Knowledge Management and Analysis
Project 02: Code Search

Davide Mammarella

Section 1 - Data Extraction

The first step of the project is to extract the data. This means extracting all names of top level **classes, functions** and class **methods** in *.py files found under input directory `<dir>` and any directory below `<dir>` and in addition, when available, comment lines following class/function/method declarations. In order to do this, I used the **AST** package to parse each Python file. In detail I used `ast.NodeVisitor`, defining a custom subclass able to visit classes, methods, functions and comments. All visited entities are extracted and filtered, via a blacklist, by removing names starting with "_", names containing the word "test" (and all it's case variants) and names that are "main". In addition, since the repository follows the PEP 8 style guide, I applied additional filters to the comments in order to make them a simple and direct explanation of the entity they refer to. First of all I removed everything after the first paragraph, i.e. I applied a regex that removes everything after the first line break. Subsequently I applied minor refinements, i.e. I removed the remaining examples, years and punctuation. All the extracted information are saved in a CSV file, an excerpt of which can be seen in Figure 1.

	name	file		type	comment
250	generate_raw_ops_doc	../tensorflow/tensorflow/tools/docs/generate2.py	97	function	Generates docs for tf.raw_ops
251	TfExportAwareVisitor	../tensorflow/tensorflow/tools/docs/generate2.py	134	class	A tf_export keras_export and estimator_export aware doc_visitor
252	build_docs	../tensorflow/tensorflow/tools/docs/generate2.py	176	function	Build api docs for tensorflow v2
253	build_md_page	../tensorflow/tensorflow/tools/docs/pretty_docs.py	36	function	Given a PageInfo object return markdown for the page
254	DocGeneratorVisitor	../tensorflow/tensorflow/tools/docs/doc_generator_visitor.py	28	class	A visitor that generates docs for a python object when __call__ed
255	set_root_name	../tensorflow/tensorflow/tools/docs/doc_generator_visitor.py	49	method	Sets the root name for subsequent __call__s
256	index	../tensorflow/tensorflow/tools/docs/doc_generator_visitor.py	55	method	A map from fully qualified names to objects to be documented
257	tree	../tensorflow/tensorflow/tools/docs/doc_generator_visitor.py	66	method	A map from fully qualified names to all its child names for traversal
258	reverse_index	../tensorflow/tensorflow/tools/docs/doc_generator_visitor.py	78	method	A map from id to the preferred fully qualified name
259	duplicate_of	../tensorflow/tensorflow/tools/docs/doc_generator_visitor.py	93	method	A map from duplicate full names to a preferred fully qualified name
260	duplicates	../tensorflow/tensorflow/tools/docs/doc_generator_visitor.py	107	method	A map from preferred full names to a list of all names for this symb…
261	do_not_generate_docs	../tensorflow/tensorflow/tools/docs/doc_controls.py	24	function	A decorator: Do not generate docs for this object
262	do_not_doc_inheritable	../tensorflow/tensorflow/tools/docs/doc_controls.py	105	function	A decorator: Do not generate docs for this method
263	for_subclass_implementers	../tensorflow/tensorflow/tools/docs/doc_controls.py	168	function	A decorator: Only generate docs for this method in the defining class
264	should_skip	../tensorflow/tensorflow/tools/docs/doc_controls.py	246	function	Returns true if docs generation should be skipped for this object
265	should_skip_class_attr	../tensorflow/tensorflow/tools/docs/doc_controls.py	264	function	Returns true if docs should be skipped for this class attribute
266	write_docs	../tensorflow/tensorflow/tools/docs/generate_lib.py	40	function	Write previously extracted docs to disk
267	add_dict_to_dict	../tensorflow/tensorflow/tools/docs/generate_lib.py	221	function	<null>
268	DocControlsAwareCrawler	../tensorflow/tensorflow/tools/docs/generate_lib.py	246	class	A docs_controls aware API-crawler
269	extract	../tensorflow/tensorflow/tools/docs/generate_lib.py	255	function	Extract docs from tf namespace and write them to disk
270	process_title	../tensorflow/tensorflow/tools/docs/generate_lib.py	285	method	<null>
271	build_doc_index	../tensorflow/tensorflow/tools/docs/generate_lib.py	298	function	Build an index from a keyword designating a doc to _DocInfo objects
272	make_md_link	../tensorflow/tensorflow/tools/docs/generate_lib.py	340	method	<null>
273	process	../tensorflow/tensorflow/tools/docs/generate_lib.py	351	method	Index a file reading from full_path with base_name as the link
274	process_section	../tensorflow/tensorflow/tools/docs/generate_lib.py	364	method	<null>
275	process_line	../tensorflow/tensorflow/tools/docs/generate_lib.py	368	method	Index the file and section of each symbol reference
276	update_id_tags_inplace	../tensorflow/tensorflow/tools/docs/generate_lib.py	397	function	Set explicit ids on all second-level headings to ensure back-links w…

Figure 1: Excerpt of data.csv

During entity extraction, metrics were also collected. They can be seen in table 1 and the numbers reflect a large repository, such as the one analysed: **Tensorflow**.

Type	Number
Python files	2817
Classes	1883
Functions	4580
Methods	6009
Total entities	12472

Table 1: Count of created classes and properties.

During further extraction experiments, I noticed that many entities were repeated. Therefore, to make future analyses faster and more accurate, I chose to ignore entities that had the same *name, type* and *comment*, leaving the *files* containing these entities outside the filter as there were identical helper classes/methods/functions that were repeated within different files. After such additional refinement we obtain the results shown in table 2.

Type	Number
Python files	2817
Classes	1785
Functions	4350
Methods	4285
Total entities	10420

Table 2: Count of created classes and properties after refinement.

Section 2: Training of search engines

The second step of the project is to train the search engines. Initially, I created a corpus from the code entity name and comments, processing the data obtained in step 1. In detail, I separated the entity names by camel case and underscore, then I filtered them using "test", "tests" and "main" as stopwords and finally, I converted everything to lowercase. No further refinement is necessary as the comments had already undergone this process in the previous step.
Once the corpus is created, I processed it according to the standards of each search engine in order to train them. The search engines used in the project are: **FREQ, TF IDF, LSI, DOC2VEC** and all the models obtained are also saved for possible reuse. Finally, they were queried according to the following query: *"Optimizer that implements the Adadelta algorithm"* and the results of the 5 most similar entities for each search engine are shown in figures 2, 3, 4, 5. The ground truth for the query are the classes *"Adadelta"* and *"AdadeltaOptimizer"*, with the respective files: *"tensorflow/python/keras/optimizers.py"* and *"tensorflow/python/training/adadelta.py"*.

The results show that all search engines correctly classify at least one of the two queries in the ground truth, placing it first in the ranking. In detail, we can say that FREQ and TF IDF proved to be the best search engines, correctly placing the searched ground truths in the first two positions, followed by LSI which managed to place them in the top 5 but at positions 1 and 5 and finally we have DOC2VEC as it failed to correctly classify the Adadelta ground truth.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 FREQ  top-5 most similar entities
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1   Python class:    Adadelta
    File:    ../tensorflow/tensorflow/python/keras/optimizer_v2/adadelta.py
    Line:    32
--------------------------------------------------------------------------------
2   Python class:    AdadeltaOptimizer
    File:    ../tensorflow/tensorflow/python/training/adadelta.py
    Line:    29
--------------------------------------------------------------------------------
3   Python class:    Nadam
    File:    ../tensorflow/tensorflow/python/keras/optimizer_v2/nadam.py
    Line:    34
--------------------------------------------------------------------------------
4   Python class:    FtrlOptimizer
    File:    ../tensorflow/tensorflow/python/training/ftrl.py
    Line:    29
--------------------------------------------------------------------------------
5   Python class:    AdagradOptimizer
    File:    ../tensorflow/tensorflow/python/training/adagrad.py
    Line:    32
--------------------------------------------------------------------------------
```

Figure 2: FREQ top 5 most similar entities

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 TF IDF  top-5 most similar entities
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1   Python class:    AdadeltaOptimizer
    File:    ../tensorflow/tensorflow/python/training/adadelta.py
    Line:    29
--------------------------------------------------------------------------------
2   Python class:    Adadelta
    File:    ../tensorflow/tensorflow/python/keras/optimizer_v2/adadelta.py
    Line:    32
--------------------------------------------------------------------------------
3   Python class:    AdamOptimizer
    File:    ../tensorflow/tensorflow/python/training/adam.py
    Line:    32
--------------------------------------------------------------------------------
4   Python class:    AdagradOptimizer
    File:    ../tensorflow/tensorflow/python/training/adagrad.py
    Line:    32
--------------------------------------------------------------------------------
5   Python class:    Nadam
    File:    ../tensorflow/tensorflow/python/keras/optimizer_v2/nadam.py
    Line:    34
--------------------------------------------------------------------------------
```

Figure 3: TF IDF top 5 most similar entities

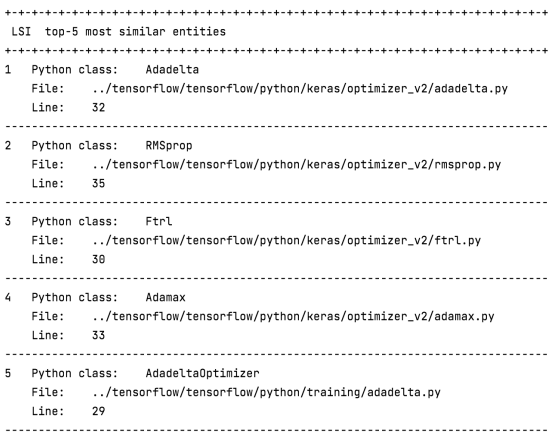

Figure 4: LSI top 5 most similar entities

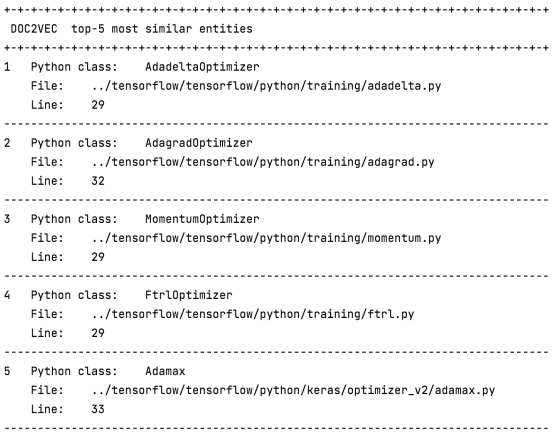

Figure 5: DOC2VEC top 5 most similar entities

Section 3: Evaluation of search engines

The third step is to evaluate the search engines. Since the ground truth is supplied as a `.txt` file containing 10 reference triplets, this file is firstly analysed to extract the queries, function/class name and file path. Next, all search engines, which had been trained in the previous step, are tested against the queries. Finally, precision and recall are evaluated for each search engine as follows:

- the ground truth entity is correctly reported by the search 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

The computed results can be seen in table 3. Analysing them, it turns out that they do not fully reflect the ranking of search engines from the previous section, as now LSI is in the first position followed by TF IDF, FREQ and DOC2VEC. However, it should be noted that these are the best results obtained, since during multiple runs they all remain identical except for LSI, which obtains on average the same results as TF IDF. In any case, we can say that all search engines do their job very well, having high recall and high average precision and thus being able to correctly identify most queries. LSI, TF IDF and FREQ have almost the same results, making them the best search engines as they can correctly identify 9 out of 10 queries and almost always place them in high positions on the top 5 ranking. DOC2VEC, on the other hand, performs worse, so it is worth investigating it further in the next section.

Engine	Avg Precision	Recall
LSI	0.87	1.00
TD-IDF	0.85	0.90
Frequencies	0.83	0.90
Doc2Vec	0.60	0.70

Table 3: Evaluation of search engines.

Section 4: Visualisation of query results

The fourth step is to display the result of the queries. In order to do so, I computed the **embedding vectors** of each query and the corresponding top 5 most similar entities for the LSI and DOC2VEC search engines.
Given the structural difference between the two, the embedding vectors are obtained in a different way: for LSI they are obtained by index from the LSI corpus, while for DOC2VEC they are obtained by calling `model.infer_vector` on vectors from the original corpus.
Finally, `t-SNE` (from package **sklearn**) was used to produce a 2D plot in which each group (vectors composed of each query and the associated top 5 answers) are plotted as points with the same colour. Figures 6 and 7 show the plots produced using the **seaborn** package on the 2D plot, where the classes relating to the ground truth queries are shown in the legend, with different colours.
The results reflect those obtained in the previous step as, at first glance, the graph obtained for LSI is better divided into clusters than the graph obtained for DOC2VEC. To better understand the graph, it is useful to point out that a **cluster** is represented by the query document and its top 5 most similar entities, thus it is reasonable to assume that a graph with dense and clearly separated clusters is the ultimate reference for a search engine that does its job well. Looking at DOC2VEC graph, we can see that the points are separated but do not always form dense clusters, in fact `custom_gradient` is very scattered, almost forming two different clusters, as are `get_loss_function` and `stride_slice`. This sparsity, which is more pronounced for `PastaAnalyzeVisitor`, reflects the facts documented in the previous sections, which is that DOC2VEC is the least performing search engine of the five analysed.
During further experiments, I also noticed that it happens, albeit rarely, that both graphs have outlier points that are extremely detached from all other points. I then carried out an in-depth analysis and discovered that the entities obtained as a result of the query "Gather gpu device info" are the cause.
This is caused by the fact that the relative ground truth file is: *../tensorflow/tensorflow/tools/test/gpu_info_lib.py*, i.e. it is a file in the **test** folder and since all entities containing the word test (and its variants) were deleted in the pre-training, the search engines probably run into a top 5 which is not very reliable, because the models have not been well trained to predict on test entities. In conclusion, it is possible to state that LSI performs better than DOC2VEC and also to confirm the ranking of the previous section.

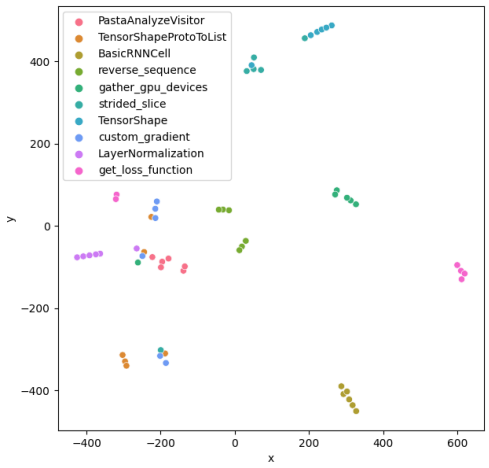

Figure 6: LSI projection of the embedding vectors of queries and top 5 answers

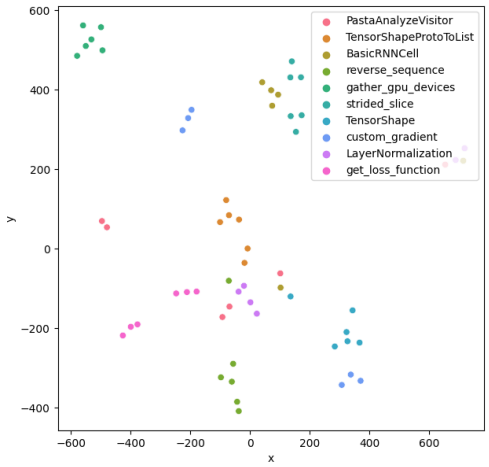

Figure 7: DOC2VEC projection of the embedding vectors of queries and top 5 answers