Analyzing the performance of synonyms.py

Danial Hasan & Dylan Vogel

2016, November 29

SUBPART (A)
Experimenting on a large corpus of text

A semantic descriptors dictionary is built from Marcel Proust's *Swann's Way* and Leo Tolstoy's *War and Peace*. Using the cosine similarity function our program correctly answers 70% of the questions in the provided *test.txt* file.

SUBPART (B)
Experimenting with alternative similarity measures

Using the same semantic descriptors dictionary and *test.txt* file from Subpart (A), two alternative similarity measures are tested. The negative Euclidian distance similarity measure returns 35% of the correct answers, while the normalized negative Euclidian distance measure returns 70% of the correct answers. In this case, the latter similarity function shows identical performance to the cosine similarity function tested in Subpart (A), while the former similarity measure shows decreased performance. The code used for the two similarity measures is provided below.

In [ ]:
```
def euclidian_similarity(vec1, vec2):

    vector_sub = {}
    combined_vec = {**vec1, **vec2}

    for word in combined_vec:
        vector_sub[word] = vec1.get(word, 0) - vec2.get(word, 0)

    return norm(vector_sub) * -1
```

In [ ]:
```
def euclidian_similarity_norm(vec1, vec2):
    vector_sub = {}
    combined_vec = {**vec1, **vec2}

    norm1 = norm(vec1)
    norm2 = norm(vec2)

    for word in combined_vec:
        vector_sub[word] = (vec1.get(word, 0) / norm1) - (vec2.get(word,
 0) / norm2)

    return norm(vector_sub) * -1
```

SUBPART (C)
Experimenting with smaller corpora of text: efficiency and performance

The performance of the algorithm was tested in terms of runtime and percent of questions answered correctly for varying volumes of text from *Swann's Way* and *War and Peace*. The following code was used to generate text files that contained between 10% and 100% of the given texts, with increments of 10%.

```
In [ ]: def subpart_c(filenames):

            for file in filenames:
                with open(file, 'r', encoding="utf8") as input_text:
                    input_text = input_text.read()

                    for i in range(1, 11):
                        new_file_name = "%s_%d.txt" % (file[:-4], i)
                        with open(new_file_name, 'w', encoding="utf8") as to_wri
        te:
                            slice = int(len(input_text) // (10/i))
                            to_write.write(input_text[:slice])
```

The generated text files were then used to build semantic descriptor libraries and tested against *test.txt*, with the results summarized in Figure 1. The code used to record the results is provided below.

```
In [ ]: def test_cases():
            times = []
            scores = []

            for test_no in range(1, 11):
                swann = "swann_%s.txt" % (test_no)
                war = "war_and_peace_%s.txt" % (test_no)

                start = time.time()
                semantic_desc = build_semantic_descriptors_from_files([swann, wa
        r])
                scores.append(run_similarity_test("test.txt", semantic_desc, cos
        ine_similarity))
                times.append(time.time() - start)

            return times, scores
```

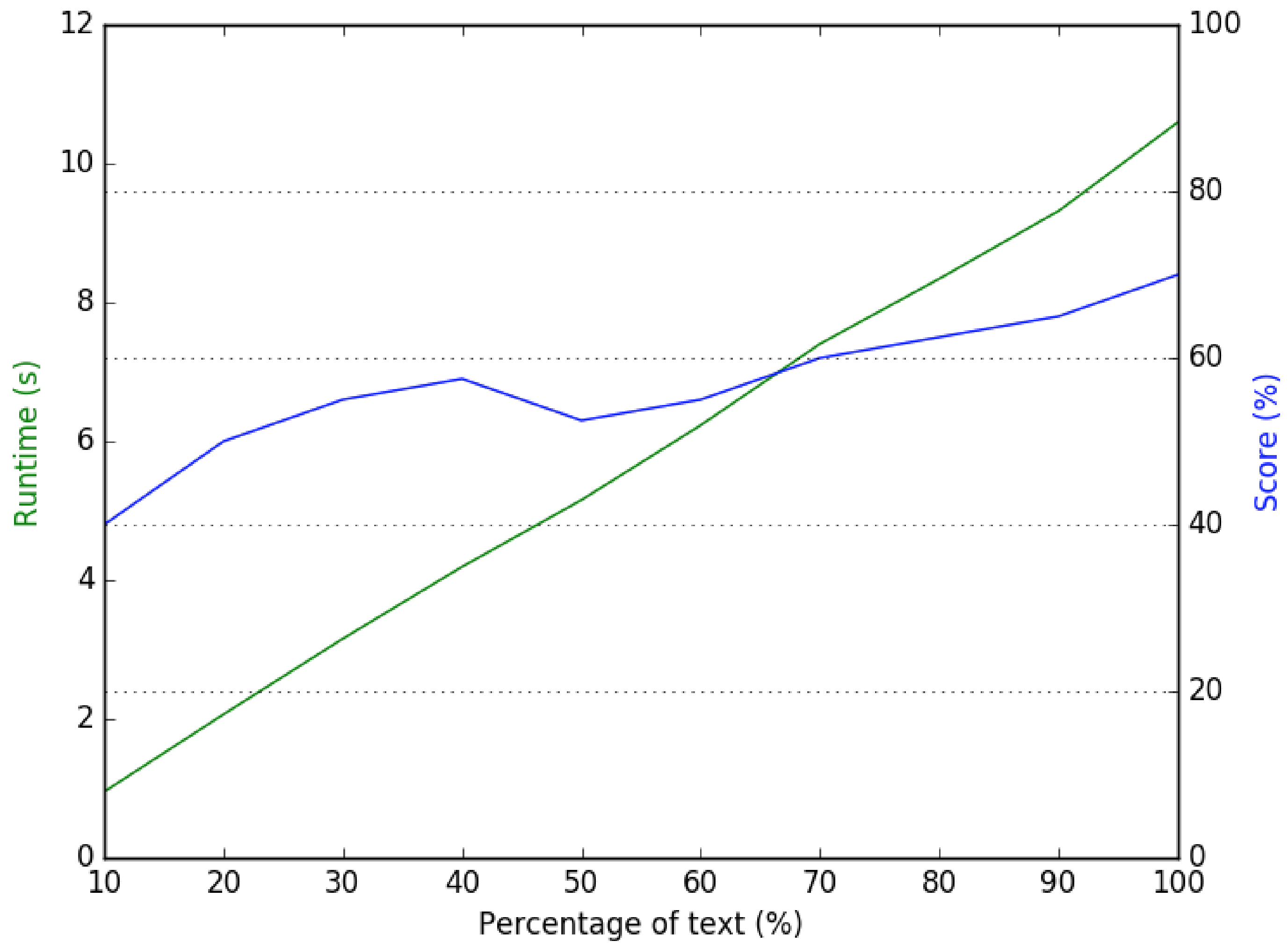

Figure 1: *Performance and runtime of the algorithm for different volumes of text from Swann's Way and War and Peace*