Knowledge Search and Extraction
Project 01: Code Search

Navdeep Singh Bedi

Section 1 - Data Extraction

In Part 01 of the project, we crawled into the 'tensorflow' directory using 'os.walk' and then selected all the Python files (files with '.py' extension). Then I created the AST (Abstract Syntax Tree) using the 'ast.NodeVisitor' method of the 'AST' module Python in the 'Visitor' class. Then we one by one select all the classes, methods, and functions which are not part of the blacklist, which is defined by the list of functions and the classes names following the below-mentioned rules:
- 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.

Similarly, we extract the line number, comment (if available), and path corresponding to each class and function and put them in a dictionary. Finally, we convert the data extracted in the dictionary into a Pandas data frame as shown in table 1. The code is in the '**extract-data.py**' file in the GitHub repository. We can run this code and enter the name of the directory 'tensorflow' in our case and get the data frame. The number of Python files, classes, methods, and, functions are given in the table 1.

Type	Number
Python files	1885
Classes	1883
Functions	4580
Methods	7060

Table 1: Count of created classes and properties.

Section 2: Training of search engines

In Part 02 we created the corpus of 'name' and 'comment' corresponding to each class and method. To do this, we first split the names by camel-case and underscore (e.g., go_to_myHome -> [go, to, my, home]). Then, we also removed stop words from the name and comments, to make the search engine more efficient. The list of stopwords that we removed is: ['test', 'tests', 'main', 'this', 'is', 'in', 'to']. Next, we concatenated the list of the names and the comments corresponding to each class and method and converted all of them to lowercase

letters. Then, we create a frequency dictionary of the words presented in the filtered concatenated list of names and comments corresponding to each class and method and also the dictionary assigning the token IDs to each word. Then this dictionary is converted into the corpus for 'Bag of Words' (FREQ). Then this corpus is further transformed into the corpus for 'TF-IDF' and then into the corpus for 'LSI'. Further, the similarity matrix was created for the FREQ, LSI, and TF-IDF, and a random query from the user was also given to the different models, and the top 5 matching results in the data frame for the query for each model were selected. Next, the top 5 queries were found for the Doc2Vec model and the results from each of the models were reported and the Doc2Vec model was saved. For example for the,
Query: 'Optimizer that implements the Adadelta algorithm'
the results for different models are:
TF-IDF
AdadeltaOptimizer
Algorithm
Adadelta
Adadelta
GradientDescentOptimizer
LSI
RMSPropOptimizer
AdadeltaOptimizer
AdagradOptimizer
FtrlOptimizer
GradientDescentOptimizer
FREQ
AdadeltaOptimizer
GradientDescentOptimizer
RMSPropOptimizer
Algorithm
AdagradOptimizer
Doc2Vec
applyOptimizer
applyOptimizer
bias_add
applyOptimizer
increment_variable_v1
The repetitions in the results of the different models are the classes or functions in the different files. The code is in the '**search-data.py**' file in the GitHub repository. We can run this code and enter the name of the file name 'data.csv' in our case and the query as 'Optimizer that implements the Adadelta algorithm' in our case for the search engines.

Section 3: Evaluation of search engines

In Part 03 of the project we calculate the average precision and recall for the search engines created using different models. To do this we use the 'ground-truth-unique.txt' data provided to us, which contains the query, ground truth function/class name, and file location. For each of the 10 queries, we calculate the top 5 most probable answers using all four search engine models. Next, we calculate the precision for each query for each model and then find the average precision for each model. Next, we calculate the recall and report the answers in the table 2.

Engine	Avg Precision	Recall
Frequencies	0.28	0.4
TD-IDF	0.25	0.3
LSI	**0.425**	**0.5**
Doc2Vec	0.1	0.1

Table 2: Evaluation of search engines.

It is seen that the LSI model gives the best result whereas the Doc2Vec has the worst. The code is in the '**prec-recall.py**' file in the GitHub repository. We can run this code and enter the name of the file 'data.csv' in our case and the ground truth file 'ground-truth-unique.txt' in our case.

Section 4: Visualisation of query results

The t-SNE plots for LSI and Doc2Vec are given in fig.1 and fig.2 respectively. The 'q1', 'q2',...., and 'q10' represent the ten queries. These results agree with the results of the table 2, in which the LSI has good precision and recall as the LSI is able to better search the results given the query compared to Doc2Vec.

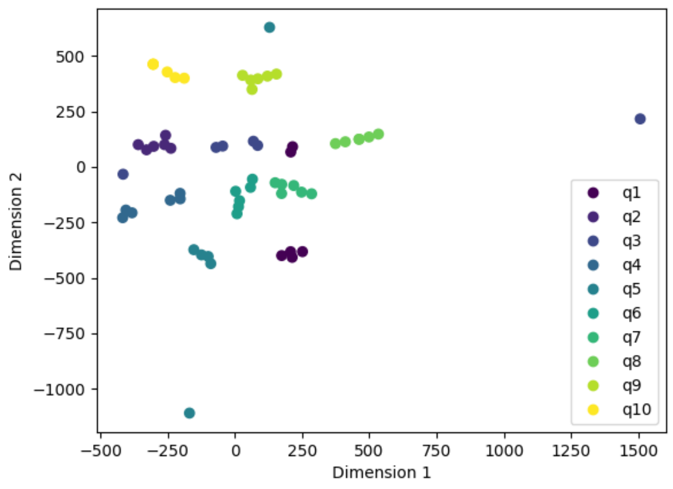

Figure 1: t-SNE plot for LSI.

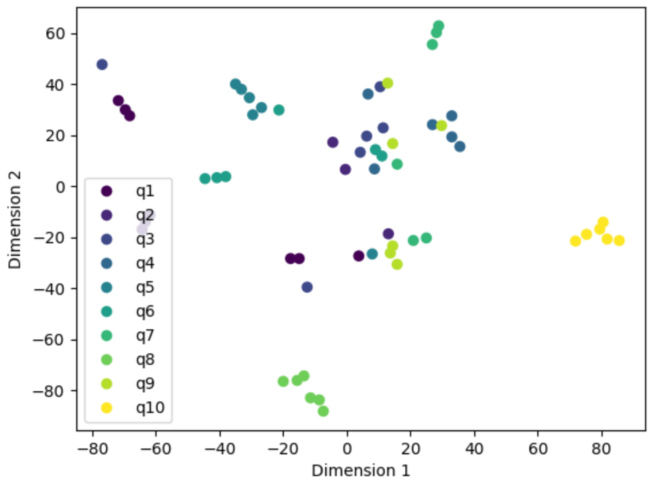

Figure 2: t-SNE plot for Doc2Vec.