-Baptiste Lamy. 2017. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial intelligence in medicine* 80 (2017), 11–28.
[15] Monica Landoni and Forbes Gibb. 2000. The role of visual rhetoric in the design and production of electronic books: the visual book. *The electronic library* 18, 3 (2000), 190–201.
[16] Tobias Lauinger, Abdelberi Chaabane, Sad Arshad, William Robertson, Christo Wilson, and Engin Kirda. 2017. Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web. In *Proceedings of the 24th Annual Network and Distributed System Security Symposium*. Internet Society.
[17] Avraham Leff and James T Rayfield. 2001. Web-application development using the model/view/controller design pattern. In *Proceedings of the 5th IEEE International Enterprise Distributed Object Computing Conference*. IEEE, 118–127.
[18] Marlaine E Lockheed, Stephen C Vail, and Bruce Fuller. 1986. How textbooks affect achievement in developing countries: Evidence from Thailand. *Educational Evaluation and Policy Analysis* 8, 4 (1986), 379–392.
[19] Bernardo Magnini, Carlo Strapparava, Giovanni Pezzulo, and Alfio Gliozzo. 2002. Comparing ontology-based and corpus-based domain annotations in wordnet. In *Proceedings of the First International WordNet Conference*. 21–25.
[20] George A Miller. 1995. WordNet: a lexical database for English. *Commun. ACM* 38, 11 (1995), 39–41.
[21] Jeanne Moulton. 1994. How do teachers use textbooks and other print materials: A review of the literature. *The Improving Educational Quality Project, South Africa* (1994).
[22] H-M Müller, Kimberly M Van Auken, Yuling Li, and Paul W Sternberg. 2018. Textpresso Central: a customizable platform for searching, text mining, viewing, and curating biomedical literature. *BMC bioinformatics* 19, 1 (2018), 94.
[23] Matthew North. 2012. *Data mining for the masses*. Global Text Project Athens.
[24] Aannemarie Sullivan Palinscar and Ann L Brown. 1984. Reciprocal teaching of comprehension-fostering and comprehension-monitoring activities. *Cognition and instruction* 1, 2 (1984), 117–175.
[25] Hilary E Pearson. 2000. Open source licences: Open source—the death of proprietary systems? *Computer Law & Security Review* 16, 3 (2000), 151–156.
[26] Ronald Rivest. 1992. The MD5 message-digest algorithm. (1992). Issue RFC1321.
[27] Amanda J Rockinson-Szapkiw, Jennifer Courduff, Kimberly Carter, and David Bennett. 2013. Electronic versus traditional print textbooks: A comparison study on the influence of university students' learning. *Computers & Education* 63 (2013), 259–266.
[28] Bella Ross, Ekaterina Pechenkina, Carol Aeschliman, and Anne-Marie Chase. 2017. Print versus digital texts: understanding the experimental research and challenging the dichotomies. *Research in Learning Technology* 25 (2017), 1976–1981.
[29] Suzanne Sackstein, Linda Spark, and Amy Jenkins. 2015. Are e-books effective tools for learning? Reading speed and comprehension: iPad® i vs. paper. *South African Journal of Education* 35, 4 (2015).
[30] Michel F Sanner et al. 1999. Python: a programming language for software integration and development. *J Mol Graph Model* 17, 1 (1999), 57–61.
[31] Irena Spasic, Sophia Ananiadou, John McNaught, and Anand Kumar. 2005. Text mining and ontologies in biomedicine: making sense of raw text. *Briefings in bioinformatics* 6, 3 (2005), 239–251.
[32] John Sweller. 1988. Cognitive load during problem solving: Effects on learning. *Cognitive science* 12, 2 (1988), 257–285.
[33] Cornelis J Van Rijsbergen. 1979. Information Retrieval. 2nd. Newton, MA.
[34] Xiaoyun Wang and Hongbo Yu. 2005. How to break MD5 and other hash functions. In *Annual international conference on the theory and applications of cryptographic techniques*. International Association for Cryptologic Research, 19–35.