Key Phrase Extractor Web Application

Stefan Radic Webster, hn19405@bristol.ac.uk

Abstract

A web application has been developed to provide an user interface for a previously trained Natural Language Processing (NLP) model to extract key phrases from scientific publications. A basic UI allows a user to enter text which is input into the NLP model to identify key phrases. The text and key phrases are stored in a SQLite database allowing multiple texts with accompanying key phrases to be stored. The web application was created using the Flask framework in Python, following good software engineering practices such as version control and unit testing. The code is available from GitHub[1].

I. Introduction

The back end of the web application consists of an Natural Language Processing (NLP) model which formed the solution to SemEval 2017 Task 10 to extract key phrases from scientific publications[2]. The NLP task consisted of sequence tagging where every token in a paragraph is labelled being at the beginning, inside or outside a key phrase. A Hidden Markov Model (HMM) was developed using NLTK as well as a Bi-LSTM-CRF model with embeddings using Flair. The Flair model showed superior performance to extract key phrases from a passage of text compared with the HMM with the Flair model achieving an F1 score of 0.62 on test data compared with 0.44 for the HMM. However, the Flair model size and time to extract key phrases deems it less suitable for the web application, so the HMM was used. The report follows the process of building a web application to progress towards deploying the NLP model.

II. Software Engineering

There exists a range of principles, methods and tools that promote the development of high-quality software in an efficient way that meets design specifications. The good practices in software engineering that were utilised for the project are discussed below.

A. Virtual environments

A Python virtual environment was created for the project using Anaconda. This is an isolated area where Python packages required for the project can be downloaded and stored. Creating a virtual environment for each project prevents issues arising from conflicting dependencies and compatibility issues between the different versions of libraries, or even Python itself.

B. Version Control

Git was used for version control for the project which keeps a record of changes made to files and directories and facilitates moving forwards or backwards through different versions. Although Git is very useful for collaborative work as teams can work together, share and modify the files all at the same time, for this individual project Git was used primarily as a backup system during development.

C. Integrated Development Environment

Pycharm was used as the IDE for the project. An IDE is a software development environment used to write good quality programs with code insight and also supports debugging and compiling. An IDE is especially useful when coding in various languages for example Python, HTML and CSS as used in the project. Pycharm supports web developing using HTML, CSS and JavaScript and allows live editing which instantaneously shows code changes on a web browser. Pycharm also supports web frameworks such as Flask as used for the project.

D. Web Framework

A Web Application Framework is a collection of libraries and modules that enables web development without having to consider low-level details such as protocols and thread management. Flask is a Python web framework which keeps the core simple with little code needed to get a basic app up and running. Flask has the following dependencies which are managed by the virtual environment.

- *Werkzeug*: implements WSGI (standard Python interface between applications and servers)
- *Jinja*: template language that renders the pages the application serves
- *MarkupSafe, ItsDangerous and Click*

[1]https://github.com/sradicwebster/keyphrase_web_application
[2]https://github.com/sradicwebster/dialogue_and_narrative

III. Key Phrase Extractor Web Application

The key phrase extractor web application allows users to enter text, for example a journal publication abstract, and displays the identified unique key phrases. The web application allows users to upload and view key phrases for multiple texts and delete them when no longer needed (screenshots available in Appendix A). The construction of the web application followed the PalletsProjects tutorial to produce a blog named Flaskr[3] and has a project layout as shown in figure 1.

```
/project_folder
├── keyphrases/
│   ├── __init__.py
│   ├── db.py
│   ├── schema.sql
│   ├── keyphrase.py
│   ├── results.py
│   ├── templates/
│   │   ├── base.html
│   │   ├── keyphrase/
│   │   │   └── input.html
│   │   └── results/
│   │       ├── index.html
│   │       └── view.html
│   └── static/
│       └── style.css
├── tests/
│   ├── conftest.py
│   ├── data.sql
│   ├── test_factory.py
│   ├── test_db.py
│   ├── test_keyphrase.py
│   └── test_results.py
├── venv/
├── setup.py
└── MANIFEST.in
```

Fig. 1: Project Layout

The `__init__.py` file contains the application factory which creates a Flask instance which acts as a central registry containing everything about the application (configurations, URLs, templates, etc). Running the application in development mode restarts the server whenever changes are made to the code. The key phrase application used the built-in Python SQLite database to store texts and identified key phrases.

Flask Blueprints organise a group of related view functions which are the part of the code that responds to or generates requests from the application URL. Views are registered with the blueprints which in turn are registered with the application in the factory function. The key phrase application has two blueprints, one for uploading/entering texts and the other to view the key phrases.

When a user visits the `keyphrase/input` URL, the input view returns HTML with a form to enter a text along with a title. Error messages are produced if either of the cells remain blank or if the title is already used in the database. Upon submitting the form, the text is input to the imported NLP model (NLTK HMM tagger) which outputs a list of key phrases. The title, text and key phrases are inserted into the SQLite database and then the URL is redirected to the results index page.

The `results/index` page shows a list of the uploaded texts and the number of identified key phrases. The view buttons reveal the key phrases and also supports deleting the entry from the database. Navigation buttons allow the user to move between the different web pages.

A. Templates and static files

The display of the application on a web browser is achieved by use of templates which are rendered in Flask using Jinja. Templates contain static data as well as placeholders to send and receive data. The web three pages in the application have the same basic layout as is defined in `templates/base.html` which is extended for specific pages. A static CSS file was added to add style to the HTML templates.

[3]https://flask.palletsprojects.com/en/1.1.x/tutorial/

B. Testing

Unit testing ensures the key phrase web application works as expected according to the written code. Test coverage refers to the amount of source code that was run during testing, with a high value reducing the likelihood of bugs. However, 100% coverage does not mean application is error free.

Testing was carried out using pytest[4] following the Flask blog tutorial. Testing in Flask is achieved by use of a test client that simulates a user sending and receiving requests. The `tests/conftest.py` file contains the client and other pytest fixtures that are used by each test. Each test creates a new temporary database file which is automatically populated with data from the `data.sql` file and then used in the tests. Unit tests are functions that start with test_ and ideally cover all application functions and branches such as "if" statements. The unit test functions are located within Python files in the test directory and cover testing of the application factory, SQL database as well as the key phrase and results blueprints. The coverage[5] report is shown in table I.

Name	Statements	Missing	Coverage
tests/conftest.py	20	0	100%
tests/test_db.py	19	0	100%
tests/test_factory.py	4	0	100%
tests/test_keyphrase.py	11	0	100%
tests/test_results.py	17	0	100%
TOTAL	71	0	100%

TABLE I: Coverage report

C. Installing and Building the Application

In order to deploy the project on other virtual environments or machines, the project first needs to be made installable. Installing the project also manages the dependencies as defined in the `setup.py` file and isolates the test environment from the development environment.

To deploy the project then a distribution file needs building. The Python standard for distribution is the wheel format[6]. The file `keyphrases-1.0.0-py3-none-any.whl` in the GitHub repository can be downloaded, installed and run on other machines.

IV. Further Good Practices

A. Testing during Development

During the project, the unit tests were written after the application had been and any bugs found were acted upon. Good software engineering practice would involve adding a unit test alongside writing new code and then running the test suite to ensure the new code did not introduce any bugs. As a step further, test-driven development involves writing the docstring and unit test for a new function before the main body of the function allowing immediate verification that the code works as expected.

B. Continuous Integration

Larger software development occurs in teams and collaboration can be achieved by continuous integration. With lots of people simultaneously making and merging changes and sending pull requests, manual reviewing and verification become infeasible. CI solves this problem by using automated verification and testing with tests being run every time a developer commits work before being integrated. This frequent commit, build and test pipeline can be implemented using Jenkins, an automation server which promotes rapid and error free software development.

V. Conclusion

The developed web application serves as a front end user interface for a key phrase identification NLP model. The project was conducted following good software engineering practice using virtual environments, Pycharm for development and Git for version control. The application was constructed using the Python framework Flask with use of pytest for unit testing. Further good practices highlighted for use in future projects were test-driven development and continuous integration for large, collaborative projects.

[4]https://docs.pytest.org/en/latest/
[5]https://coverage.readthedocs.io/en/coverage-5.1/
[6]https://pypi.org/project/wheel/

References

[1] Real Python, *Python Virtual Environments: A Primer*,
https://realpython.com/python-virtual-environments-a-primer/reader-comments
[2] Anirudh Rao, *PyCharm Tutorial: Writing Python Code In PyCharm (IDE)*,
https://www.edureka.co/blog/pycharm-tutorial
[3] Chrys Woods, *Version Control using Git*,
https://chryswoods.com/beginning_git/README.html
[4] GitHub Guides, *Hello World*,
https://guides.github.com/activities/hello-world/
[5] Pallets Projects, *Flask*,
https://flask.palletsprojects.com/en/1.1.x/
[6] Pallets Projects, *Flask Tutorial*,
https://flask.palletsprojects.com/en/1.1.x/tutorial/
[7] Kenneth Reitz, *Testing Your Code*,
https://docs.python-guide.org/writing/tests/
[8] Matt Williams, *Best practices in software engineering*,
https://milliams.gitlab.io/software_engineering_best_practices/index.html
[9] Joaquín Menchaca, *Jenkins CI Pipeline with Python*,
https://medium.com/@Joachim8675309/jenkins-ci-pipeline-with-python-8bf1a0234ec3
[10] Martin Heller, *What is Jenkins? The CI server explained*,
https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html

Appendix A
Web application screenshots

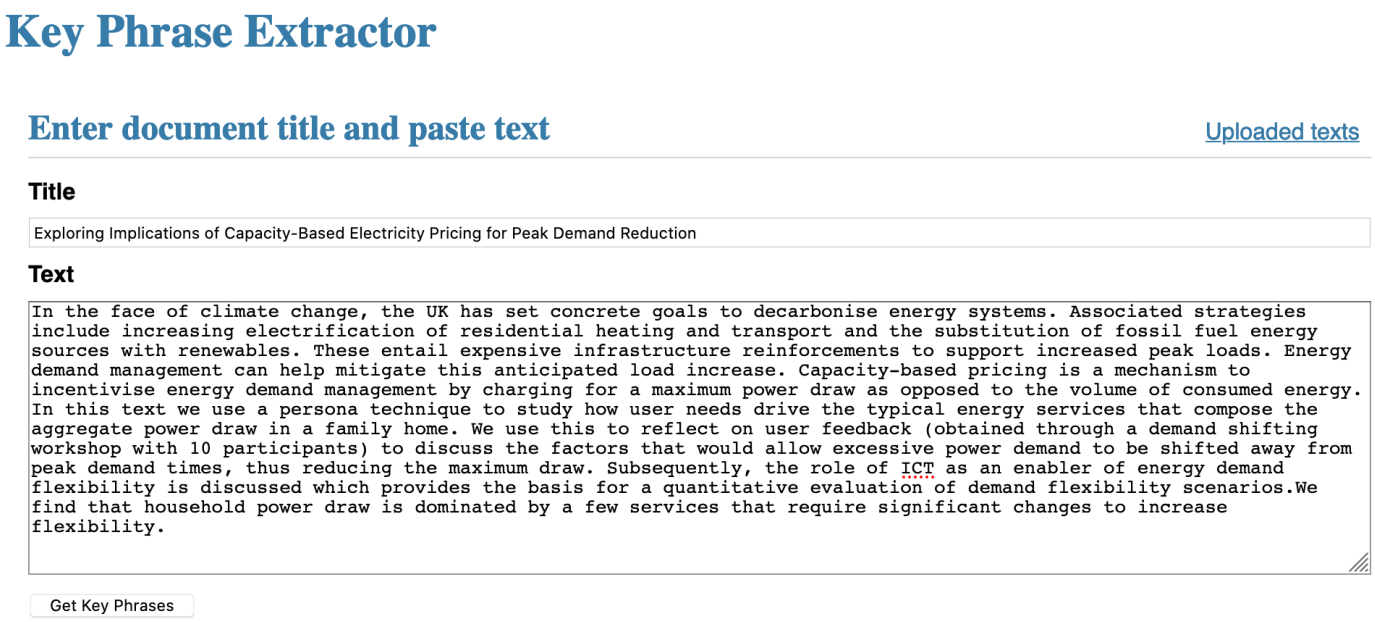

Fig. 2: Input page to enter text

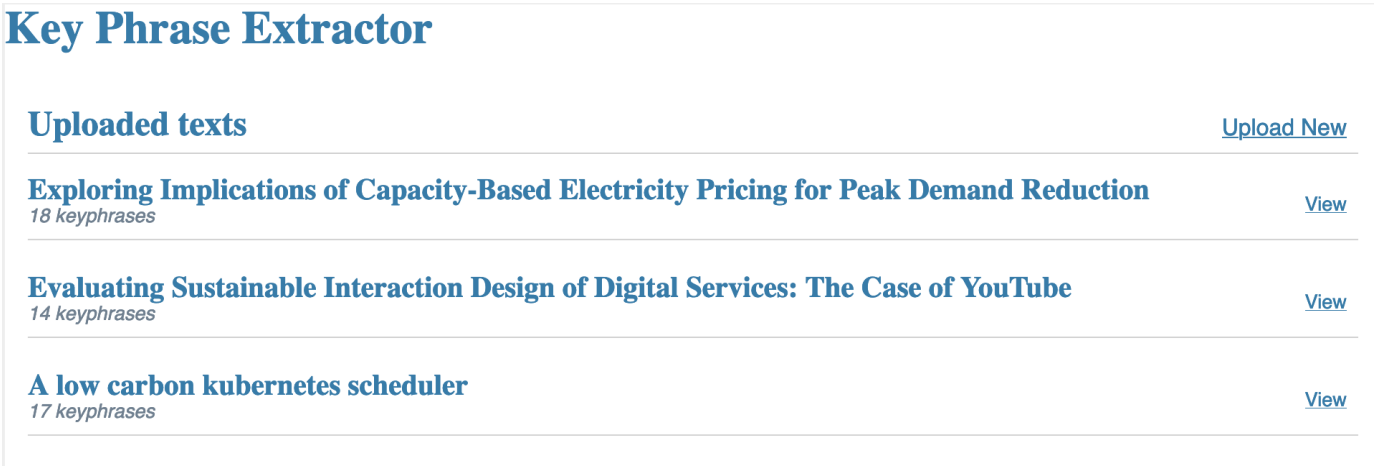

Fig. 3: Index page listing all uploaded documents

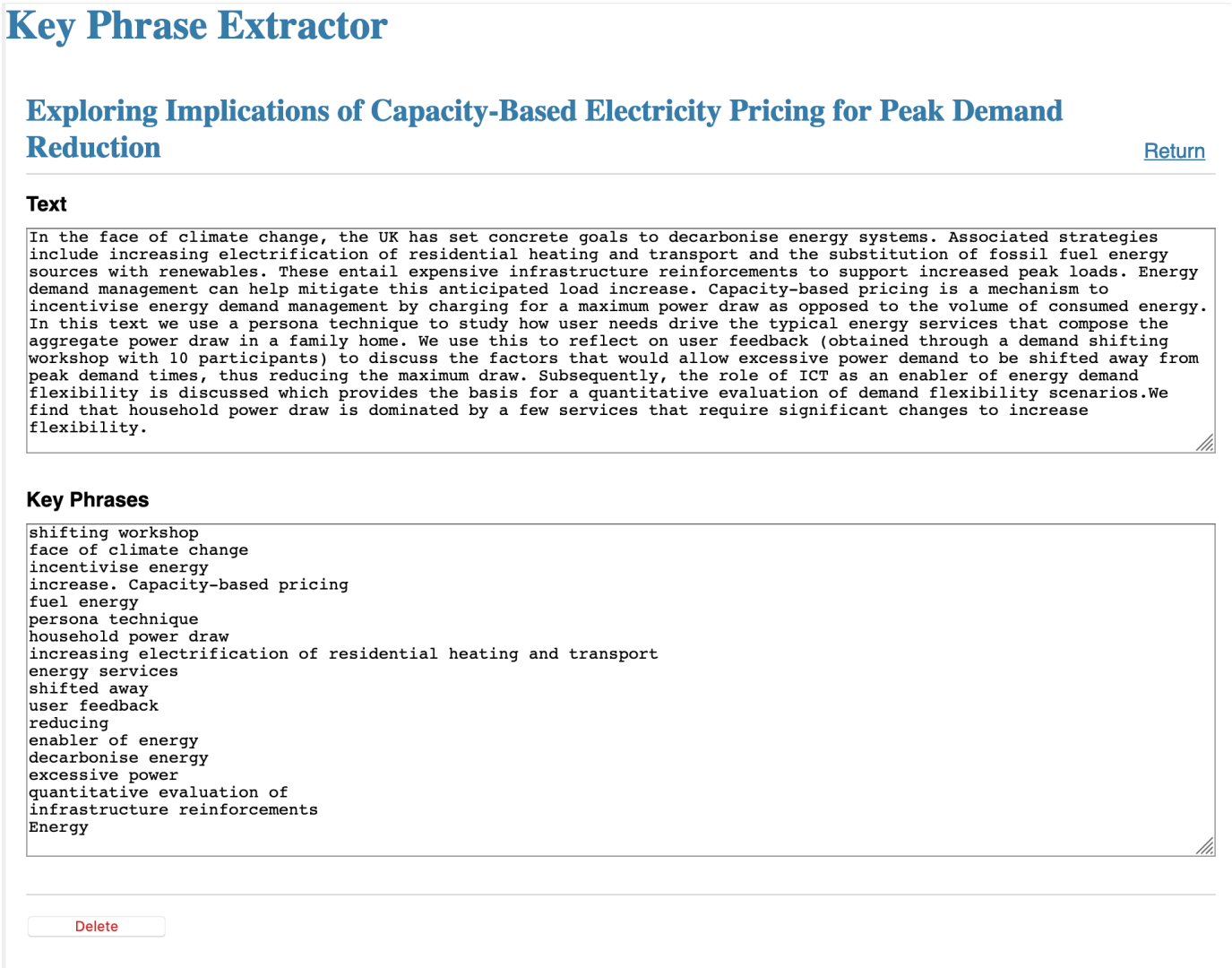

Fig. 4: View and delete key phases