Project: Multi-source code search

Knowledge Analysis & Management

Paolo Tonella

paolo.tonella@usi.ch

Goal of the project

*Develop a **search engine** that can query a large Python code repository using multiple sources of information*

Multi Source Code Search

Optimiser that implements the Adadelta algorithm

```
#1 Python class: Adadelta
   File: tensorflow/python/keras/optimizers.py
   Line: 354

#2 Python class: AdadeltaOptimizer
   File: tensorflow/python/training/adadelta.py
   Line: 28
```

TensorFlow

2,817 py files
11,881 searchable code entities:

- classes
- functions
- methods

Steps of the project

1. [**extract data**] extract names/comments of Python classes, methods, functions
2. [**train search engines**] represent code entities using four embeddings: frequency, TF-IDF, LSI and Doc2Vec and report the entities most similar to the given query string
3. [**evaluate search engines**] define the ground truth for a set of queries and measure average precision and recall for the four search engines
4. [**visualize query results**] for LSI and Doc2Vec, project the embedding vectors of queries and of the top-5 answers to a 2D plot using t-SNE

Subject

TensorFlow: open source framework for machine learning / deep learning

The code is provided in Github Classroom: do not download it from any existing repository

Tools and libraries

Gensim:

```
pip install --upgrade gensim
/Applications/Python\ 3.7/Install\ Certificates.command
```

documentation: https://radimrehurek.com/gensim/apiref.html; see also examples from lecture slides: `THEO-10-gensim.pdf`

AST:

documentation: https://docs.python.org/2/library/ast.html; https://greentreesnakes.readthedocs.io; see e.g. `ast.NodeVisitor`

t-SNE:

```
pip install sklearn / pip install scikit-learn
pip install seaborn
```

documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
https://builtin.com/data-science/tsne-python
https://towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d1

Step 1: Extract data

Extract all names of top level **classes**, **functions** and class **methods** in `*.py` files found under input directory `<dir>` and any directory below `<dir>`. Use AST for Python parsing. Save the extracted info into a CSV file. When available, extract also the **comment line** following the class, function or method declaration.

	name	file	line	type	comment
0	UserInputError	../tensorflow/configure.py	74	class	
1	is_windows	../tensorflow/configure.py	78	function	
2	is_linux	../tensorflow/configure.py	82	function	
3	is_macos	../tensorflow/configure.py	86	function	
4	is_ppc64le	../tensorflow/configure.py	90	function	
5	is_cygwin	../tensorflow/configure.py	94	function	
6	get_input	../tensorflow/configure.py	98	function	
7	symlink_force	../tensorflow/configure.py	109	function	"""Force symlink,
8	sed_in_place	../tensorflow/configure.py	126	function	"""Replace old string with new string in file.

Black list:

- name starts with '_'
- name is 'main'
- name contains the word 'test' (including all case variants, such as 'TEST')

* If a class is blacklisted, its methods get blacklisted as well

Step 1: Extract data

Hints:

- You can use `ast.NodeVisitor` for Python parsing; attribute `lineno` gives you the line number of each entity; inner functions/methods can be skipped by just avoiding to trigger a recursive `generic_visit` call.
- To extract the comment line, you can:
  - read the source file up to the line number of each entity; then, read the next lines until the declaration ends (i.e., `'):'` or `'->'` is found in Python): if the next line starts with `"""`, in Python it is a comment line to be extracted
  - use this function call: `comment = ast.get_docstring(node)`
  - locate the following types of nodes: `Str(s='comment')` (Python 3.7 and earlier) or `Constant(value='comment', kind=None)` (Python 3.8 and later)

Github

1. If you do not have a Github account, go to GitHub.com and create one
2. We use **GitHub Classroom**: you can find instructions on how to access it on iCourse ("Project 01 - Guidelines", top of the page)

   - https://classroom.github.com/a/d9ABIPO1

Useful git commands

```
git clone <URL>
git add file.py
git commit -m “message”
git push
git pull
git status
```

Code repository

- Code should be runnable and should produce the results expected for the task
- To run the code there should be no need for manual changes
- A README file should be provided that describes how to run the code and access the results
- All the files and folders necessary to run the code should be contained inside the root folder of the project
- Remove/disable all the excessive print statements that were possibly used to debug the code
- Code should be clean and sufficiently commented; function and variable names should be meaningful and self-explanatory
- If any libraries are used that are not native to Python and are not mentioned in project slides, list them in the README file
- For Python scripts, use the naming that were suggested in project slides (you can use '_' instead of '-')
- https://github.com/MiWeiss/Project-Tips

Project report

Section 1: Data extraction

- report figures about the extracted data (e.g., number of files; number of code entities by type)

Section 2: Training of search engines

- report and comment an example of query

Section 3: Evaluation of search engines

- report recall and average precision for each of the four search engines; comment the differences among search engines

Section 4: Visualization of query results

- include and comment the t-SNE plots for LSI and for Doc2Vec

Github repository: Python code

- Python code for data extraction
- Python code for training of search engines
- Python code for evaluation and visualization of query results

By the deadline, submit the **report in PDF format and the Github commit ID.**
Name the PDF file according to the following pattern: “report_01_lastname”, where lastname is student’s lastname.
Refer to file **Project 01 - Report Template (pdf)** on icoursi for a template of the report (Latex sources are also provided)

Project: Multi-source code search

Knowledge Analysis & Management

Paolo Tonella

paolo.tonella@usi.ch

Project: Multi-source code search

Knowledge Analysis & Management

Paolo Tonella

paolo.tonella@usi.ch

Goal of the project

*Develop a **search engine** that can query a large Python code repository using multiple sources of information*

Multi Source Code Search

Optimiser that implements the Adadelta algorithm

```
#1 Python class: Adadelta
   File: tensorflow/python/keras/optimizers.py
   Line: 354

#2 Python class: AdadeltaOptimizer
   File: tensorflow/python/training/adadelta.py
   Line: 28
```

TensorFlow

2,817 py files
11,881 searchable code entities:

- classes
- functions
- methods

Steps of the project

1. [**extract data**] extract names/comments of Python classes, methods, functions
2. [**train search engines**] represent code entities using four embeddings: frequency, TF-IDF, LSI and Doc2Vec and report the entities most similar to the given query string
3. [**evaluate search engines**] define the ground truth for a set of queries and measure average precision and recall for the four search engines
4. [**visualize query results**] for LSI and Doc2Vec, project the embedding vectors of queries and of the top-5 answers to a 2D plot using t-SNE

Step 2: Train search engines

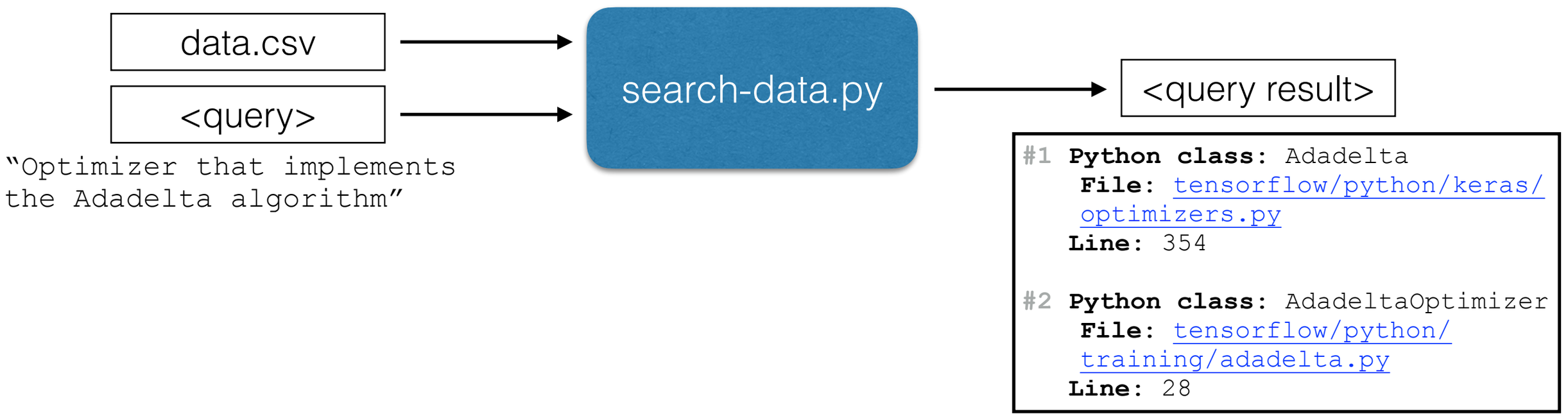
Create a corpus from the code entity names and comments:

1. split entity names by camel-case and underscore (e.g., go_to_myHome -> [go, to, my, home])
2. filter stopwords = {test, tests, main, this,..} (enlarge this list with appropriate candidates)
3. convert all words to lowercase
4. analyse whether you need the whole comment for training or if cleaning (e.g. removing code snippets)/using only a part of it could be beneficial for performance

Represent entities using the following vector embeddings:

- **FREQ**: frequency vectors
- **TF-IDF**: TF-IDF vectors
- **LSI**: LSI vectors with k = 300
- **Doc2Vec**: doc2vec vectors with k = 300

Given a query string, for each embedding print the top-5 most similar entities (entity name, file name, line of code), based on cosine similarity



Step 2: Train search engines

Hints:

- Use regular expressions (`re.search`) for camel-case splitting
- Refer to the Python examples in `THEO-10-gensim.pdf` and to the Python scripts mentioned in the slides and available on iCorsi
- Sort the documents in the corpus by similarity to get the top-5 entities most similar to the query for FREQ, TF-IDF, LSI
- Use function `most_similar` with `topn=5` to get the top-5 entities most similar to the query for Doc2Vec
- Save and reuse the trained models to avoid re-trainings each time your code is executed

Project report

Section 1: Data extraction

- report figures about the extracted data (e.g., number of files; number of code entities by type)

Section 2: Training of search engines

- report and comment an example of query

Section 3: Evaluation of search engines

- report recall and average precision for each of the four search engines; comment the differences among search engines

Section 4: Visualization of query results

- include and comment the t-SNE plots for LSI and for Doc2Vec

Github repository: Python code

- Python code for data extraction
- Python code for training of search engines
- Python code for evaluation and visualization of query results

By the deadline, submit the **report in PDF format and the Github commit ID.**
Name the PDF file according to the following pattern: “report_01_lastname”, where lastname is student’s lastname.
Refer to file **Project 01 - Report Template (pdf)** on icoursi for a template of the report (Latex sources are also provided)

Project: Multi-source code search

Knowledge Analysis & Management

Paolo Tonella

paolo.tonella@usi.ch

Project: Multi-source code search

Knowledge Analysis & Management

Paolo Tonella

paolo.tonella@usi.ch

Goal of the project

*Develop a **search engine** that can query a large Python code repository using multiple sources of information*

Multi Source Code Search

Optimiser that implements the Adadelta algorithm

```
#1 Python class: Adadelta
   File: tensorflow/python/keras/optimizers.py
   Line: 354

#2 Python class: AdadeltaOptimizer
   File: tensorflow/python/training/adadelta.py
   Line: 28
```

TensorFlow

2,817 py files
11,881 searchable code entities:

- classes
- functions
- methods

Steps of the project

1. [**extract data**] extract names/comments of Python classes, methods, functions
2. [**train search engines**] represent code entities using four embeddings: frequency, TF-IDF, LSI and Doc2Vec and report the entities most similar to the given query string
3. [**evaluate search engines**] define the ground truth for a set of queries and measure average precision and recall for the four search engines
4. [**visualize query results**] for LSI and Doc2Vec, project the embedding vectors of queries and of the top-5 answers to a 2D plot using t-SNE

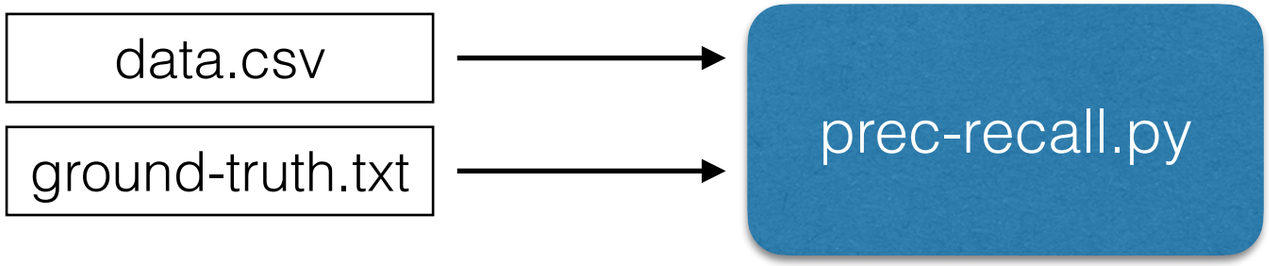
Step 3: Evaluate search engines

See the **ground-truth file** in the GitHub Classroom repository:

- it contains the reference triples (query, function/class name, file)

Measure **precision** and **recall** for each search engine (FREQ, TF-IDF, LSI, Doc2Vec):

- the ground truth entity is correctly reported by the search (TP) if it appears among the top-5 most similar entities (the entity must have the same name and file name);
- precision is 1 / POS where POS is the position of the correct answer (in case there is a correct answer among the top-5 entities) or 0 if the correct answer is not reported; POS ranges between 1 and 5;
- **average precision** is the average across all queries;
- **recall** is the number of correct answers (among the top-5, regardless of the position) over the total number of queries




```
Optimizer that implements the Adadelta algorithm
Adadelta
tensorflow/python/keras/optimizers.py

…
```

```
FREQ:
Average precision = 0.2
Recall = 0.5

TF-IDF:
Average precision = 0.5
Recall = 0.7

…
```

Step 3: Evaluate search engines

Hints:

- Convert query strings to lowercase before using them
- Remove trailing spaces by calling function `strip`

Project report

Section 1: Data extraction

- report figures about the extracted data (e.g., number of files; number of code entities by type)

Section 2: Training of search engines

- report and comment an example of query

Section 3: Evaluation of search engines

- report recall and average precision for each of the four search engines; comment the differences among search engines

Section 4: Visualization of query results

- include and comment the t-SNE plots for LSI and for Doc2Vec

Github repository: Python code

- Python code for data extraction
- Python code for training of search engines
- Python code for evaluation and visualization of query results

By the deadline, submit the **report in PDF format and the Github commit ID.**
Name the PDF file according to the following pattern: “report_01_lastname”, where lastname is student’s lastname.
Refer to file **Project 01 - Report Template (pdf)** on icoursi for a template of the report (Latex sources are also provided)

Project: Multi-source code search

Knowledge Analysis & Management

Paolo Tonella

paolo.tonella@usi.ch

Project: Multi-source code search

Knowledge Analysis & Management

Paolo Tonella

paolo.tonella@usi.ch

Goal of the project

*Develop a **search engine** that can query a large Python code repository using multiple sources of information*

Multi Source Code Search

Optimiser that implements the Adadelta algorithm

```
#1 Python class: Adadelta
   File: tensorflow/python/keras/optimizers.py
   Line: 354

#2 Python class: AdadeltaOptimizer
   File: tensorflow/python/training/adadelta.py
   Line: 28
```

TensorFlow

2,817 py files
11,881 searchable code entities:

- classes
- functions
- methods

Steps of the project

1. [**extract data**] extract names/comments of Python classes, methods, functions
2. [**train search engines**] represent code entities using four embeddings: frequency, TF-IDF, LSI and Doc2Vec and report the entities most similar to the given query string
3. [**evaluate search engines**] define the ground truth for a set of queries and measure average precision and recall for the four search engines
4. [**visualize query results**] for LSI and Doc2Vec, project the embedding vectors of queries and of the top-5 answers to a 2D plot using t-SNE

Step 4: Visualize query results

- Only for **LSI** and **Doc2Vec**, compute the embedding vectors of each query and of the corresponding top-5 most similar entities
- Use t-SNE (from package `sklearn`) to produce a 2D plot where each group (consisting of vectors for one query and the associated top-5 answers) are plot as points with the same colour (suggested parameters: `perplexity=2`, `n_iter=3000`)
- Use package `seaborn` to produce the 2D scatterplots

Step 4: Visualize query results

Hints:

- Get embedding vectors by index for the top-5 answers from the corpus for LSI (e.g., `idx, score = sims[i]`, then `lsi_corpus[idx]`)
- Get embedding vectors for Doc2Vec by calling `model.infer_vector` on vectors from the original corpus
- Example of creation of t-SNE plot:
  - Example 1
  - Example 2
  - Gensim lecture, file word2vec.py
- If the performance of the trained models is sufficiently good and the implementation is correct, top-5 answers should form a visible cluster with their corresponding queries from ground truth file

Project report

Section 1: Data extraction

- report figures about the extracted data (e.g., number of files; number of code entities by type)

Section 2: Training of search engines

- report and comment an example of query

Section 3: Evaluation of search engines

- report recall and average precision for each of the four search engines; comment the differences among search engines

Section 4: Visualization of query results

- include and comment the t-SNE plots for LSI and for Doc2Vec

Github repository: Python code

- Python code for data extraction
- Python code for training of search engines
- Python code for evaluation and visualization of query results

By the deadline, submit the **report in PDF format and the Github commit ID.**
Name the PDF file according to the following pattern: “report_01_lastname”, where lastname is student’s lastname.
Refer to file **Project 01 - Report Template (pdf)** on icoursi for a template of the report (Latex sources are also provided)

Project: Multi-source code search

Knowledge Analysis & Management

Paolo Tonella

paolo.tonella@usi.ch