Knowledge Management and Analysis
Project 02: Code Search

Armend Azizi
https://github.com/armendazizi1/proj2-multi-source-code-search

Section 1 - Data Extraction

The goal of this part was to extract all names of top-level classes, functions, and class methods in *.py files for the **TensorFlow** library. The way I did that was using python's *os.walk* to traverse through the directories and all the python files. Afterward, for each python file I use **ast** to parse it and then use the parsed tree body to extract class, method, and function, definitions. For each node in the tree body, I check whether it is of type *ast.ClassDef*. If it is, I first check whether the class name is in the blacklist. If it also passes this condition, I add the class name and its comment. Afterward, I proceed to traverse its methods in *node.body* and check if it is of type *ast.FunctionDef*. If true, I then add the method name and its comment to the dataframe.

If the node in tree.body is not of type *ast.ClassDef*, I check whether it is of type function with *type(node) == ast.FunctionDef*. If true, I perform the same operations as above only that in this case I classify the node as type 'function' in the dataframe, which is then is saved into a CSV file.

Besides entity names, file path, line number, and type of entity, we were asked to also extract the comment line following the class, function, or method declaration. I did that using the *ast.get_ docstring(node)* command. After performing these steps and after saving the results into a CSV file, in Table 1 we can see the cumulative results of the findings.

Type	Number
Python files	2817
Classes	1883
Functions	4564
Methods	7526
Total entities	13973

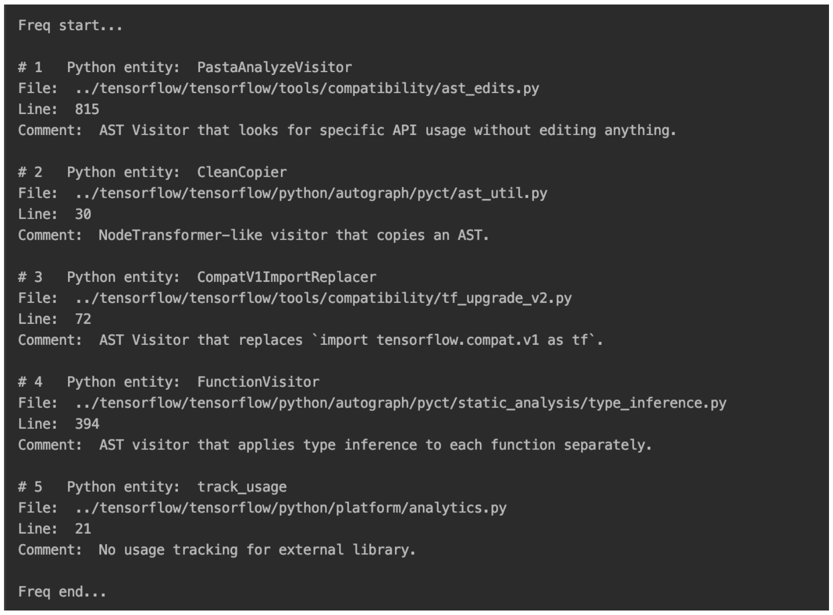
Table 1: Count of created classes and properties.

Section 2: Training of search engines

After extracting the data the next step was to use them to train the four search engines, *FREQ, TF-IDF, LSI, Doc2Vec* . We start by creating the corpus from the code entity names and comments, performing the following operations:

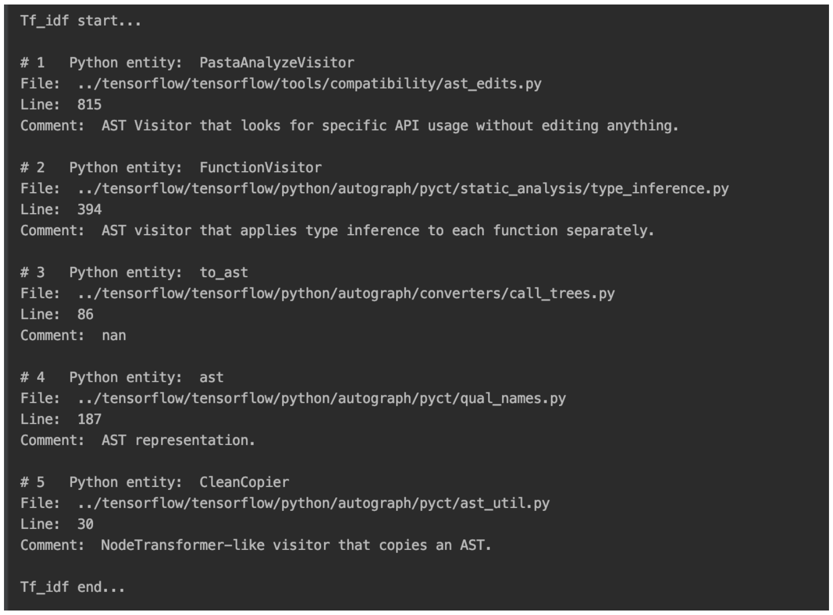
splitting entity names by camel-case and underscore, filtering out the stopwords, and last but not least converting all the words to lowercase.

Finally, we test them with a query (in our case the query is: *"AST Visitor that looks for specific API usage without editing anything"*) and print the top-5 most similar entities for each embedding.



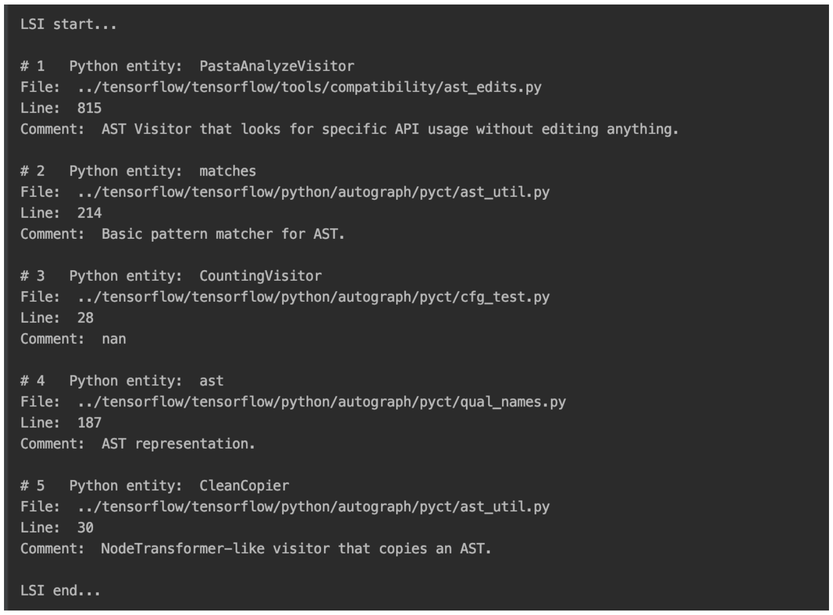
```
Freq start...

# 1   Python entity:  PastaAnalyzeVisitor
File:  ../tensorflow/tensorflow/tools/compatibility/ast_edits.py
Line:  815
Comment:  AST Visitor that looks for specific API usage without editing anything.

# 2   Python entity:  CleanCopier
File:  ../tensorflow/tensorflow/python/autograph/pyct/ast_util.py
Line:  30
Comment:  NodeTransformer-like visitor that copies an AST.

# 3   Python entity:  CompatV1ImportReplacer
File:  ../tensorflow/tensorflow/tools/compatibility/tf_upgrade_v2.py
Line:  72
Comment:  AST Visitor that replaces `import tensorflow.compat.v1 as tf`.

# 4   Python entity:  FunctionVisitor
File:  ../tensorflow/tensorflow/python/autograph/pyct/static_analysis/type_inference.py
Line:  394
Comment:  AST visitor that applies type inference to each function separately.

# 5   Python entity:  track_usage
File:  ../tensorflow/tensorflow/python/platform/analytics.py
Line:  21
Comment:  No usage tracking for external library.

Freq end...
```

Figure 1: top-5 most similar entities for FREQ



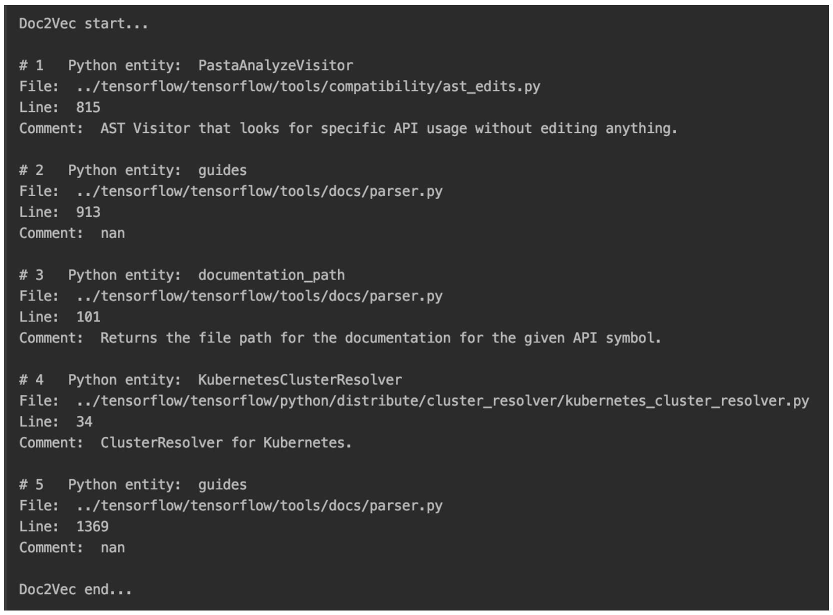
```
Tf_idf start...

# 1   Python entity:  PastaAnalyzeVisitor
File:  ../tensorflow/tensorflow/tools/compatibility/ast_edits.py
Line:  815
Comment:  AST Visitor that looks for specific API usage without editing anything.

# 2   Python entity:  FunctionVisitor
File:  ../tensorflow/tensorflow/python/autograph/pyct/static_analysis/type_inference.py
Line:  394
Comment:  AST visitor that applies type inference to each function separately.

# 3   Python entity:  to_ast
File:  ../tensorflow/tensorflow/python/autograph/converters/call_trees.py
Line:  86
Comment:  nan

# 4   Python entity:  ast
File:  ../tensorflow/tensorflow/python/autograph/pyct/qual_names.py
Line:  187
Comment:  AST representation.

# 5   Python entity:  CleanCopier
File:  ../tensorflow/tensorflow/python/autograph/pyct/ast_util.py
Line:  30
Comment:  NodeTransformer-like visitor that copies an AST.

Tf_idf end...
```

Figure 2: top-5 most similar entities for TF-IDF



```
LSI start...

# 1   Python entity:  PastaAnalyzeVisitor
File:  ../tensorflow/tensorflow/tools/compatibility/ast_edits.py
Line:  815
Comment:  AST Visitor that looks for specific API usage without editing anything.

# 2   Python entity:  matches
File:  ../tensorflow/tensorflow/python/autograph/pyct/ast_util.py
Line:  214
Comment:  Basic pattern matcher for AST.

# 3   Python entity:  CountingVisitor
File:  ../tensorflow/tensorflow/python/autograph/pyct/cfg_test.py
Line:  28
Comment:  nan

# 4   Python entity:  ast
File:  ../tensorflow/tensorflow/python/autograph/pyct/qual_names.py
Line:  187
Comment:  AST representation.

# 5   Python entity:  CleanCopier
File:  ../tensorflow/tensorflow/python/autograph/pyct/ast_util.py
Line:  30
Comment:  NodeTransformer-like visitor that copies an AST.

LSI end...
```

Figure 3: top-5 most similar entities for LSI



```
Doc2Vec start...

# 1   Python entity:  PastaAnalyzeVisitor
File:  ../tensorflow/tensorflow/tools/compatibility/ast_edits.py
Line:  815
Comment:  AST Visitor that looks for specific API usage without editing anything.

# 2   Python entity:  guides
File:  ../tensorflow/tensorflow/tools/docs/parser.py
Line:  913
Comment:  nan

# 3   Python entity:  documentation_path
File:  ../tensorflow/tensorflow/tools/docs/parser.py
Line:  101
Comment:  Returns the file path for the documentation for the given API symbol.

# 4   Python entity:  KubernetesClusterResolver
File:  ../tensorflow/tensorflow/python/distribute/cluster_resolver/kubernetes_cluster_resolver.py
Line:  34
Comment:  ClusterResolver for Kubernetes.

# 5   Python entity:  guides
File:  ../tensorflow/tensorflow/tools/docs/parser.py
Line:  1369
Comment:  nan

Doc2Vec end...
```

Figure 4: top-5 most similar entities for Doc2vec

Before commenting the results, for the query *"AST Visitor that looks for specific API usage without editing anything"* the ground truth states that the function for that query should be **PastaAnalyzeVisitor** from the python file **../tensorflow/tensorflow/tools/compatibility/ast_edits.py**.

If we look at the results, we can see that each of the search engines correctly classified the most similar entity. If we look at the other 4 entities in the top-5 we see somewhat similar results in the first 3 search engines. For example in Figure 1 we see functions such as *CleanCopier* and *FunctionVisitor* which also appear for *TF-IDF* and/or *LSI* in Figure 2 and Figure 3. The comments of these functions talk about "AST visitor..." so I would say they are pretty accurate.

In the top-5 most similar entities for *DOC2VEC* we don't see such good results, even though it got the top most similar entity correctly, the other 4 have very little aspects in common with the query.

Section 3: Evaluation of search engines

The third part of the project was to use a ground truth provided, evaluate and report recall and average precision for each of the four search engines, and comment on the differences among search engines. The calculation of precision and recall is done as follows:

- 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.

Table 2 shows the recall and average precision for each of the four search engines.

Engine	Avg Precision	Recall
Frequencies	0.85	0.9
TF-IDF	0.8	0.8
LSI	0.95	1.0
Doc2Vec	0.65	0.8

Table 2: Evaluation of search engines.

From the table, we can see that *LSI* performed the best for the 10 queries in the ground truth. It correctly classified all of them in its top five documents and with the highest precision among the other search engines. The second best is *FREQ* by having 9/10 times the correct entity most similar to the query (according to the ground truth) being in the top five most similar entities. Also by average precision *FREQ* is the second best with **0.85**, followed then by *TF-IDF* with **0.8** average precision. *TF-IDF* and *Doc2Vec* share the same recall, both with **0.8**, but *Doc2Vec* has a worse performance in average precision with **0.65**.

Section 4: Visualisation of query results

Before commenting plots, as we can see each color represents the correct function that we get from the ground truth for 10 questions, and the mapping (function-query) is as follows:

1. PastaAnalyzeVisitor
   - AST Visitor that looks for specific API usage without editing anything
2. TensorShapeProtoToList
   - Convert a TensorShape to a list
3. BasicRNNCell
   - The most basic RNN cell
4. reverse_sequence
   - Reverses variable length slices
5. gather_gpu_devices
   - Gather gpu device info

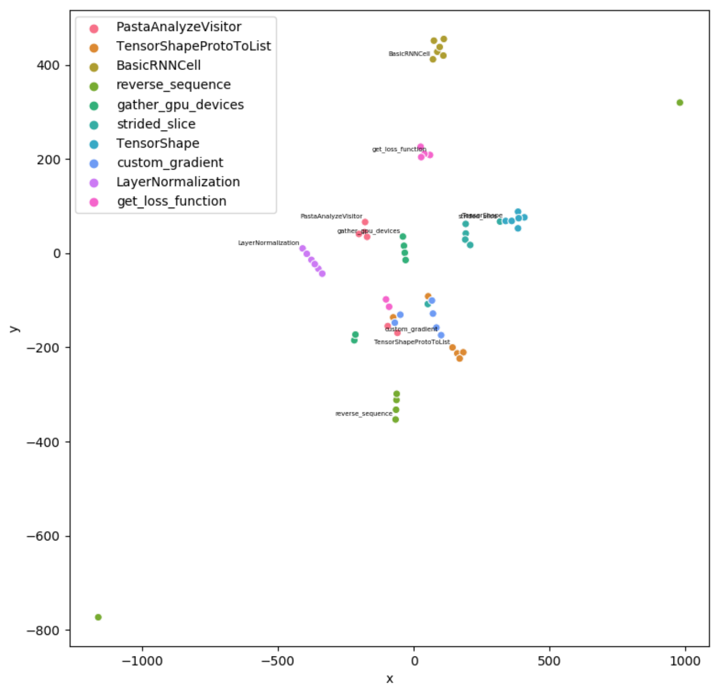


Figure 5: t-SNE plot for LSI

6. strided_slice
   - Extracts a strided slice of a tensor generalized python array indexing

7. TensorShape
   - Represents the shape of a Tensor

8. custom_gradient
   - Decorator to define a function with a custom gradient

9. LayerNormalization
   - Layer normalization layer

10. get_loss_function
   - loss function

Looking at the two plots we can say that both did a relatively good job of clustering the top five most similar entities for a query. For example if we look

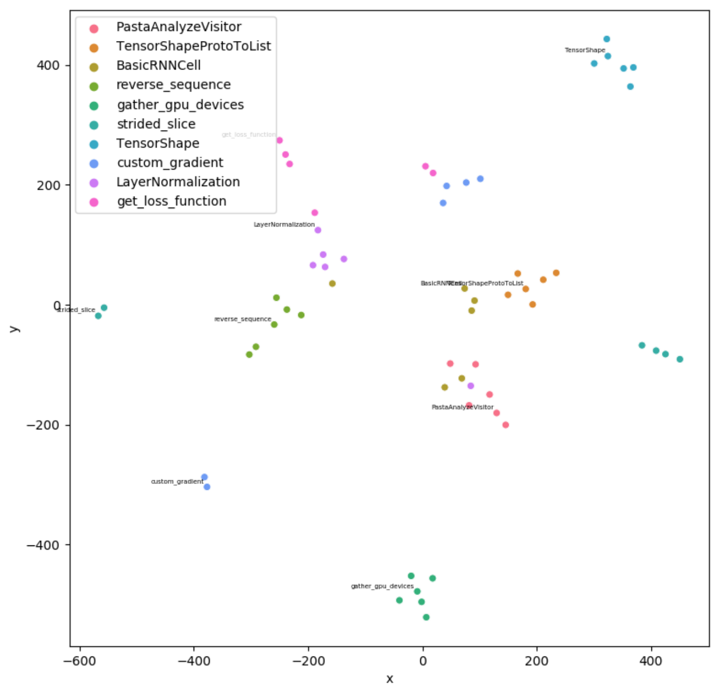


Figure 6: t-SNE plot for Doc2Vec

at Figure 5 for the query *"Layer normalization layer"* we see all 6 (1 query + 5 top five most similar entities to it) documents clustered together. I checked that manually and it is actually the case.

Similarly, if we look at the t-SNE plot for *Doc2Vec* (Figure 6) for the same query *"Layer normalization layer"* we see a similar result where all the 6 documents are clustered together. I also checked that manually and the results all contained the word "layer" either in the name or in its comment. In this plot though, we see a trend that we also saw when we trained the search engines and outputted the top 5 most similar entities Figure 4. The trend is that the topmost similar entity is similar to the entity, but the other remaining four are not. In the plot we can see that for cluster *"strided_ slice"* and *"custom_ gradient"*, where we see the query and the top entity on a different cluster, and the other remaining four on a different cluster. I have seen multiple plots for different runs and sometimes for doc2Vec we see clusters with single elements which is only the query. This makes sense since also in the recall only 7/10 queries had successful results in the top 5 most similar entities found.

To conclude, from the precision and recall results, I expected *LSI* to have a better clustering than *DOC2VEC*, and this was somehow the case here since not only are the 5 entities better clustered with the query, but we also see a smaller radius of the clusters in *LSI*.