Summary of the steps for this exercise

- In this homework, we use Python to complete these three tasks
- The first task is to create a function returns the likelihood that the Spring and Fall will offer that class. For extra credit, I also included the summer term.
- The second task is to return the special graduate course for each professor
- The last task is to creates vectors for all professors with respect to their graduate courses and plot them into a 2D space using t-SNE.

Task 1

In order to complete the first task, I first created a complementary function called course by terms which counts for the number of classes in each term for all the courses in the course info. Then, I calculated the total count and loop through each course to get the results. The methodology of the likelihood **P(Likelihood) = Course in one term /Total count of that course**

Task 2

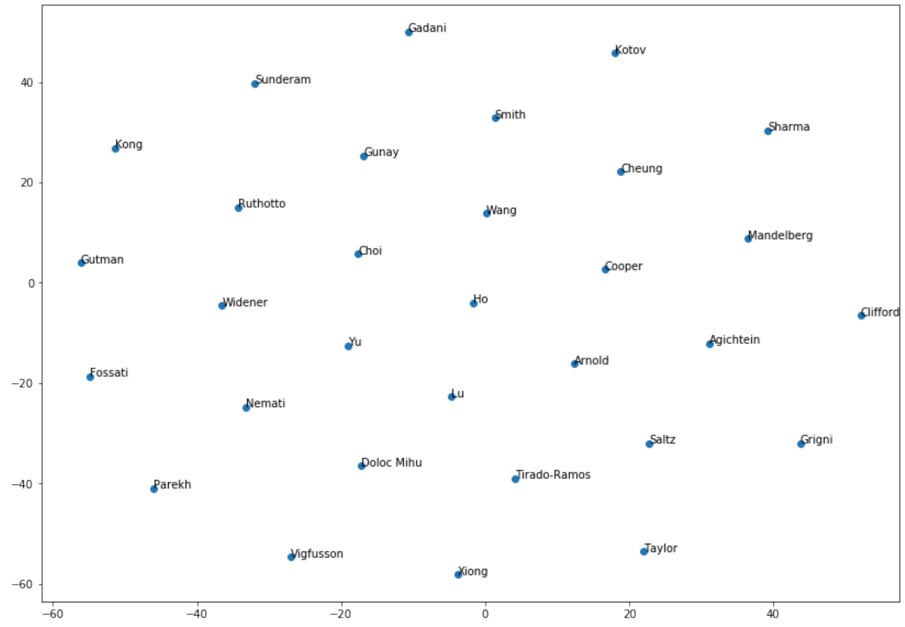
Similar to what we do in classes, I created a dictionary where the key is the professor name and the value is the set of special graduate classes the professor teaches. In order to keep the classes in a descending order, I changed the value to list and sorted them.

Task 3

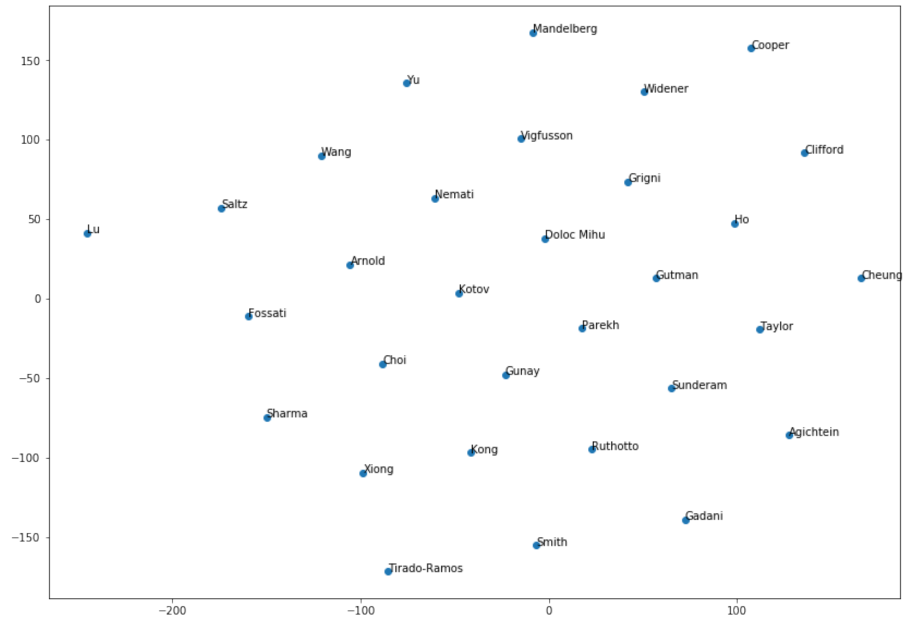
I complete the vector plot function in hw2.py that that takes course info and creates vectors for all professors with respect to their graduate courses and plot them into a 2D space using t-SNE.Because t-SNE give different result each time, I ran the program three time and see which one is the most appropriate.

Result Figures

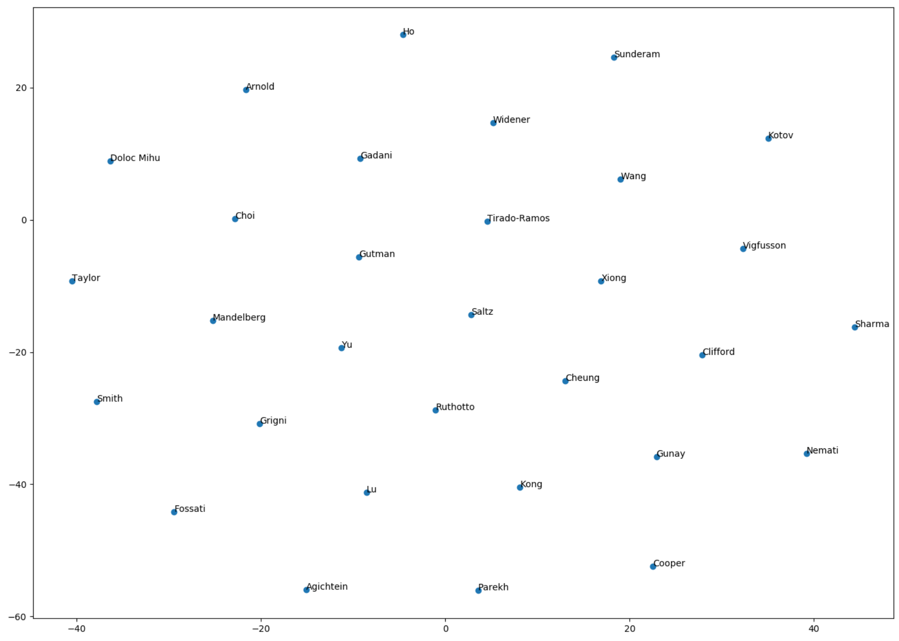
The result of the first time



The result of the second time



The result of the third time



Conclusion

Comparing these three graphs, graph 2 makes less sense to me as Professor Lu is away from other. And all other professors seem to be more similar. Comparing to graph 3, graph 1 is better because the graph should tell us the similarity of professor's graduation courses and research topic. For instance, Dr. Choi focuses on NLP and Dr. Ho and Dr. Ruthotto focus on machine learning. As there is many intersection between these two topics, these professors should be close to each other in the graph. Such idea is also illustrated by graph 1 rather than graph 2 or graph 3.

Appendix

The code for python functions

```
HW2:

#!/usr/bin/env python
# coding: utf-8

# In[280]:


# This is my own work: Huilan You
# ========================================================================
# Copyright 2018 Emory University
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ========================================================================
import csv
from types import SimpleNamespace
from collections import Counter


# load course
def load_course_info(csv_file):
    def skip(i, row):
        return i == 0 or int(row[11]) == 0 or row[12].strip() == '' or row[14].strip() !=

    def info(row):
        # term is normalized to (year, term_id) (e.g., 5181 -> (2018, 1))
        # term_id = 1: Spring, 6: Summer, 9: Fall
        r = row[0]
        term = (2000 + int((int(r) - 5000) / 10), int(r[-1]))

        # name = lastname,firstname
        r = row[12].split(',')
        instructor = (r[0].strip(), r[1].strip())

        return SimpleNamespace(
            term=term,
            subject=row[3].strip(),
            catalog=row[4].strip(),
            section=row[5].strip(),
            title=row[6].strip(),
            min_hours=int(row[8]),
            max_hours=int(row[9]),
            enrollment=int(row[11]),
            instructor=instructor)

    with open(csv_file) as fin:
        reader = csv.reader(fin)
        course_info = [info(row) for i, row in enumerate(reader) if not skip(i, row)]

    return course_info


# count the course by fall, spring, summer( for extra credit)


def course_by_terms(course_info):
    fall_courses = 0;
    fall_list = []
    spring_list = []
    summer_list = []
    for c in course_info:
        if c.term[1] == 9:
            fall_courses = (*c.term, c.subject, c.catalog)
            fall_list.append(fall_courses)
        elif c.term[1] == 1:
            spring_courses = (*c.term, c.subject, c.catalog)
            spring_list.append(spring_courses)
        elif c.term[1] == 6:
            summer_courses = (*c.term, c.subject, c.catalog)
            summer_list.append(summer_courses)

    return (Counter(t[2:] for t in fall_list), Counter(t[2:] for t in spring_list), Counte


def course_trend(course_info):
    """
    :param course_info: the output of load_course_info().
    :return: a dictionary where the key is a course ID (e.g., 'CS170') and
             the value is the likelihood of each course being offered in the Fall and Sprin
    """

    # TODO: to be filled
    courses = course_by_terms(course_info)
    total_count = {}

    # make sure total covers all course since some classes only offer in one term
    for course in courses[0]:
        total = courses[0][course] + courses[1][course] + courses[2][course]
        total_count[course] = total
    for course in courses[1]:
        total = courses[0][course] + courses[1][course] + courses[2][course]
        if course not in total_count:
            total_count[course] = total
    for course in courses[2]:
        total = courses[0][course] + courses[1][course] + courses[2][course]
        if course not in total_count:
            total_count[course] = total

    # calculation: the course offered in one terms/ the total count of the course is the li

    fall_average = {course: count / total_count[course] for course, count in courses[0].ite
    spring_average = {course: count / total_count[course] for course, count in courses[1].i
    summer_average = {course: count / total_count[course] for course, count in courses[2].i

    result = []
    for course in total_count:
        result.append({' '.join(course): (fall_average[course] if course in fall_average el
                                          spring_average[course] if course in spring_averag
                                          summer_average[course] if course in summer_averag

    print(result)

    return result


# help to identify the research courses and the undergraduate courses


def is_research_course(catalog):
    return catalog != '130R' and 'R' in catalog


def is_undergraduate_course(catalog):
    return int(catalog[0]) < 5


def special_topics(course_info, include_research=False, include_undergraduate=False):
    """
    :param course_info: the output of load_course_info().
    :return: a dictionary where the key is a professor name (e.g., 'Jinho D Choi') and the
```

```
            special graduate courses excluding research courses ranked in descending order
    """
    # create a dictionary where key is the professor name and the value is the set of
    # special graduate courses excluding research courses ranked in descending order
    d = {}
    for c in course_info:
        if (include_research or not is_research_course(c.catalog)) and (
                include_undergraduate or not is_undergraduate_course(c.catalog)):
            key = c.instructor
            val = (c.subject + c.catalog)
            # use set first to exclude the duplicates
            if key in d:
                d[key].add(val)
            else:
                d[key] = {val}

    # Change the set to list in a reversed order
    for key, value in d.items():
        value = sorted(list(value), reverse=True)
        d[key] = value


    return d


# Count for the number of each special course for all the professors


def courses_by_instructors(course_info, include_research=False, include_undergraduate=False
    d = {}
    for c in course_info:
        if (include_research or not is_research_course(c.catalog)) and (
                include_undergraduate or not is_undergraduate_course(c.catalog)):
            key = c.instructor
            val = (*c.term, c.subject, c.catalog)
            if key in d:
                d[key].add(val)
            else:
                d[key] = {val}

    return {k: Counter([t[2:] for t in v]) for k, v in d.items()}


# calculate the frequency of a professor for a particular class
```

```
def professor_frequency(prof_course_dict):
    return Counter([k for v in prof_course_dict.values() for k in v.keys()])


import matplotlib.pyplot as plt
import math
from sklearn.manifold import TSNE
import numpy as np


def vector_plot(course_info):
    """
    :param course_info: the output of load_course_info().
    """
    prof_course_dict = courses_by_instructors(course_info)
    prof_freq = professor_frequency(prof_course_dict)
    N = len(prof_course_dict)
    course_dict = {c: i for i, c in enumerate(sorted(list(prof_freq.keys())))}
    p2v = {}

    for prof_name, counts in prof_course_dict.items():
        vec = np.zeros(len(course_dict))

        for k, v in counts.items():
            if k in course_dict:
                i = course_dict[k]
                vec[i] = v * math.log(N / prof_freq[k])

        p2v[prof_name] = vec

    profs = sorted(p2v.keys())
    vectors = np.array([p2v[p] for p in profs])

    xy = TSNE(n_components=2).fit_transform(vectors)
    fig, ax = plt.subplots()
    ax.scatter(xy[:, 0], xy[:, 1])
    plt.show()

    fig, ax = plt.subplots()
    fig.set_size_inches(14, 10)

    x = xy[:, 0]
    y = xy[:, 1]
    ax.scatter(x, y)

    for i, p in enumerate(profs):
        ax.annotate(p[0], (x[i], y[i]))

    plt.show()


if __name__ == '__main__':
    csv_file = 'cs_courses_2008_2018.csv'
    course_info = load_course_info(csv_file)
    trend = course_trend(course_info)
    topics = special_topics(course_info)
    vector_plot(course_info)
    print(topics)
    # print(trend)
```