Stable Marriage Problem as a Solution to Weighted-Questionare Matching Algorithms:

Janeet Bajracharya

Consider two disjoint sets

$$M = \{m_1, m_2, ..., m_n\}$$

$$F = \{f_1, f_2, ..., f_n\}$$

We consider M to be the set of Males and F to be the set of Females. As in many implementations of this algorithm, the Males propose to the Female and we maintian that tradition. So each man has to be paired up with some woman unqiuely and vice versa.

More formally: We assume that we are finding bijections from $M \to F$ that is for every $m \in M$ we want there to be a unqiue $f \in F$ and vice versa. We find these bijective pairings using a Questionare that has k Categories of Questions (Life Style, Love Language, Romantic Preferences and etc) and each of those k Categories has l Questions. Therefore, in total we have $l \times k$ questions.

In the Python Code we represent the result of an entire session of n people answering $k \times l$ questions in a (n, l, k) dimension matrix.

For testing purposes we generate a fake dataset that follows a binomial distribution for responses because everything follows a binomial distribution. Here we make the row containing the answers for the questions have an extra element in the first index that represents the weight or importance of that entire category to the person. We also add 1 to all entries to make sure we do not have any $0'$s as a 0 weight would be disasterous!

We create two such fake datasets: one for the men and one for the women. We use the following code to do so:

```python
NUMBER_OF_PEOPLE = 20
NUMBER_OF_CATEGORIES = 5
NUMBER_OF_QUESTIONS = 10


def generate_responses():
    return responses = (
        np.random.binomial(
            n=4,
            p=0.5,
            size=(
                NUMBER_OF_PEOPLE,
                NUMBER_OF_CATEGORIES,
                NUMBER_OF_QUESTIONS + 1,
            ),
        )
        + 1
    )
```

The way that Stable Marriage Algorithms work, specifically Gale-Shapley Algorithm which is verified, proven solution to Stable Marraige, is that there is a table where each $m \in M$ has an ordered list of its prefered matches.

We represent this using a dictionary where the key is the $m \in M$ and the value is a list of $f \in F$ in order of preference [descending order in this specific case as we use Eucledian Distances and lower is better]. For example the following has a set of 0-4 men and women who are disjoint sets but they have the same names so when 0 best matches with 2 it is not the man 2 but the woman 2.

```
Male Preference Table
{
0: [2, 1, 4, 0, 3]
1: [3, 4, 2, 0, 1]
2: [3, 0, 4, 1, 2]
3: [3, 1, 2, 4, 0]
4: [3, 2, 4, 0, 1]
}

Female Preference Table
{
0: [3, 2, 0, 1, 4]
1: [4, 3, 0, 2, 1]
2: [1, 3, 4, 2, 0]
3: [4, 2, 3, 1, 0]
4: [1, 3, 0, 2, 4]
}
```

Creating this table is rather involed coming from the matrix of responses we have.

What we need to do is basically is for every man $m \in M$ match with every female $f \in F$ and calculate the eucledian distance between the responses and weight the responses with the weight that is found in the first index. If you have experience with numpy it helps. We can do this using a few vectorized operations and no loops !

```python
def generate_prefrence_list(male, female):
    male_dict = {}
    female_dict = {}

    for man_index in range(len(male)):
        current_man = male[man_index]

        male_dict[man_index] = []
        male_weight = current_man[:, 0]
        male_responses = current_man[:, 1:]

        for woman_index in range(len(female)):
            current_woman = female[woman_index]
            female_weight = current_woman[:, 0]
            female_responses = current_woman[:, 1:]

            result = male_responses - female_responses

            if female_dict.get(woman_index) is None:
                female_dict[woman_index] = []
```

```
        male_weighted_distance = np.sqrt(
            male_weight.reshape((len(male_weight), 1)) * (result * result)
        ).sum(1)
        female_weighted_distance = np.sqrt(
            female_weight.reshape((len(female_weight), 1)) * (result * result)
        ).sum(1)
```

We basically get the weighted eucledian distance for every single category and then we will end up having a $(1, k)$ vector where we get the eucledian distance per category between some man and some woman. Then we basically get the average of all the categories and then I have added the variance as well because of some explanation that is found in the code. Bisect insort allows us to insert things into the table in a sorted fashion such that we get the result in an ascending order of Eucledian Distance.

```
        male_weighted_distance = np.mean(male_weighted_distance) + np.var(
            male_weighted_distance
        )
        female_weighted_distance = np.mean(female_weighted_distance) + np.var(
            female_weighted_distance
        )

        bisect.insort(
            male_dict[man_index],
            {"name": woman_index, "value": male_weighted_distance},
            key=lambda x: x["value"],
        )

        bisect.insort(
            female_dict[woman_index],
            {"name": man_index, "value": female_weighted_distance},
            key=lambda x: x["value"],
        )

    return {i: [j["name"] for j in male_dict[i]] for i in male_dict.keys()}, {
        i: [j["name"] for j in female_dict[i]] for i in female_dict.keys()
    }
```

The return distance removes some meta information we needed and strips off all thevalues that help us sort things.

The Issue with the Stable Matching Approach:

It is completely one-sided, that is whatever side is making the proposals to get married gets the best matchings. Here, since we assume Men are making the proposals the Men will always find a Woman that they most prefer. Woman, on the other hand do not get to choose who they want to be with. Sure, they can reject any man that they do not prefer for a Man that they do prefer but the lack of action prevents them from seeking the maximal preference man or the man that they like the most! It is the case that tf their preference is reciprocated then only will they get their best match. If some man $m \in M$ that they do not prefer is the only one to propse to her and no one else proposes to her, she is stuck with him. Thus making this algorithm man-optimal and women-pessimal. Or in a gender neutral categorization: proposer-optimal.

Life Lesson: Be active, taking initiative; go out there and get what you want !

Ref: https://econweb.ucsd.edu/~jsobel/172f12/matchingnotes.pdf

Globally Optimal Match Making:

We would like to find a Pareto-Optimal Matching ie. a universally optimal matching where any other matching would cause someone else to be worse off. There are a few things we want to assume such that it makes matching a bit easier. We want a strict order in our preference lists as indifference causes a wide range of issues that are important and dealt with and some would argue realistic in real life situations but dealing with that is beyond the scope of match makingin this domain.

Ref: https://economics.brown.edu/sites/default/files/papers/2008-12_paper.pdf