Knowledge Management and Analysis
Project 01: Code Search

Jury Andrea D'Onofrio

November 2023

Section 1 - Data Extraction

The primary goal was to identify the number of Python files in each directory. To do so, I visit each folder to find out the number of Python files. Subsequently, I explored into each Python file to extract valuable insights. For each Python file, I counted how many classes and functions it contains, taking in account the specific black list. Therefore, classes and functions with names beginning with underscores '_', 'main', or 'test' were omitted from the counts.
Moreover, for each legal class, I extended the examination to each of its child, if any. In particular, I visited each child to compute the number of methods filtering those starting with '_' or 'main' or 'test' using the same black list I used before. The results of the analysis are presented in Table 1.

Type	Number
Python files	2817
Classes	1883
Functions	4580
Methods	5418

Table 1: Count of created classes and properties.

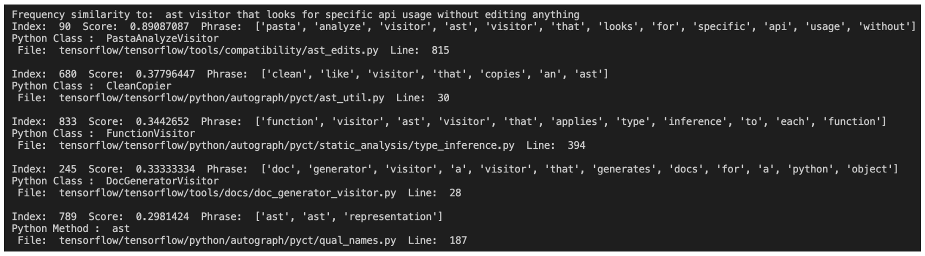
Section 2: Training of search engines

I started processing my data frame on the name and comment columns. Firstly, I removed alphanumeric symbols from the comments list and I split entity names by camel-case and underscore. Then, I joined the two lists into a new one, removing the stopwords and utilizing only the first 13 words to get better results during my models' evaluation. Finally, not to lose the matching with the ground truth for the evaluation part, I substituted the empty strings with "INVALID STRING" and remove all ":".

As the second step, I created a dictionary {'key': 'value'} where key is a word of the previous processed corpus and value is the occurrence of the word, then I kept only the entries with value > 1.

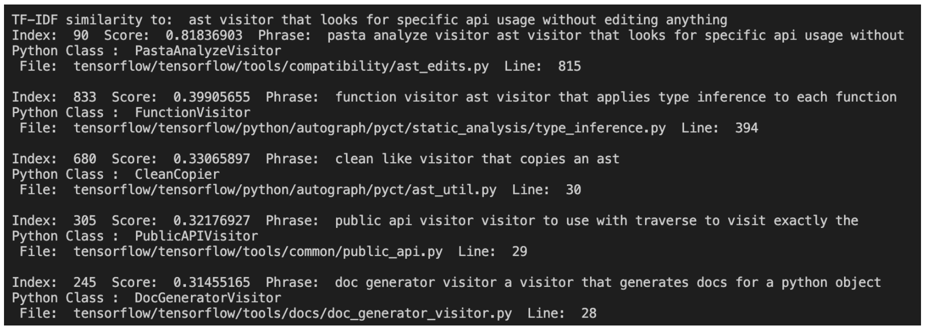
As the last step, I filtered the processed corpus removing the words that are not in the dictionary created in the previous step.

In the following images, you can see an example of my results returned by each model.



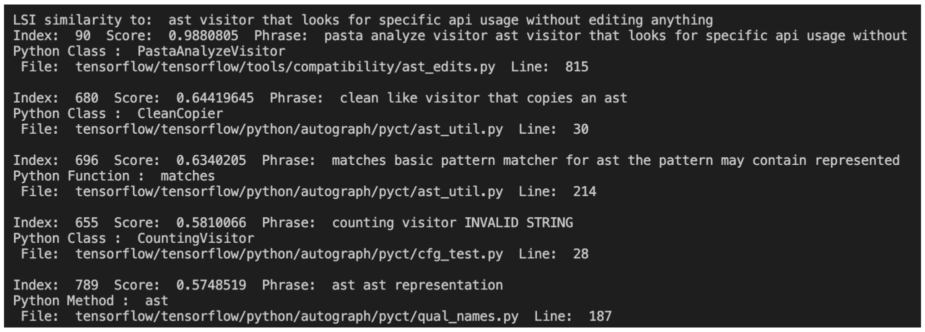
```
Frequency similarity to:  ast visitor that looks for specific api usage without editing anything
Index:  90  Score:  0.89087087  Phrase:  ['pasta', 'analyze', 'visitor', 'ast', 'visitor', 'that', 'looks', 'for', 'specific', 'api', 'usage', 'without']
Python Class :  PastaAnalyzeVisitor
 File:  tensorflow/tensorflow/tools/compatibility/ast_edits.py  Line:  815

Index:  680  Score:  0.37796447  Phrase:  ['clean', 'like', 'visitor', 'that', 'copies', 'an', 'ast']
Python Class :  CleanCopier
 File:  tensorflow/tensorflow/python/autograph/pyct/ast_util.py  Line:  30

Index:  833  Score:  0.3442652  Phrase:  ['function', 'visitor', 'ast', 'visitor', 'that', 'applies', 'type', 'inference', 'to', 'each', 'function']
Python Class :  FunctionVisitor
 File:  tensorflow/tensorflow/python/autograph/pyct/static_analysis/type_inference.py  Line:  394

Index:  245  Score:  0.33333334  Phrase:  ['doc', 'generator', 'visitor', 'a', 'visitor', 'that', 'generates', 'docs', 'for', 'a', 'python', 'object']
Python Class :  DocGeneratorVisitor
 File:  tensorflow/tensorflow/tools/docs/doc_generator_visitor.py  Line:  28

Index:  789  Score:  0.2981424  Phrase:  ['ast', 'ast', 'representation']
Python Method :  ast
 File:  tensorflow/tensorflow/python/autograph/pyct/qual_names.py  Line:  187
```

Figure 1: Frequency similarity.



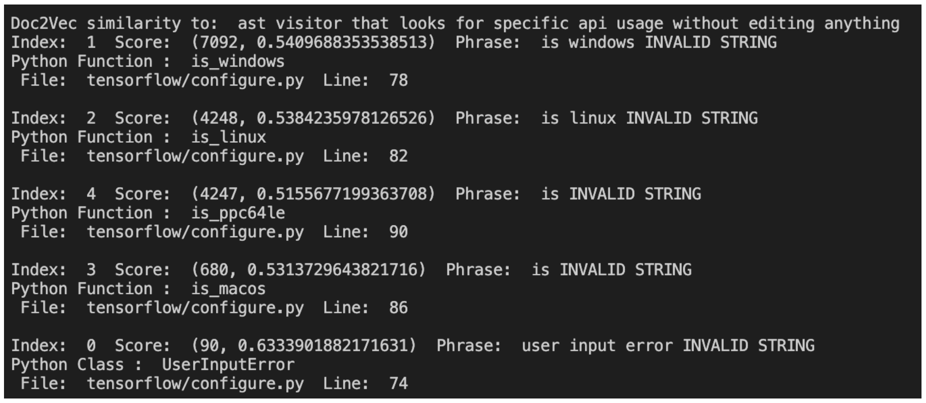
```
TF-IDF similarity to:  ast visitor that looks for specific api usage without editing anything
Index:  90  Score:  0.81836903  Phrase:  pasta analyze visitor ast visitor that looks for specific api usage without
Python Class :  PastaAnalyzeVisitor
 File:  tensorflow/tensorflow/tools/compatibility/ast_edits.py  Line:  815

Index:  833  Score:  0.39905655  Phrase:  function visitor ast visitor that applies type inference to each function
Python Class :  FunctionVisitor
 File:  tensorflow/tensorflow/python/autograph/pyct/static_analysis/type_inference.py  Line:  394

Index:  680  Score:  0.33065897  Phrase:  clean like visitor that copies an ast
Python Class :  CleanCopier
 File:  tensorflow/tensorflow/python/autograph/pyct/ast_util.py  Line:  30

Index:  305  Score:  0.32176927  Phrase:  public api visitor visitor to use with traverse to visit exactly the
Python Class :  PublicAPIVisitor
 File:  tensorflow/tensorflow/tools/common/public_api.py  Line:  29

Index:  245  Score:  0.31455165  Phrase:  doc generator visitor a visitor that generates docs for a python object
Python Class :  DocGeneratorVisitor
 File:  tensorflow/tensorflow/tools/docs/doc_generator_visitor.py  Line:  28
```

Figure 2: TF-IDF similarity.



```
LSI similarity to:  ast visitor that looks for specific api usage without editing anything
Index:  90  Score:  0.9880805  Phrase:  pasta analyze visitor ast visitor that looks for specific api usage without
Python Class :  PastaAnalyzeVisitor
 File:  tensorflow/tensorflow/tools/compatibility/ast_edits.py  Line:  815

Index:  680  Score:  0.64419645  Phrase:  clean like visitor that copies an ast
Python Class :  CleanCopier
 File:  tensorflow/tensorflow/python/autograph/pyct/ast_util.py  Line:  30

Index:  696  Score:  0.6340205  Phrase:  matches basic pattern matcher for ast the pattern may contain represented
Python Function :  matches
 File:  tensorflow/tensorflow/python/autograph/pyct/ast_util.py  Line:  214

Index:  655  Score:  0.5810066  Phrase:  counting visitor INVALID STRING
Python Class :  CountingVisitor
 File:  tensorflow/tensorflow/python/autograph/pyct/cfg_test.py  Line:  28

Index:  789  Score:  0.5748519  Phrase:  ast ast representation
Python Method :  ast
 File:  tensorflow/tensorflow/python/autograph/pyct/qual_names.py  Line:  187
```

Figure 3: LSI similarity.



```
Doc2Vec similarity to:  ast visitor that looks for specific api usage without editing anything
Index:  1  Score:  (7092, 0.5409688353538513)  Phrase:  is windows INVALID STRING
Python Function :  is_windows
 File:  tensorflow/configure.py  Line:  78

Index:  2  Score:  (4248, 0.5384235978126526)  Phrase:  is linux INVALID STRING
Python Function :  is_linux
 File:  tensorflow/configure.py  Line:  82

Index:  4  Score:  (4247, 0.5155677199363708)  Phrase:  is INVALID STRING
Python Function :  is_ppc64le
 File:  tensorflow/configure.py  Line:  90

Index:  3  Score:  (680, 0.5313729643821716)  Phrase:  is INVALID STRING
Python Function :  is_macos
 File:  tensorflow/configure.py  Line:  86

Index:  0  Score:  (90, 0.6333901882171631)  Phrase:  user input error INVALID STRING
Python Class :  UserInputError
 File:  tensorflow/configure.py  Line:  74
```

Figure 4: Doc2Vec similarity.

Section 3: Evaluation of search engines

As evident from the results presented in Table 2, my evaluation of search engines has yielded insightful findings. LSI, TD-IDF, and Frequencies have demonstrated strong performance, with LSI outperforming as the top-performing method, achieving a 95% average precision.

In contrast, Doc2Vec exhibited the lowest average precision among the tested models, scoring a value of 67%. Despite its lower precision, Doc2Vec has a recall of 0.8, meaning that it successfully identifies 8 out of 10 queries.

Moreover, unlike LSI, TF-IDF, and Frequency models, which use straightforward methods, Doc2Vec employs a neural network, therefore it is harder to handle. The obtained results may be attributed to several factors, including the possibility of overfitting, incorrect hyperparameter settings, and the absence of a validation set. It is essential to consider these elements as potential contributors to the observed outcomes. These complexities make Doc2Vec computationally demanding, especially when compared to the simpler approaches of LSI, TF-IDF, and Frequency-based models.

For the LSI model, I set the `num_topics` parameter to 300; this value was suggested in the pack of slides 'PROJ-02-multi-search.pdf'. This parameter is related to the dimensionality of the latent space, which defines the number of concepts that the model aims to discover within the data.

In the case of the Doc2Vec model, I made the following parameter choices:

- `vector_size = 300` as defined in the 'PROJ-02-multi-search.pdf' slide deck. This parameter, as before, set the dimensionality of the latent space in the Doc2Vec model;

- `min_count = 2` since it maximized the quality of my results. This parameter specifies that words with a total frequency less than 2 are ignored during training;
- `epochs = 500` as suggested from the teaching assistant. This parameter defines how many times the model iterates over the entire dataset during training.

It's also noteworthy that LSI, TD-IDF, and Frequencies achieved a recall score of 1.0, indicating that they were able to retrieve all 10 queries during the evaluation on the ground-truth.

Engine	Avg Precision	Recall
Frequencies	93%	1.0
TD-IDF	88%	1.0
LSI	95%	1.0
Doc2Vec	67%	0.8

Table 2: Evaluation of search engines.

Section 4: Visualisation of query results

Figure 5 provides the t-SNE plot illustrating the distribution of data points for LSI, while Figure 6 shows the t-SNE plot for the Doc2Vec model.

As you can see in Figure 5, the LSI model yields remarkably consistent results, with each data point close to the others within its respective class. This high level of clustering is also reflected in the results presented in Table 2, where the LSI model achieves an average precision value of 95%.

As for the Doc2Vec model, as Figure 6 shows, it still works well. However, how the plot illustrates, the results are not so uniform as well as LSI. This observation is confirmed also with the findings in Table 2 where its average precision is 67%. Albeit the low value, it's important to note that this score still boosts random performance.

Note that for both the visualization, I used the recommended parameters `perplexity = 2` and `n_iter = 3000`. Moreover, since I gave priority to a better visualization, sometimes some data points are out of bounds of the figure.

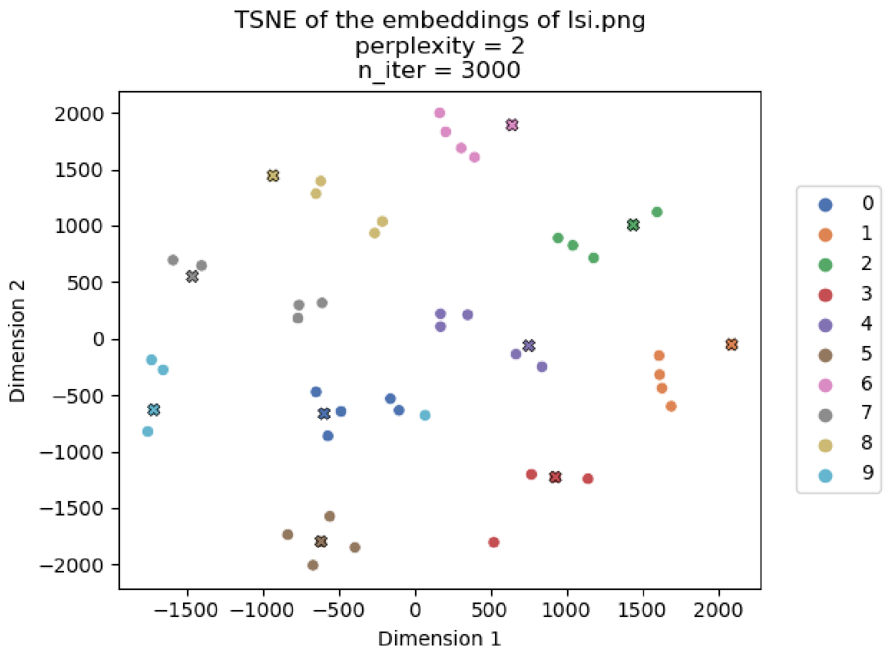

TSNE of the embeddings of lsi.png
perplexity = 2
n_iter = 3000
Dimension 1
Dimension 2
2000
1500
1000
500
0
−500
−1000
−1500
−2000
−1500
−1000
−500
0
500
1000
1500
2000
0
1
2
3
4
5
6
7
8
9

Figure 5: LSI

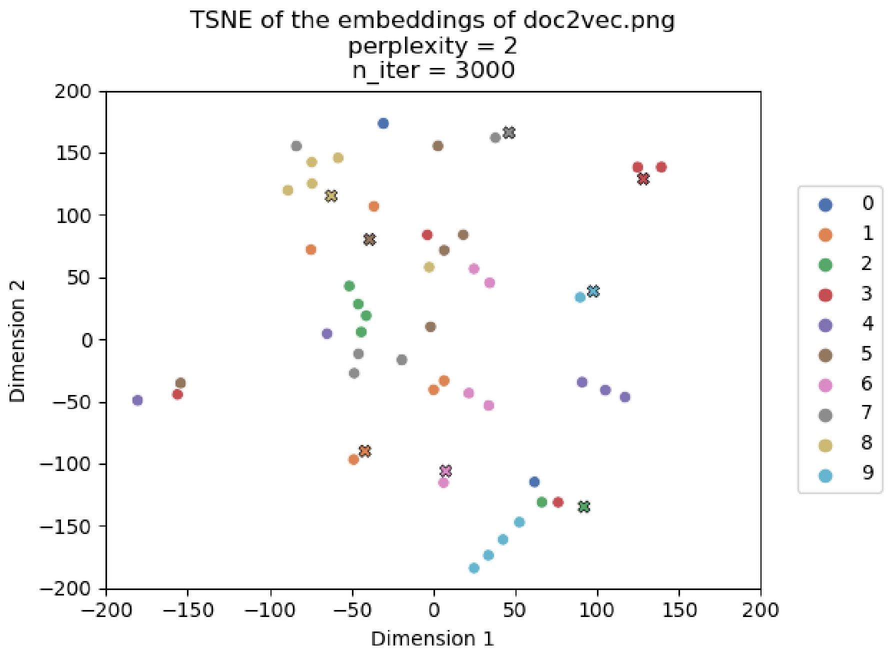


Figure 6: Doc2Vec

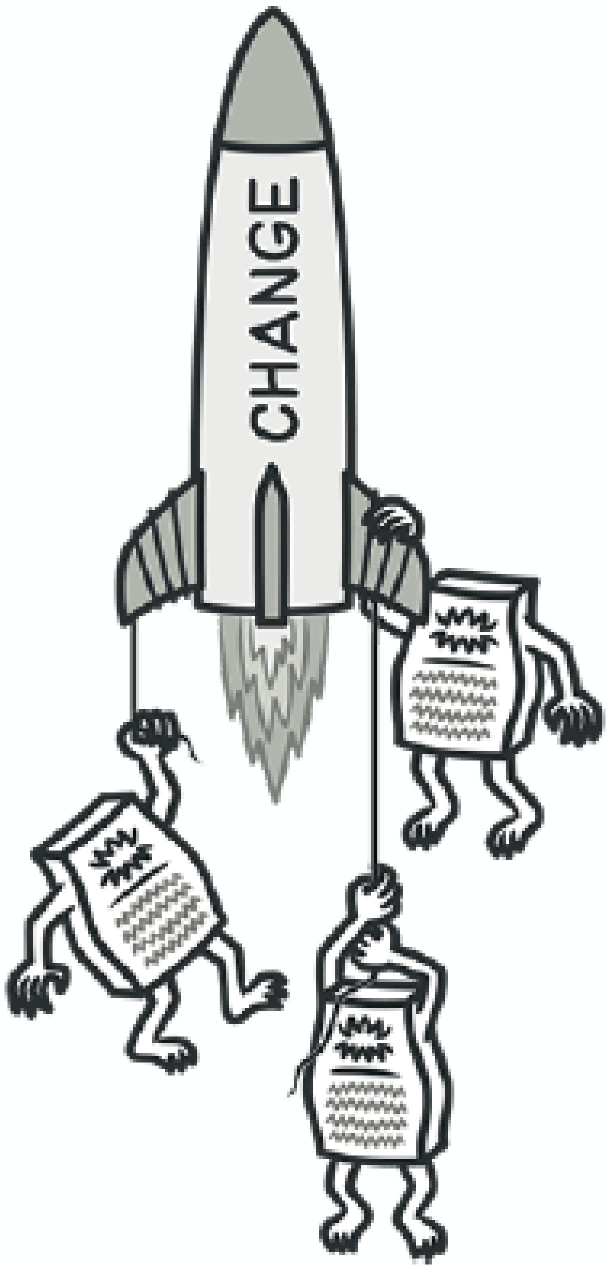


Figure 7: Caption