CS321 - Software Engineering

Isabella Feng

Project 2: Python2Vec

GitHub Repository: https://github.com/isabella-feng/CS421-project2

Abstract

In this project, I applied the Gensim Word2Vec model to source code of multiple python libraries to create Python2Vec, which is a machine learning structure that generate word embeddings based on Python code.

Results

First off all, I downloaded a few Python repositories in an automated way using Python. Then, I crawled through the repositories looking for Python files, and aggregated the Python files found into a single text file. After counting the number of lines and words and tokenized to words, I put the training set to the Gensim Word2Vec model.

Exploring the model, I discover that the Python2Vec model can accurately detect and perdict similar syntax. For example, it is able to find the closest words to "if" as elif, when, while, whether, etc. It also can find the closest words to terms, for example, it finds closest words for "numpy" as py, api, scipy, etc. To an extent it understands similar meanings of words pretty well, for example, it finds similar words to "number" as amount, part, ratio, presence, combination, total, numbers, etc.

It can also find similarity between words. For example, "numpy" and "math" are loosely similar with a similarity of 0.21401104. "true" and "false" are both booleans and are usually used in similar settings, so they have high similarity. "big_endian" and "define" are competely unrelated so they have very low similarity.

It can also find the words that doesn't match in a group of words. For example, among the words 'if', 'for', 'while' and 'string', 'string' is the only one that's not a syntax word. Python2Vec can successfully detect that. In another example, among the words 'string', 'integer', 'boolean', 'define', 'define' is the only one that's not a data type. Python2Vec can successfully detect that as well.

In analysis of the similarity between identifiers, we're able to have interesting discoveries. For one, the pair with highest similarity is 'ValueError' and 'TypeError', as both of them are errors and usually appear in similar settings. For another, common variable names, such as 'x' and 'y', also have high similarities, as they're usually just names and can be interchangeable.

On the other hand, it seems like variable names that carry certain meanings are unlikely to have similarity as others, such as 'plt' & 'obj', 'res' & 'fig' have very low similarity with negative values.

Here below is the Heatmap showing the similarity between each pair of words. The yellower the color, the higher the similarity.

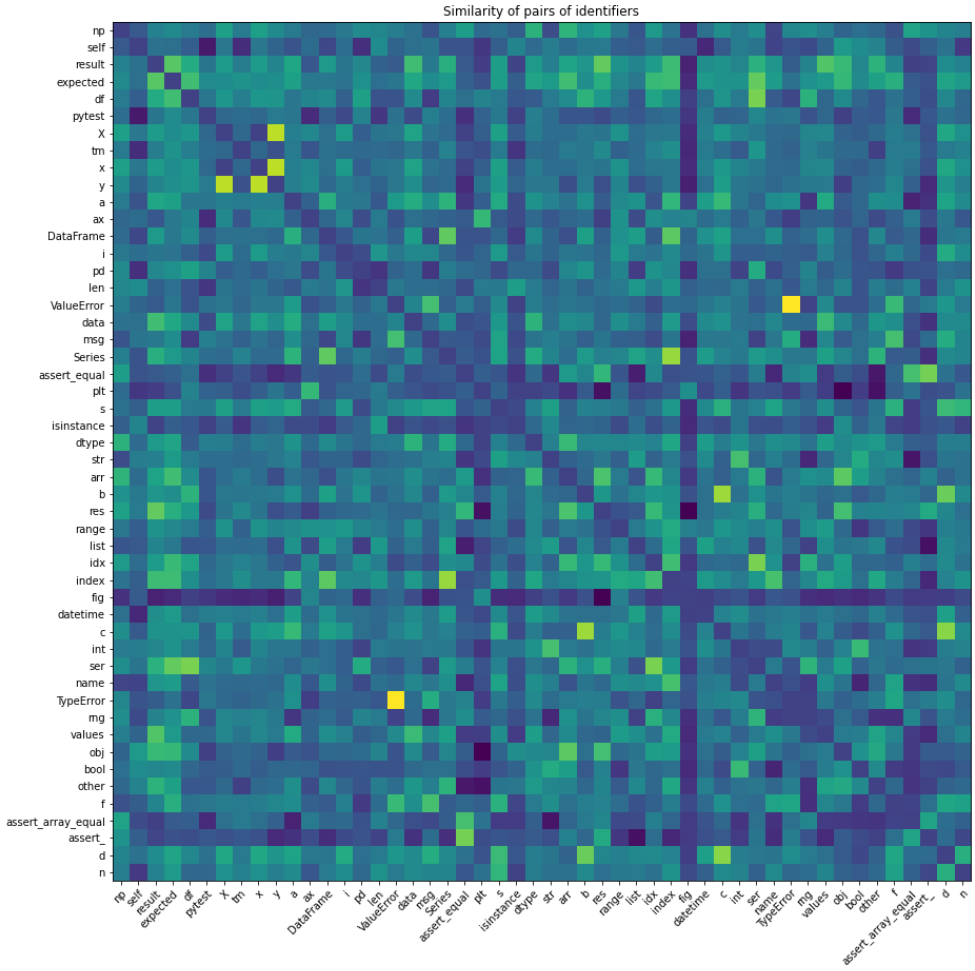

Fig 1. Similarity of pairs of identifiers

Furthermore, I enhanced the way to flag identifiers as similar/dissimilar. I categorize the similarities into 3 categories: Similar pairs are the ones with similarity > 0.5; Not-so-similar pairs are the ones with 0 < similarity < 0.5; and Dissimilar pairs are the ones with similarity < 0. In this grayscale Heatmap below, Similar pairs are white, Dissimilar pairs are black, and Not-so-similar pairs are gray. As we can see from the graph, the identifiers that have lots of whites are the variable and function names that carry certain meanings, such as "fig",

"assert_equal", and "plt".

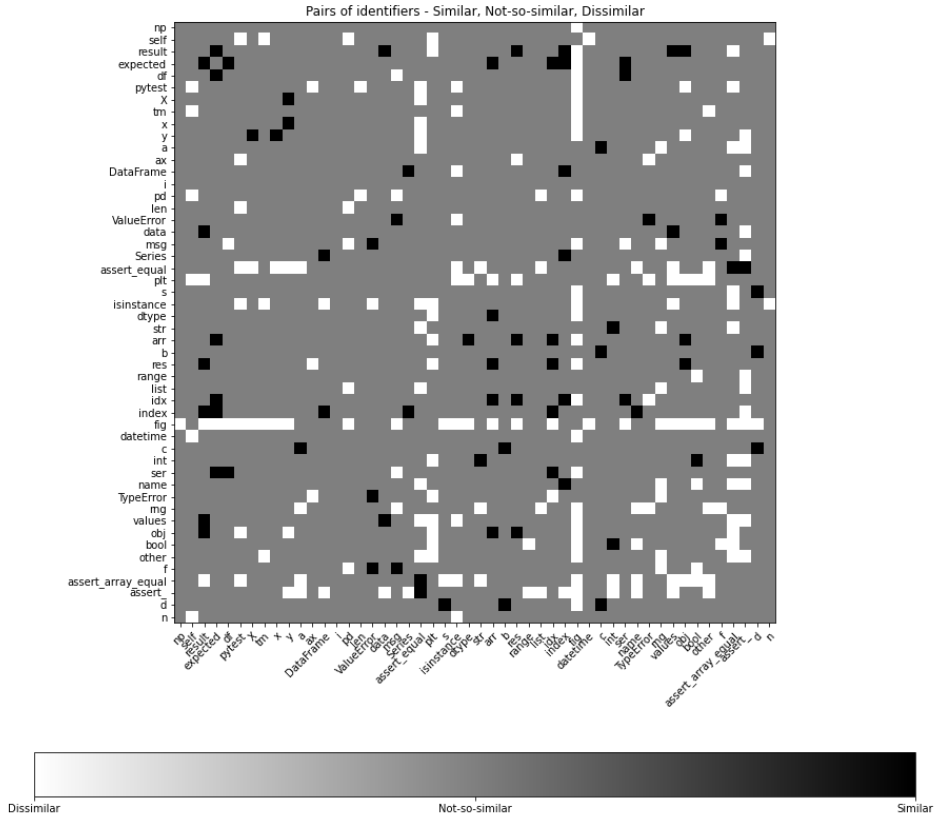

Fig 2. Similarity Categorization

Discussions

The analysis of similarity of the identifiers could provide us with some insights on how codes are made. For example, homogeneous variable names that don't carry certain meanings could be quite similar, which might decrease the readability of the codes, and may create confusion and increase the possibility of having bugs. In the opposite, variable names that carry certain meanings are unlikely to have similarity with others. This reminds us that when we name our functions and variables, it is the best to use less homogeneous names (such as 'x' or 'y'), and embed the meaning/intensino in the name instead.

Extensions

- Apply your Python2Vec model in other software engineering applications, such as autocomplete and suggesting method names. For this extension, I wrote a

function called "what_is_this_code" to summarize the content of a string of texts in one word. I achieve it by taking the mean of the vector of all words and find the most similar word.

```
test_text_1 = "import matplotlib.pyplot as plt"
print("test_text_1:", test_text_1, "\n\n")

result_1 = what_is_this_code(test_text_1)

print("This line of code is likely:")
for i in range(5):
    print(i+1, ".", result_1[i][0])
```

```
test_text_1: import matplotlib.pyplot as plt


This line of code is likely:
1 . matplotlib
2 . axisartist
3 . pyplot
4 . mpl_toolkits
5 . plotting
```

Fig 3. Extension - Example 1

In the above example, with the line of code that imports the matplotlib.pylot package, it successfully finds out "matplotpin" as the mostly likely keyword.

```
test_text_2 = '''from io import BytesIO
import os
import tarfile
import zipfile

import numpy as np
import pytest'''

print("test_text_2:", test_text_2, "\n\n")

result_2 = what_is_this_code(test_text_2)

print("This line of code is likely:")
for i in range(5):
    print(i+1, ".", result_2[i][0])
```

```
test_text_2: from io import BytesIO
import os
import tarfile
import zipfile

import numpy as np
import pytest


This line of code is likely:
1 . import
2 . contextmanager
3 . __all__
4 . distutils
5 . import_optional_dependency
```

Fig 4. Extension - Example 2

In the above example, with the line of code that imports multiple packages, it successfully finds out "import" as the mostly likely keyword.

Reference

I refer to Gensim Word2Vec documentation https://radimrehurek.com/gensim/models/word2vec.html for methods.