Università della Svizzera italiana

Faculty of Informatics

Knowledge Analysis & Management 2019/2020

Assignment Report

Project: Multi-language code search

Simone Giacomelli

In this project, we want to develop a search engine that can query a large multi-language (Python, C++) code repository. The subject of the indexing is TensorFlow, an open-source framework for machine learning and deep learning.

Table of contents

Extract data

We want to extract all names of classes, functions and methods in *.cc and *.py files of the TensorFlow Github repository. The name of the program described in this section is ‘extract-data.py’ and it will produce an output file called data.csv containing the information of the extracted entity.

Structure

This phase of extraction will require to walk the folder of TensorFlow looking for python (.py) and C++ files (.cc).

For the python files I will employ the embedded AST library for parsing; for C++ files CLANG with binding is used.

This program has two main functions:

- extract_cc()
- extract_py()

Both use python os.walk(...) API to retrieve the list of all files that match the corresponding file mask.

Extraction of cc entities

The library CLANG needs us to crate an Index instance, parse the C++ source file and the result of this operation will be an AST:

```
index = clang.cindex.Index.create()
tree = index.parse(file)
```

The AST is now contained in the ‘tree’ variable; the following step see the invocation of a function with the root node of the tree:

```
find_typerefs(tree.cursor)
```

Such ‘find_typerefs(node)’ function is recursive and will invoke itself for all the child nodes of node.

In every invocation, it will look for interesting entities (function, class or method) and it will append a line to the CSV file. The full source code can be found in the appendix.

Extraction of py entities

The extraction of the information from the python file is accomplished using the internal ‘ast’ module.

Similarly to the CLANG library, ‘ast’ modules gives us the means to inspect the tree of the python source code:

```
tree = ast.parse(source.read())
```

And then we can loop through the nodes of the AST:

```
 for node in ast.walk(tree):
```

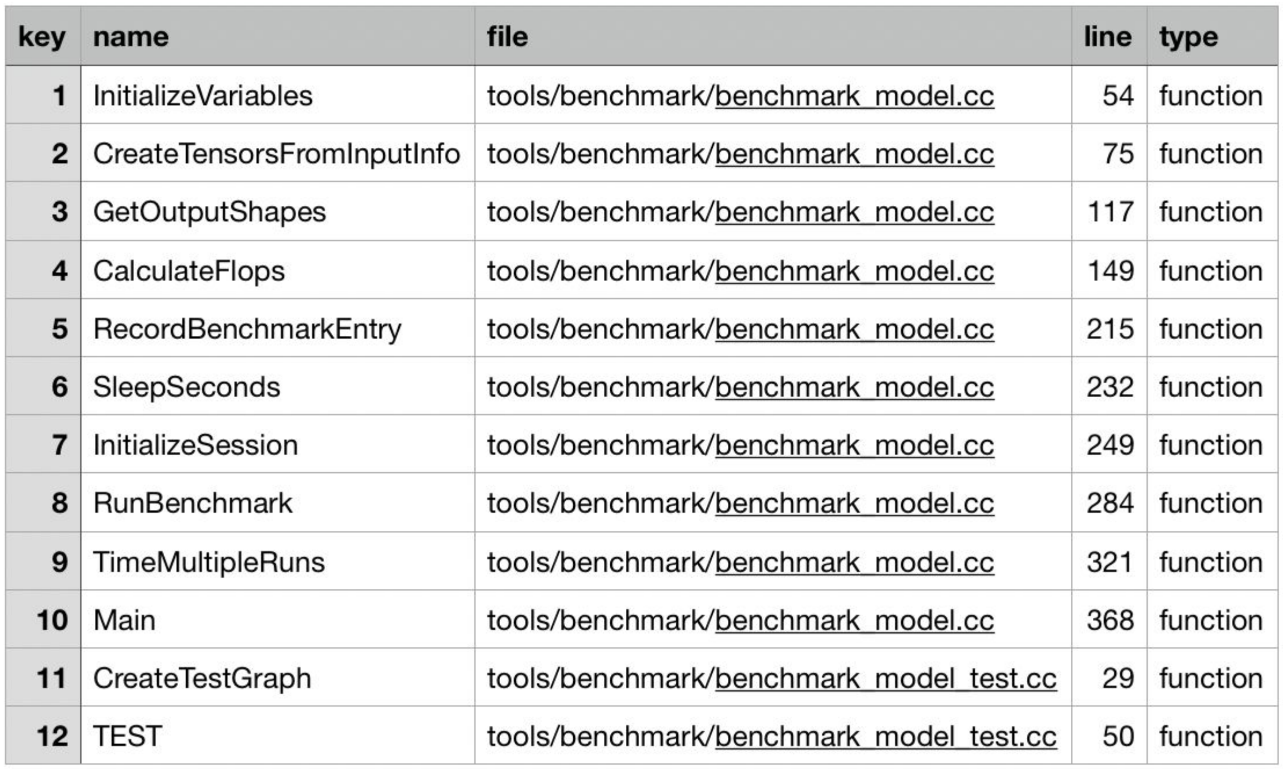
The rest of this function looks for the entity we want and write the information to the CSV file.

Csv file

The CSV file has five columns:

- key, it is just a progressive counter
- name of the entity
- file that contains the entity
- line number of the file
- type of the entity

Figure 1 contains a few lines of the produced csv file.



key	name	file	line	type
1	InitializeVariables	tools/benchmark/benchmark_model.cc	54	function
2	CreateTensorsFromInputInfo	tools/benchmark/benchmark_model.cc	75	function
3	GetOutputShapes	tools/benchmark/benchmark_model.cc	117	function
4	CalculateFlops	tools/benchmark/benchmark_model.cc	149	function
5	RecordBenchmarkEntry	tools/benchmark/benchmark_model.cc	215	function
6	SleepSeconds	tools/benchmark/benchmark_model.cc	232	function
7	InitializeSession	tools/benchmark/benchmark_model.cc	249	function
8	RunBenchmark	tools/benchmark/benchmark_model.cc	284	function
9	TimeMultipleRuns	tools/benchmark/benchmark_model.cc	321	function
10	Main	tools/benchmark/benchmark_model.cc	368	function
11	CreateTestGraph	tools/benchmark/benchmark_model_test.cc	29	function
12	TEST	tools/benchmark/benchmark_model_test.cc	50	function

Figure 1. an excerpt of data.csv

Comments on extracted data

The total number of files parsed is 5867. They are divided in:

- Python files: 2279
- C++ files: 3588

The number of entities is 50373; not all of them will be searchable. This will be explained in the proper section.

The discovered entities have an attribute that specifies the type, these are the distribution:

- class type: 6840
- function type: 40089
- method type: 3444
- *total: 50373

Training of search engines

This section is about the training of four different search engines:

- FREQ: frequency-based
- TF-IDF: Term frequency-inverse document frequency based
- LSI: Latent semantic indexing based
- Doc2Vec: doc2vec based

Training code description

This code is almost all contained in a file called ‘query.py’. This is a module created because it will be entirely reused in the following project steps.

The file ‘search-data.py’ is almost empty; in-fact it contains just the invocation of the ‘query’ module. For example, to print a simple query:

```
q = query.Query("Optimizer Adadelta")
q.print()
```

The following section will describe how the ‘query’ module works.

Source code of query.py

This file defines the stop words; I did not actually use the recommended stop words because I did some additional processing that superseded the original stopwords.

After many runs and observations of this program output, I noticed that many entities are related to automatic test or verification code. I opted to remove any entity that contains the words:

- test
- tests

This is why I decided to keep only one stopword: **main**. More on this later in this section.

Duplicates removal

Many entities are found multiple times in all the TensorFlow repository; for example, the functions ‘build_graphs’ and ‘build_inputs’ are defined roughly one hundred times each (in different files).

There are multiple approaches to handle this; because it was not explicitly stated how, I decided to just remove the duplicates. This is the code that removes the duplicates:

```
all_documents_dict = {d[1]: d for d in [process_line(line) for line in lines]}
documents_no_dup = list(all_documents_dict.values())
```

Entity name processing

In order to extract the tokens from the entity name I used two different techniques:

- Regular expression for camel case (‘MutableHashTable’)
- Simple string split when it was detected the underscore (‘find_scalar_and_max_depth’)

Entity token filtering rules

After extracting the tokens from the entity names I proceeded to apply a filter on the token array; these are the rules employed:

1. removed the tokens with string length less or equal than two (discard if len(tokens) <= 2)
2. removed tokens with global frequency of only one

The first did yield better performance but the second point dropped the accuracy and recall; this is why I skipped the removal of tokens with frequency one.

Entity removal

In some cases, after the entity token filtering, the token vector was empty; for example, the entity:

```
‘is_it_ok’
```

generates the token vector:

```
[is,it,ok]
```

After the application of rule 1, the vector is *empty*. In such cases I dropped the entity completed; i.e. the entity ‘is_it_ok’ will not be part of the corpus for indexing.

I added another heuristic to remove entirely an entity: if any token matches ‘test’ or ‘tests’ the entity is not added to the corpus.

Training

The training is quite straightforward. It is easy to create the following objects:

- the dictionary used by all the documents: `corpora.Dictionary(texts)`
- the bag of words for every document: `dictionary.doc2bow(text)`

The following steps deal with instantiating the search engine models:

- The FREQ model just uses the bag of words
- LSI and TF-IDF use the bag of words and are created respectively with
    - `models.LsiModel(...)`
    - `models.TfidfModel(...)`

  Both of these uses `similarities.MatrixSimilarity(...)` to change into the correct vector space

- The creation of the doc2vec model is more involved; it does not use a bag of words but it requires access to the tokens of the entities. The model object is created with the library gensim using this constructor: `gensim.models.doc2vec.Doc2Vec()`

Class Query

The class Query has a constructor that requires a query string. This string is preprocessed to be compatible with the corpus (split, lowercase).

Using the models instantiated in the previous section, the query tokens are translated into the vector space for each search engine.

For FREQ, TF-IDF and LSI we calculate the similarity score for every document.
Then we get the top five hits; see source code `Query.top5()`.

The doc2vec search engine requires these two calls:

- `vector = doc2vec_model.infer_vector(query_words)` will transform the query words into the doc2vec vector space
- `doc2vec_model.docvecs.most_similar([vector], topn=5)` will give back the top five most similar documents

Notice that a Query class instance, in the end, will hold:

- the string tokens of the query string
- the corresponding bag of words
- one vector for each algorithm containing the result of the query (four vectors in total)
    - Every result vector is composed of five tuples (document_index, similarity_score) that represent the top-5 most similar entities

This information will be handy later when we will be needed to compute the embedding vectors of the query and the top-5 hits.

Example of query

The following query

```
q = query.Query("A generic hash table that is immutable once initialized")
q.print()
```

yield this result:

```
FREQ

#1 Entity name: HashTable
   File: python/ops/lookup_ops.py
   Line: 367
   Type: class

#2 Entity name: getHashTable
   File: python/kernel_tests/lookup_ops_test.py
   Line: 56
   Type: function

#3 Entity name: StaticHashTable
   File: python/ops/lookup_ops.py
   Line: 245
   Type: class
```

I just reported the first hits for the FREQ algorithm for the sake of brevity. The full output shows the top five hits for every algorithm.

Evaluation of search engines

Source code description

The file ‘prec-recall.py’ makes use of ‘query.py’; what it additionally does is to read and parse the file ‘ground-truth.txt’.

This txt file is organized in triplets with the following data:

- query string
- entity name
- file

The parsing of the txt file is handled by a class named GroundTruthRow. This class is quite trivial because it solves the following simple tasks:

- verify that the entity specified in the ground truth really exists; otherwise, it throws an exception
- creates an instance of Query with the string query of the ground truth
- calls query.execute() and for each of the four results create an instance of the class Measure

The **Measure** class constructor requires:

- the result of a search query
- the expected correct document index

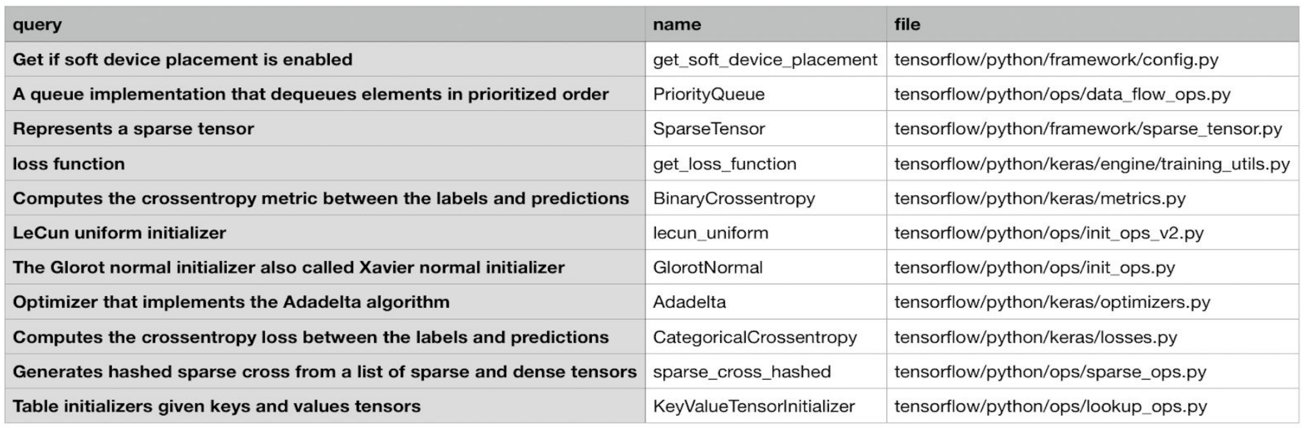
With these data, it proceeds to store in its fields the following information:

- field ‘prec’: the precision calculated as by requirements
- field ‘found’: it will be 1 if the correct answer is present in the hits, 0 otherwise. This field will be used later to calculate the recall

Finally, there is a loop that calculates the recall and prints out the results.

Ground truth used for the evaluation

I gathered eleven queries from the TensorFlow website. The full list is in figure 2.



query	name	file
Get if soft device placement is enabled	get_soft_device_placement	tensorflow/python/framework/config.py
A queue implementation that dequeues elements in prioritized order	PriorityQueue	tensorflow/python/ops/data_flow_ops.py
Represents a sparse tensor	SparseTensor	tensorflow/python/framework/sparse_tensor.py
loss function	get_loss_function	tensorflow/python/keras/engine/training_utils.py
Computes the crossentropy metric between the labels and predictions	BinaryCrossentropy	tensorflow/python/keras/metrics.py
LeCun uniform initializer	lecun_uniform	tensorflow/python/ops/init_ops_v2.py
The Glorot normal initializer also called Xavier normal initializer	GlorotNormal	tensorflow/python/ops/init_ops.py
Optimizer that implements the Adadelta algorithm	Adadelta	tensorflow/python/keras/optimizers.py
Computes the crossentropy loss between the labels and predictions	CategoricalCrossentropy	tensorflow/python/keras/losses.py
Generates hashed sparse cross from a list of sparse and dense tensors	sparse_cross_hashed	tensorflow/python/ops/sparse_ops.py
Table initializers given keys and values tensors	KeyValueTensorInitializer	tensorflow/python/ops/lookup_ops.py

Figure 2. full list of ground-truth.txt

On the website I could not find description for the C++ api, so I could not mix queries for both languages. This is why my ground truth contains python files only.

Recall and average precision

The recall and average precision computed with the ground truth from the previous section are the following:

```
FREQ:
```

```
Average precision = 0.2
Recall = 0.6

LSI:
Average precision = 0.1
Recall = 0.3

TF-IDF:
Average precision = 0.1
Recall = 0.6

DOC2VEC:
Average precision = 0.1
Recall = 0.2
```

The best two algorithms are FREQ and TF_IDF. They actually have almost the same measured performances. Even if they are the most simple algorithms of the group they perform the best.

This can be explained because the body that is indexed is very small: it comprises only the tokens inside the entity name; so a simpler algorithm performs better.

This is the reason why the two more complex algorithms (LSI and doc2vec) do not show their potential: there is not enough text to work on.

Visualization of query results

Code for visualization

I added three functions at the bottom of ‘prec-recall.py’ python file:

- calc_tsne_lsi()
- calc_tsne_doc2vec()
- plot_tsne(filename, all_results, hue, size)

Both the first two functions compute the numbers and the plot_tsne() will create the plot:

```
plot_tsne('plot-doc2vec.png', *calc_tsne_doc2vec())
plot_tsne('plot-lsi.png', *calc_tsne_lsi())
```

Both `calc_tsne_lsi()` and `calc_tsne_doc2vec()` have similar structure:

- loop through all the ground truth queries; for each ground truth:
    - compute the embedding vector of the query in the model space
    - retrieve the top-5 most similar entities indexes and for each of them, compute the embedding vector

After the loop, each function collects the following vectors:

- `embeddings`. length 66, because I chose 11 queries:
    - 11 embedding vectors for the queries
    - 55 embedding vectors for the results
- `hue`. length 66, it contains the information about the color for each data point.
- `size`. length 66, it contains the information about the size for each data point. I computed this vector in order to paint the query data point bigger than the points for the top-5 most similar documents

The function `plot_tsne()`writes a png image with the collected data.

Comments on plots

This section is about showing and commenting on the t-SNE plots for both LSI and doc2vec.

The t-SNE algorithm is not deterministic; I discovered that running multiple times the algorithm the plots can be very much different from each other (even with the same exact parameters).

I used the parameter settings as advised in the slides. I tried other values but the best plots are produced with the following suggested parameters:

- perplexity = 2
- n_iter = 3000

The plots have markers of two different sizes:

- the query data point projection is represented with a bigger circle
- the related top-5 most similar documents are represented with a smaller circle

Plot for LSI

The t-SNE plot for LSI has six queries with high result cohesion:

- get_soft_device_placement
- GlorotNormal
- AdaDelta
- SparseTensor
- PriorityQueue
- lecun_uniform

The representation of these queries and related documents are very close in the 2D projection (see Figure 3).

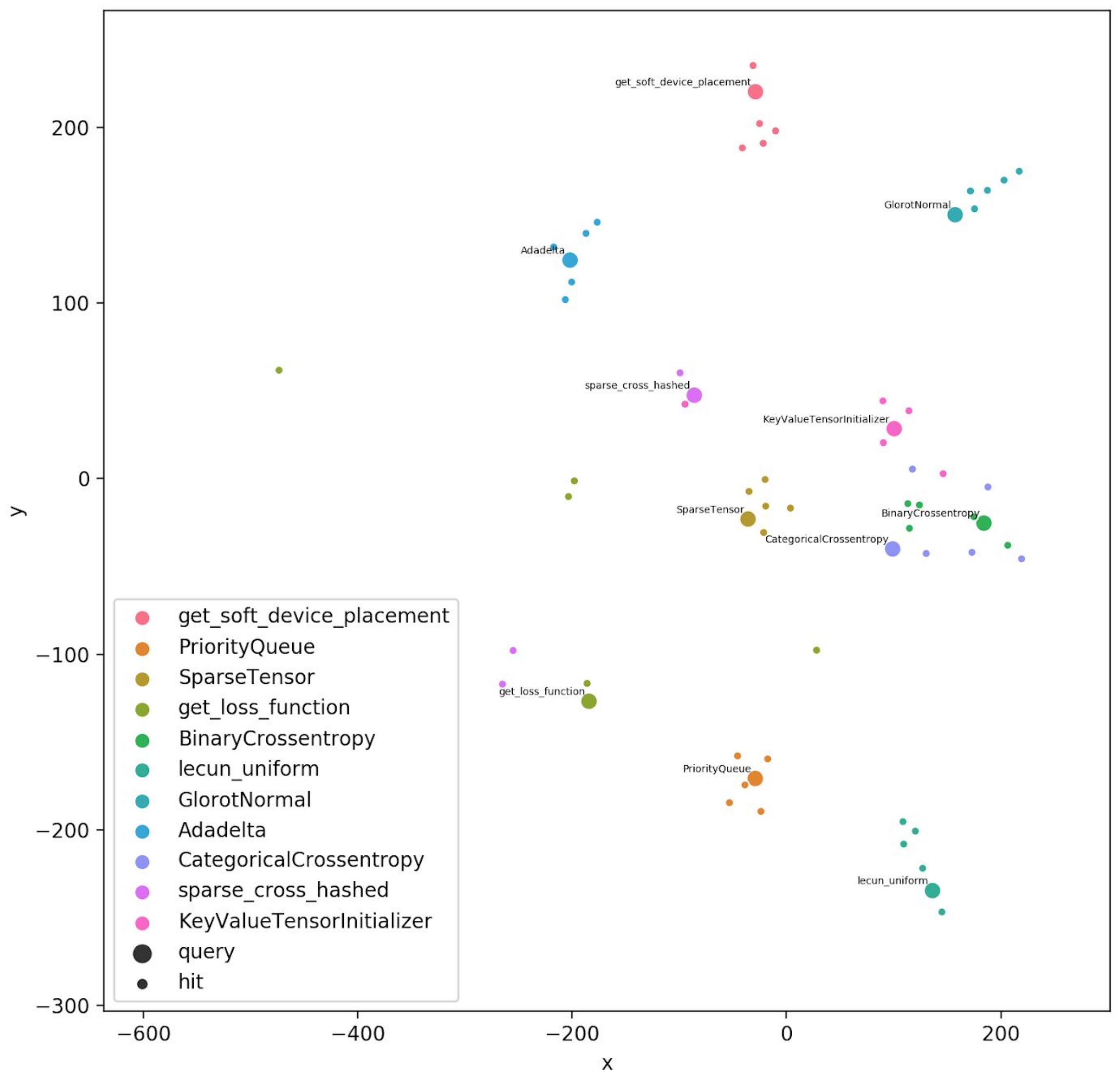


Figure 3. t-SNE plot for LSI (with zoom)

There is one interesting case that was not previously planned; the geometric distribution of:

- BinaryCrossentropy
- CategoricalCrossentropy

These two entities share 50% of the tokens. This is clearly shown in the plot; the two queries are very close to each other and the ten results are intertwined in the area - roughly in coordinates (200,0).

Not all queries have cohesive results. Some of them, get_loss_function for example, have very far result points.

Infact Figure 4 is a representation of the full dataset points (66 points in total) shown in Figure 3.

Notice Figure 4 has two very distant results points:

- one is located in the upper left corner
- one is located in the lower right corner

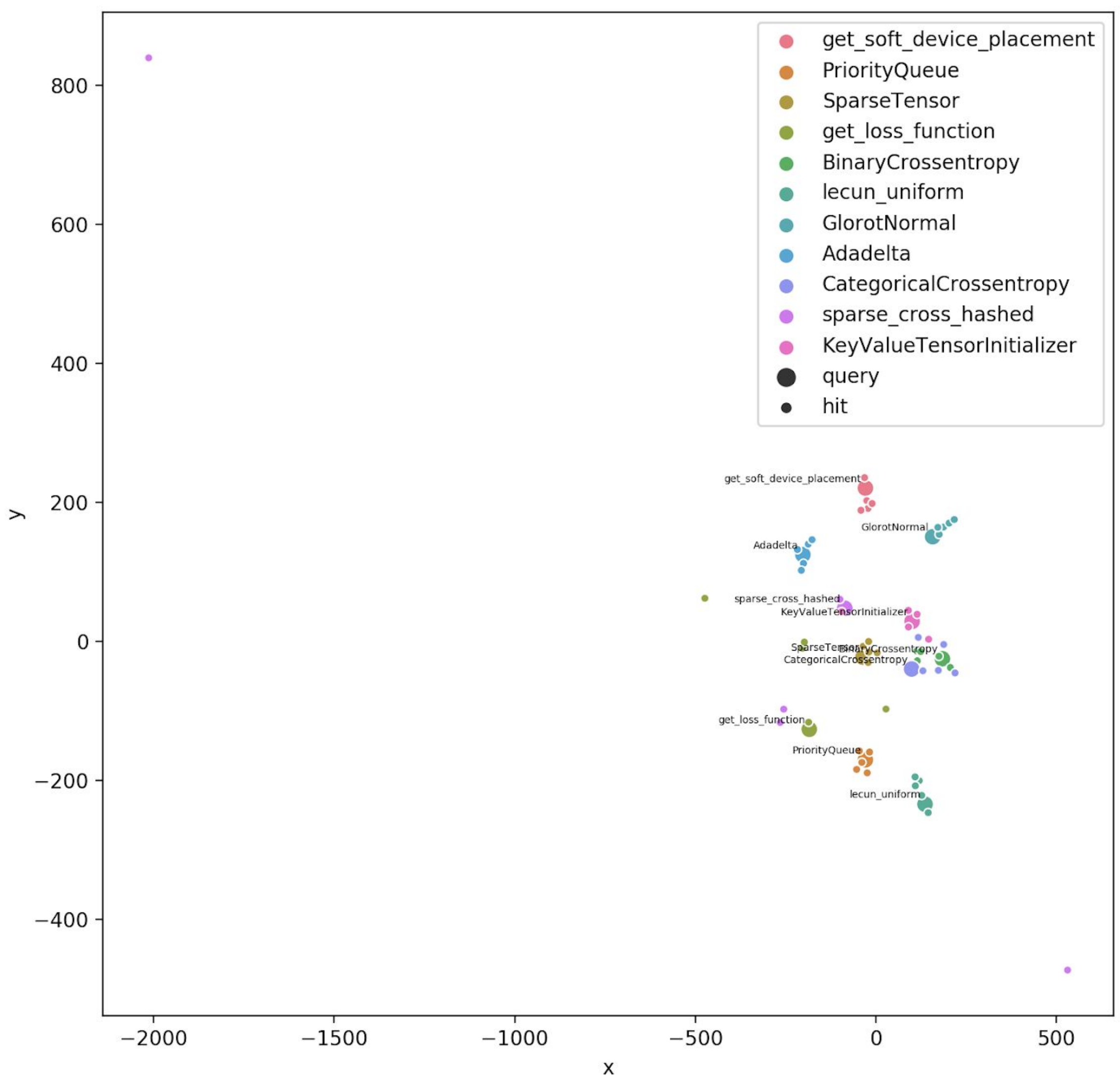


Figure 4. t-SNE plot for LSI (no zoom)

Plot for doc2vec

The t-SNE plot for doc2vec has six queries with high result cohesion:

- get_soft_device_placement
- get_loss_function
- GlorotNormal
- SparseTensor

- Adadelta
- KeyValueTensorInitializer

The query data points for CategoricalCrossentropy and BinaryCrossentropy are very close; on the other end, as we can see in Figure 5, the results data points are not very close.

Also in doc2vec plot we have two outliers; they are visible in Figure 6 where the plot is without zoom.

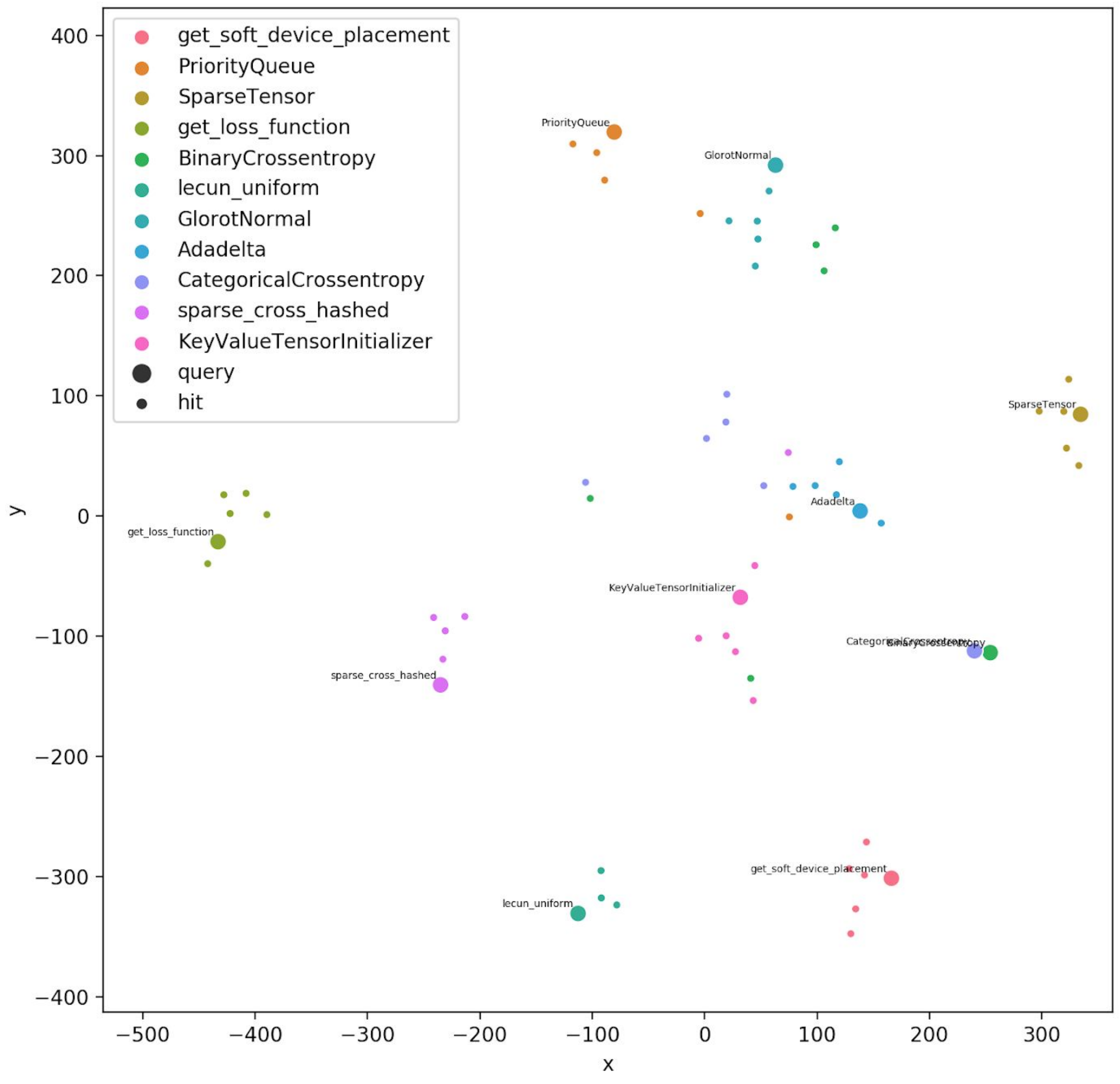


Figure 5. t-SNE plot for LSI (with zoom)

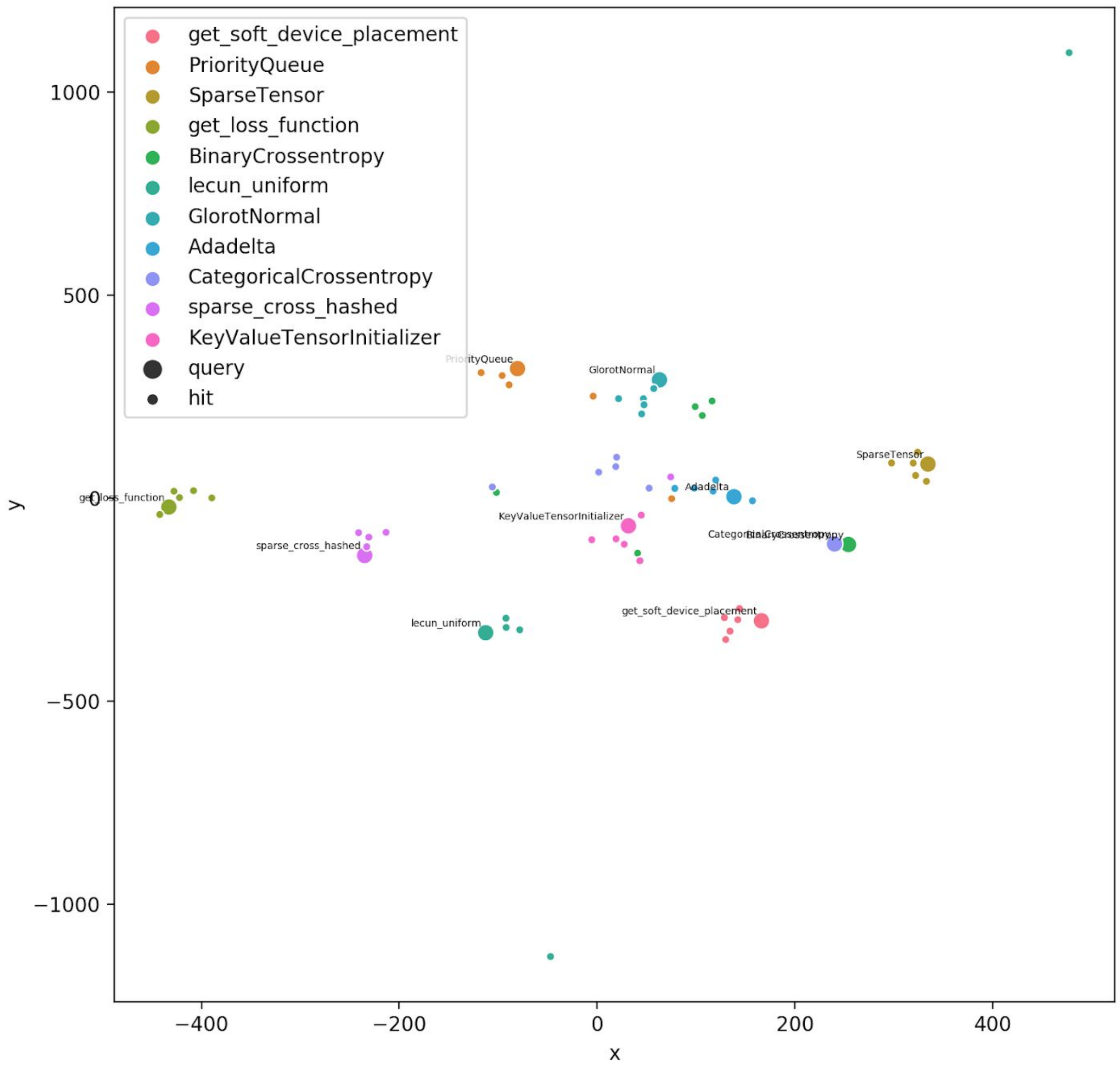


Figure 6. t-SNE plot for LSI (no zoom)

Appendix: Python code

----

extract-data.py

```
import ast
import os
from glob import glob

import clang.cindex

clang.cindex.Config.set_library_path('./clang+llvm-9.0.0/lib')


class Csv:
    def __init__(self, filename):
        self.f = open(filename, 'w')
        self.idx = 0

    def append(self, line):
        self.idx += 1
        self.f.write(f'{self.idx},' + line)
        self.f.write('\n')
        self.f.flush()

    def __del__(self):
        self.f.close()


csv = Csv('data.csv')
source_folder_path = "tensorflow/tensorflow"
if not os.path.isdir(source_folder_path):
    print('Folder ', source_folder_path, 'not found')
    exit(0)


def extract_cc():
    source_files_path = [y for x in os.walk(source_folder_path) for y in glob(os.path.join(x[0],
'*.cc'))]

    for file in source_files_path:

        filestr = file[len(source_folder_path) + 1:]

        def find_typerefs(node):
            try:
                kind = node.kind
                if kind == clang.cindex.CursorKind.FUNCTION_DECL:
                    csv.append(f'{node.spelling},{filestr},{node.extent.start.line},function')
                elif node.kind == clang.cindex.CursorKind.CLASS_DECL:
                    csv.append(f'{node.spelling},{filestr},{node.extent.start.line},class')
                elif node.kind == clang.cindex.CursorKind.CXX_METHOD:
                    csv.append(f'{node.spelling},{filestr},{node.extent.start.line},method')
            except:
                pass

            for c in node.get_children():
                find_typerefs(c)
        index = clang.cindex.Index.create()
        tree = index.parse(file)
        print(f'VISITING {filestr}')

        find_typerefs(tree.cursor)


def extract_py():
    source_files_path = [y for x in os.walk(source_folder_path) for y in glob(os.path.join(x[0],
'*.py'))]

    for file in source_files_path:
        filestr = file[len(source_folder_path) + 1:]

        with open(file, "r") as source:
            print(f'VISITING {filestr}')
            tree = ast.parse(source.read())
            for node in ast.walk(tree):
                if isinstance(node, ast.ClassDef):
                    csv.append(f'{node.name},{filestr},{node.lineno},class')
                elif isinstance(node, ast.FunctionDef):
                    csv.append(f'{node.name},{filestr},{node.lineno},function')


if __name__ == '__main__':
    extract_cc()
    extract_py()
```

----

extract-data.py

```
import query


def main():
    q = query.Query("A generic hash table that is immutable once initialized")
    q.print()


if __name__ == '__main__':
    main()
```

----

query.py

```
from collections import defaultdict
from re import finditer

import gensim
from gensim import corpora
from gensim import models

# remove common words and tokenize
stoplist = set('main'.split())
with open('data.csv', 'r') as f:
    lines = f.read().splitlines(keepends=False)


def camel_case_split(identifier):
    matches = finditer('.+?(?:(?<=[a-z])(?=[A-Z])|(?<=[A-Z])(?=[A-Z][a-z])|$)', identifier)
    return [m.group(0) for m in matches]
```

```python
def split_entity(id):
    if '_' in id:
        parts = id.split('_')
    else:
        parts = camel_case_split(id)
    parts = [p.lower() for p in parts if p.strip() != '']
    return parts


def process_line(line):
    fields = line.split(',')[1:]
    name = fields[0]
    parts = split_entity(name)
    parts = [p for p in parts if p not in stoplist]
    fields.insert(0, parts)
    return fields

# process names
all_documents_dict = {d[1]: d for d in [process_line(line) for line in lines]}

documents_no_dup = list(all_documents_dict.values())

frequency = defaultdict(int)

for doc in documents_no_dup:
    for token in doc[0]:
        frequency[token] += 1


def remove_infrequent(doc):
    # remove token that has frequency 1
    # tokens = [token for token in doc[0] if frequency[token] > 1]
    # doc[0] = tokens

    l = len([t for t in doc[0] if t == 'test' or t == 'tests'])
    if l > 0:
        doc[0] = []
    tokens = [token for token in doc[0] if len(token) > 2]
    doc[0] = tokens
    return doc


# and exclude documents that does not has tokens
documents = [remove_infrequent(doc) for doc in documents_no_dup]
documents = [d for d in documents if len(d[0]) > 0]

texts = [d[0] for d in documents]

dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]

from gensim import similarities

# freq
freq_matrix_sim = similarities.MatrixSimilarity(corpus)

# lsi
lsi_model = models.LsiModel(corpus, id2word=dictionary, num_topics=200)
lsi_martrix_sim = similarities.MatrixSimilarity(lsi_model[corpus])

# tfidf
tfidf_model = models.TfidfModel(corpus, id2word=dictionary)
tfidf_matrix_sim = similarities.MatrixSimilarity(tfidf_model[corpus])

# doc2vec
def read_corpus():
    for i, line in enumerate(texts):
        yield gensim.models.doc2vec.TaggedDocument(line, [i])
```

```python
doc2vec_train_corpus = list(read_corpus())
doc2vec_model = gensim.models.doc2vec.Doc2Vec(vector_size=50, min_count=0, epochs=60)

doc2vec_model.build_vocab(doc2vec_train_corpus)

doc2vec_model.train(doc2vec_train_corpus, total_examples=doc2vec_model.corpus_count,
epochs=doc2vec_model.epochs)
engines = ['FREQ', 'LSI', 'TF-IDF', 'DOC2VEC']


class Query:
    def __init__(self, query):
        query_string = query.lower()
        self.query_words = query_string.lower().split()
        self.query_bow = dictionary.doc2bow(self.query_words)
        self.res_freq = []
        self.res_lsi = []
        self.res_tfidf = []
        self.res_doc2vec = []

    def results(self):
        return [self.res_freq, self.res_lsi, self.res_tfidf, self.res_doc2vec]

    def execute(self):
        self.freq_query_docs()
        self.tf_idf_query_docs()
        self.lsi_query_docs()
        self.doc2vec_query_docs()
        return self.results()

    def print(self):
        self.execute()
        self.print_results('FREQ', self.res_freq)
        self.print_results('LSI', self.res_lsi)
        self.print_results('TF-IDF', self.res_tfidf)
        self.print_results('doc2vec', self.res_doc2vec)

    def freq_query_docs(self):
        sims = freq_matrix_sim[self.query_bow]  # perform a similarity query against the corpus
        self.res_freq = self.top5(sims)

    def lsi_query_docs(self):
        lsi_query_vec = lsi_model[self.query_bow]  # convert the query to LSI space
        sims = lsi_martrix_sim[lsi_query_vec]  # perform a similarity query against the corpus
        self.res_lsi = self.top5(sims)

    def tf_idf_query_docs(self):
        query_vec = tfidf_model[self.query_bow]  # convert the query to LSI space
        sims = tfidf_matrix_sim[query_vec]  # perform a similarity query against the corpus
        self.res_tfidf = self.top5(sims)

    def doc2vec_query_docs(self):
        vector = doc2vec_model.infer_vector(self.query_words)
        self.res_doc2vec = doc2vec_model.docvecs.most_similar([vector], topn=5)

    def top5(self, sims):
        res = sorted(enumerate(sims), key=lambda item: -item[1])[:5]
        return res

    def print_results(self, note, sims):
        print()
        print(note)
        for i, tup in enumerate(sims):
            pos, sim = tup
            doc = documents[pos]
            print()
            print(f'#{i + 1} Entity name: {doc[1]}')
            print(f'   File: {doc[2]}')
            print(f'   Line: {doc[3]}')
            print(f'   Type: {doc[4]}')
```

----

prec-recall.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.manifold import TSNE

import query

docmap = dict()
for idx, doc in enumerate(query.documents):
    if (doc[1] == 'Adadelta'):
        print(doc)
    docmap[(doc[1])] = idx
LSI_IDX = 1


class Measure:
    def __init__(self, q, results, correct_idx):
        self.q = q
        repr = ''
        pos = 0
        for i, tup in enumerate(results):
            repr += f',{query.documents[tup[0]]}'
            if correct_idx == tup[0]:
                pos = i + 1

        if pos != 0:
            self.found = 1
            self.prec = pos / len(results)
            st = f'in position {pos}'
        else:
            self.found = 0
            self.prec = 0
            st = 'not found'

        print('  doc', st, repr)
        self.pos = pos


class GroundTruthRow:
    def __init__(self, line):
        p = line.split(',')
        self.query = query.Query(p[0])
        self.entity = p[1]
        self.file = p[2]
        self.correct_idx = docmap.get((self.entity), -1)
        if self.correct_idx == -1:
            raise Exception(f'Ground truth not found in documents. Offending line {p}')

        print(query.documents[self.correct_idx], "query: " + p[0])
        # execute returns 4 items, one for each engine
        self.results = [Measure(p[0], r, self.correct_idx) for r in self.query.execute()]

        print('precision')


with open('ground-truth.txt', 'r') as f:
    gt_lines = [l for l in f.read().split('\n') if not l.startswith('#')]
```

```
lsi = None
gts = [GroundTruthRow(line) for line in gt_lines]
for engine in range(len(query.engines)):

    results = [g.results[engine] for g in gts]

    if engine == 1:
        lsi = results

    prec = np.average([g.prec for g in results])
    hit = np.sum([g.found for g in results])
    recall = hit / len(gts)
    print()
    print(query.engines[engine] + ':')
    print(f'Average precision = {np.round(prec, 1)}')
    print(f'Recall = {np.round(recall, 1)}')


def calc_tsne_lsi():
    embeddings = []
    hue = []
    size = []
    for i, gt in enumerate(gts):
        bows = [query.corpus[hit[0]] for hit in gt.query.res_lsi] + [gt.query.query_bow]
        lsi_query_vec = [[e[1] for e in query.lsi_model[b]] for b in bows]
        embeddings += lsi_query_vec
        hue += [gt.entity] * len(lsi_query_vec)
        # point marker size
        size += ['query']
        size += (['hit'] * (len(lsi_query_vec) - 1))
    return embeddings, hue, size


def calc_tsne_doc2vec():
    embeddings = []
    hue = []
    size = []
    for i, gt in enumerate(gts):
        query_vec = [query.doc2vec_model.infer_vector(gt.query.query_words)]
        for hit_idx, sim in gt.query.res_doc2vec:
            doc_words = query.documents[hit_idx][0]
            doc_vec = query.doc2vec_model.infer_vector(doc_words)
            query_vec += [doc_vec]

        embeddings += query_vec

        hue += [gt.entity] * len(query_vec)
        # point marker size
        size += ['query']
        size += (['hit'] * (len(query_vec) - 1))
    return embeddings, hue, size


def plot_tsne(filename, embeddings, hue, size):
    print('generating', filename)
    tsne = TSNE(n_components=2, verbose=0, perplexity=2, n_iter=3000)
    tsne_results = tsne.fit_transform(embeddings)
    df_subset = pd.DataFrame()
    df_subset['x'] = tsne_results[:, 0]
    df_subset['y'] = tsne_results[:, 1]

    plt.figure(figsize=(9, 9))
    sns.scatterplot(
        x="x", y="y",
        hue=hue,
        size=size,
        data=df_subset,
        legend="full",
        alpha=1.0
    )
    for idx, xy in enumerate(zip(tsne_results[:, 0], tsne_results[:, 1])):
        if idx % 6 != 0:
            continue
        plt.annotate(hue[idx], xy=xy, xytext=(-2, 2),
                     textcoords='offset points', ha='right', va='bottom',
                     fontsize=5)
    plt.savefig(filename)


print()
plot_tsne('plot-doc2vec.png', *calc_tsne_doc2vec())
plot_tsne('plot-lsi.png', *calc_tsne_lsi())
```