```
import synonyms as s
import time
import matplotlib.pyplot as mpl

def test():
    start = time.time()
    descriptors = s.build_semantic_descriptors_from_files\
    (['SwannsWay.txt', 'WarAndPeace.txt'])  # Actually build descriptors
    print("Done.\nRuntime was %s seconds." %(time.time() - start))
        # Track runtime for building descriptors
    print("Cosine:", s.run_similarity_test('test.txt', descriptors,\
    s.cosine_similarity))
    print("Total runtime was %s seconds." %(time.time() - start))
    print("Standard:", s.run_similarity_test('test.txt', descriptors,\
    s.sim_euc))
    print("Normalized:", s.run_similarity_test('test.txt', descriptors,\
    s.sim_euc_norm))
        # Compute and display similarity scores for all three methods

def get_graph():
    runtime = []
    accuracy = []
    tenths = range(11)  # Run from 0 to 10 tenths of the files
    for i in tenths:
        print("Now working on %d tenth(s)." %(i))
        start = time.time() # Keep track of time for each run
        desc = s.build_semantic_descriptors_from_partial_files\
        (['SwannsWay.txt', 'WarAndPeace.txt'], i)
        accuracy.append(s.run_similarity_test('test.txt', desc,\
        s.cosine_similarity))   # Build accuracy list
        runtime.append(time.time() - start) # Build runtime list
    mpl.close()
    mpl.subplot(1,2,1)
    mpl.plot(tenths, runtime, 'b-')
    mpl.title("Runtime")
    mpl.xlim([0,11])
    mpl.ylim([0,1.1*max(runtime)])      # To ensure proper scaling
    mpl.xlabel("Tenths of the books used")
    mpl.ylabel("Runtime (s)")
    mpl.subplot(1,2,2)
    mpl.plot(tenths, accuracy, 'r-')
    mpl.title("Accuracy")
    mpl.xlim([0,11])
    mpl.ylim([0,100])
    mpl.xlabel("Tenths of the books used")
    mpl.ylabel("Accuracy (%)")
    mpl.show()

'''In synonyms.py:

def build_semantic_descriptors_from_partial_files(filenames, tenth):
    return build_semantic_descriptors(get_mult_part_texts(filenames, tenth))

def get_partial_text(filename, tenth):
     '''#Return a list of sentences, each formatted as a list of words, from
    # the given amount of the file filename
    #
    # Args:
    # filename: the file to be parsed
    # tenth: the number of tenths to be used of the text from filename
    '''

    text = open(filename, encoding='latin1').read()
    text = text[:(len(text)*tenth)//10]
    text = text.lower().replace(",", "").replace("-", "").replace("--", "")\
    .replace(":", "").replace(";", "").replace("!", ".").replace("?", ".")
    split = text.split(".")
```

```
    for i in range(len(split)):
        split[i] = split[i].split()
    return split

def get_mult_part_texts(filenames, tenth):
    big_list = []
    for file in filenames:
        for list in get_partial_text(file, tenth)[:]:
            big_list.append(list[:])
    return big_list

def add_dicts(vec1, vec2):
    new_dict = {}
    for word in list(vec1.keys()):
        new_dict[word] = vec1[word]
    for word in list(vec2.keys()):
        if word not in list(new_dict.keys()):
            new_dict[word] = vec2[word]
        else:
            new_dict[word] += vec2[word]
    return new_dict

def get_negative_vector(vec):
    new_vec = {}
    for key in list(vec.keys()):
        new_vec[key] = -vec[key]
    return new_vec

def sim_euc(vec1, vec2):
    return -norm(add_dicts(vec1, get_negative_vector(vec2)))

def sim_euc_norm(vec1, vec2):
    norm_v1, norm_v2 = norm(vec1), norm(vec2)
    new_v1, new_v2 = {}, {}
    for i in range(2):
        vec = [vec1, vec2][i]
        for word in list(vec.keys()):
            [new_v1, new_v2][i][word] = vec[word] / [norm_v1, norm_v2][i]
    return sim_euc(new_v1, new_v2)

def norm(vec):
    '''# Return the norm of a vector stored as a dictionary,
    #as described in the handout for Project 3.
    '''

    sum_of_squares = 0.0
    for x in list(vec.values()):
        sum_of_squares += x**2
    return sum_of_squares**0.5
'''

if __name__ == '__main__':
    test()      # Parts a and b
    get_graph()     # Part c
```