Mapel Tutorial

Stanisław Szufa

November 15, 2023

Contents

1 Introduction

Mapel is a Python package that serve for simplifying simulations and experiments related to elections. The most fundamental functions that mapel offers are:

- Generating elections from numerous election models;
- Computing distances between elections;
- Printing a map of elections–that is, a graphical representation of elections in a 2d space, where each point is associated to a certain election.

For more details, please look at:

```
Szufa, S., Faliszewski, P., Skowron, P., Slinko, A., Talmon, N.
Drawing a map of elections in the space of statistical cultures.
In: Proceedings of AAMAS-2020, 1341-1349.
```

Citation Policy: If you are using this package, we kindly ask you to cite the above article.

1.1 Installation

For mapel to work, you will need **python 3.10** or newer.

To install mapel python package:

```
pip install mapel
```

Note that this will install necessary dependencies (e.g. matplotlip, numpy, etc).

To import mapel python package:

```
import mapel.elections as mapel
```

2 General Tasks

2.1 Generate Ordinal Election from Statistical Culture

Objective *Generate impartial culture election with 5 candidates and 50 voters*

In this section, you will learn how to generate ordinal elections from different statistical cultures. We will start by defining what we mean by an election.

Formally, an ordinal election is a pair $E = (C, V)$ that consists of a set of candidates C and a collection of voters V, where each voter has a strict preference order (vote), that is, each voter ranks all the candidates in C from the most to the least desirable one. We use term voter and vote interchangeably. In practice, an `OrdinalElection` is an abstract object that i.a., contains the following fields:

```
election.num_candidates      # number of candidates
election.num_voters          # number of voters
election.votes               # preference orders
```

By `votes`, we refer to a two-dimensional array, where each row represents a single vote.

Example 2.1.
E.g., `votes` = [[0,1,2,3],[2,0,3,1],[3,1,2,0]] refers to an election with three following votes:

$$0 \succ 1 \succ 2 \succ 3,$$
$$2 \succ 0 \succ 3 \succ 1,$$
$$3 \succ 1 \succ 2 \succ 0.$$

Solution To generate an election we use `generate_ordinal_election()` function:

```
election = mapel.generate_ordinal_election(
                                        culture_id='ic',
                                        num_candidates=5, num_voters=50)
```

A list of all implemented ordinal statistical cultures is available in Section 4.1.

2.2 Generate Approval Election from Statistical Culture

Objective *Generate impartial culture election with 20 candidates and 100 voters*

In this section, you will learn how to generate approval elections from different statistical cultures. We will start by defining what we mean by an election.

Formally, an approval election is a pair $E = (C, V)$ that consists of a set of candidates C and a collection of voters V, where each voter approves a certain subset of candidates. In practice, an `ApprovalElection` is an abstract object that i.a., contains the following fields:

```
election.num_candidates      # number of candidates
election.num_voters          # number of voters
election.votes               # list of sets
```

By `votes`, we refer to a list of sets, where each set represents a single vote.

Example 2.2.
E.g., `votes` = [{0,1},{1,2,3},{2}] refers to an election with three following votes:

$$\{0,1\},$$
$$\{1,2,3\},$$
$$\{2\}.$$

Solution To generate an election we use `generate_approval_election()` function:

```
election = mapel.generate_approval_election(
                                        culture_id='ic',
                                        num_candidates=20, num_voters=100)
```

A list of all implemented ordinal statistical cultures is available in Section 4.2.

2.3 Generate Ordinal Election from Votes

Instead of using a statistical culture you can also generate election based on your own votes using `generate_ordinal_election_from_votes` function.

```
votes = [[0,1,2,3],[2,0,3,1],[3,1,2,0]]
election = mapel.generate_ordinal_election_from_votes(votes)
```

2.4 Generate Approval Election from Votes

Instead of using a statistical culture you can also generate election based on your own votes using `generate_approval_election_from_votes` function.

```
votes = [{0,1},{1,2,3},{2}]
election = mapel.generate_approval_election_from_votes(votes)
```

2.5 Compute Borda Score

Objective *Implement a function that for a given ordinal election return Borda scores of all candidates*

First, we need to create `scores` list and fill it with zeros.

```
scores = [0 for _ in range(election.num_candidates)]
```

Second, we need to iterate through all the votes and add appropriate points to candidates.

```
    for vote in election.votes:
        for c in range(election.num_candidates):
            scores[vote[c]] += election.num_candidates - 1 - c
```

Solution The whole function will look as follows:

```
    def compute_borda_scores(election) -> list:
        """ Return: List with all Borda scores """
        scores = [0 for _ in range(election.num_candidates)]
        for vote in election.votes:
            for c in range(election.num_candidates):
                scores[vote[c]] += election.num_candidates - 1 - c
        return scores
```

2.6 Compute Distance between Two Elections

Objective *Compute the EMD-Positionwise distance between two ordinal elections*

To compute a distance we use `compute_distance` function which as an input takes two elections and `distance_id`.

```
distances, mapping = mapel.compute_distance(election_1, election_2,
                                                distance_id='emd-positionwise')
```

And it returns tuple, the distance and the mapping that witness this distance. If a given distance is not using a mapping, it returns `None` instead.

Solution We start by generating two elections, and then we compute the distance

```
    election_1 = mapel.generate_ordinal_election(
                                            culture_id='ic',
                                            num_voters=5, num_candidates=3)
    election_2 = mapel.generate_ordinal_election(
                                            culture_id='ic',
                                            num_voters=5, num_candidates=3)
    distance, mapping = mapel.compute_distance(election_1, election_2,
                                                distance_id='emd-positionwise')
```

Computing distances between approval elections works in the same way. A list of all implemented distances is available in Section 4.3.

3 Experiments

3.1 Generate Election as Part of Experiment

Objective *Generate impartial culture election with 5 candidates and 50 voters*

In this section, we introduce an abstract object called `Experiment`, which helps us keep things clear. Finally, we generate elections using the `Experiment` object.

An `Experiment` is an abstract object, which, for now, can be seen as a black box in which all the computation takes place. At first, it might be confusing, but in the long run, it simplifies things. Before carrying out any other operations we need to create an empty `Experiment`, for this, we use the function `prepare_online_ordinal_experiment()`, which returns an empty `Experiment`. So, in order to prepare an empty `Experiment`, type:

```
experiment = mapel.prepare_online_ordinal_experiment()
```

To give you a hint of what the `Experiment` is, we present some of its fields and methods:

```
experiment.elections
experiment.distances
experiment.coordinates
experiment.features

experiment.add_election()
experiment.add_family()
experiment.compute_distances()
experiment.embed_2d()
experiment.compute_feature()
experiment.print_map_2d()
```

Now, we will focus on the `add_election()` method. In order to generate an election, it suffices to run `add_election()` method, and specify the `culture_id`. For example, if we want to generate an election from impartial culture, we type:

```
experiment.add_election(culture_id='ic')
```

All elections added to the experiment are stored in a `experiment.elections` dictionary, where the key is the `election_id`, and the value is the `Election` object. If you want to specify your own `election_id`, you can do it but using `election_id` argument, for example:

```
experiment.add_election(culture_id='ic', election_id='IC')
```

By default, the generated election will have 10 candidates and 100 voters, however, if you want to generate an election with different number of candidates and different number of voters, use `num_candidates`, and `num_voters` arguments:

```
experiment.add_election(culture_id='ic',
                        num_candidates=5, num_voters=50)
```

If you want to change the default values not for a single election, but for all elections generated in the future, type:

```
experiment.set_default_num_candidates(5)
experiment.set_default_num_voters(50)
```

Solution Our aim was to generate impartial culture election (with 5 candidates and 50 voters) within the experiment. Below we present the code with the solution.

```
experiment = mapel.prepare_online_ordinal_experiment()
experiment.add_election(culture_id='ic',
                        num_candidates=5, num_voters=50)
```

3.2 Generate Family of Elections

Objective *Generate 20 elections from Normalized Mallows culture with norm-$\phi = 0.5$*

If you would like to add many elections from the same culture, instead of adding them one by one, you can add them as one family of elections.

```
experiment.add_family(culture_id='ic', size=10)
```

The main difference between `add_election`, and `add_election_family`, is the fact, that the latter function has an additional argument called size, which specifies how many elections from a given distribution will be created.

Moreover, note that instead of impartial culture, we want to generate Normalized Mallows elections, which are parameterized by *norm-ϕ*. In order to specify culture's parameters, we use `params` argument, which is a dictionary, where the keys are the parameters' names and the values are the value of given parameters. To generate a single Normalized Mallows election with *norm-$\phi = 0.5$*, we should type:

```
experiment.add_election(culture_id='norm-mallows', params={'normphi': 0.5})
```

Solution Joining the upper two things together we obtain the solution.

```
experiment = mapel.prepare_online_ordinal_experiment()
experiment.add_family(culture_id='norm-mallows', size=10,
                                 params={'normphi': 0.5})
```

3.3 Create Map of Ordinal Elections

Objective *Create a map of elections (from impartial and Norm-Mallows cultures)*

Creating a map of elections is an ultimate tool of this package. We divide the procedure into four major steps, which we describe in details one by one, with exception for the first step which was described before. The steps are the following:

1. Generate elections
2. Compute distances
3. Embed in 2D
4. Print the map

3.3.1 Compute Distances

In order to compute distances between elections, use the following function:

```
experiment.compute_distances(distance_id='emd-positionwise')
```

The distances are stored in `distances` field, which is a dictionary of dictionaries. If you want to access the distances, just type:

```
experiment.distances
```

Example 3.1.
Let us assume that we have three elections generated from impartial culture with the following ids: `ic_0, ic_1, ic_2`. Then, the `distances` (dictionary of dictionaries) look as follows:

```
{'ic_0': {'ic_1': 2.3, 'ic_2': 1.7},
 'ic_1': {'ic_0': 2.3, 'ic_2': 1.9},
 'ic_2': {'ic_0': 1.7, 'ic_1': 1.9}}
```

3.3.2 Embedding

In order to embed the elections into 2D Euclidean space, run:

```
experiment.embed_2d(embedding_id='kk')
```

The coordinates are stored in `coordinates` field, which is a dictionary of lists. If you want to access the coordinates, just type:

```
experiment.coordinates
```

More information about different embedding algorithms is available in section 4.4.

Example 3.2.
Let us assume that we have four elections generated from Normalized Mallows culture with the following ids: `mal_0, mal_1, mal_2, mal_3`. Then, the `coordinates` (dictionary of lists) look as follows:

```
{'mal_1': [0.2, 0.8],
 'mal_2': [0.4, 0.4],
 'mal_3': [0.3, 0.1],
 'mal_4': [0.9, 0.7]}
```

3.3.3 Printing

In order to print the map, run:

```
experiment.print_map_2d()
```

Initial Solution After combining four steps described above we obtain the following code:

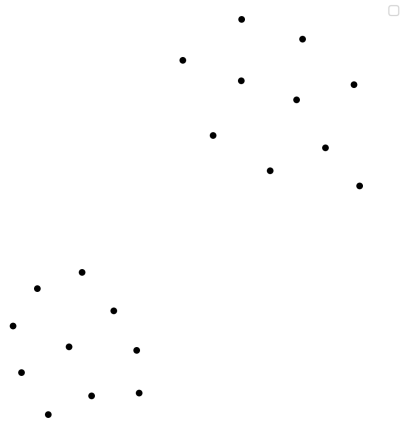

Figure 1: Example 1.

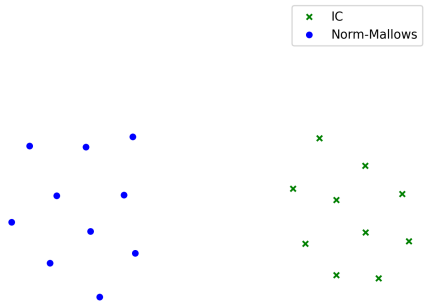

Figure 2: Example 2.

```
experiment = mapel.prepare_online_ordinal_experiment()
experiment.add_family(culture_id='ic', size=10)
experiment.add_family(culture_id='norm-mallows', size=10,
                                params={'normphi': 0.5})
experiment.compute_distances(distance_id='emd-positionwise')
experiment.embed_2d(embedding_id='fr')
experiment.print_map_2d()
```

As a result of the code above, you will see two separate black clouds of points (see Figure 1). In order to make the map more pleasing, we can specify the colors/markers/label of each election or family of elections separately. We do it via `color`, `marker`, `label` arguments.

Improved Solution

```
experiment = mapel.prepare_online_ordinal_experiment()
experiment.add_family(culture_id='ic', size=10,
                                color='green', marker='x', label='IC')
experiment.add_family(culture_id='norm-mallows', size=10,
                                params={'normphi': 0.5},
                                color='blue', marker='o',
                                label='Norm-Mallows')
experiment.compute_distances(distance_id='emd-positionwise')
experiment.embed_2d(embedding_id='fr')
experiment.print_map_2d()
```

The picture created by the improved version is presented in Figure 1. Moreover, for illustrative purpose in fig. 3 we present the map[1] for the 10x100 dataset of Szufa et al. [2020].

[1]Note, that the labels and arrows are created in PowerPoint and are not the part of mapel software.

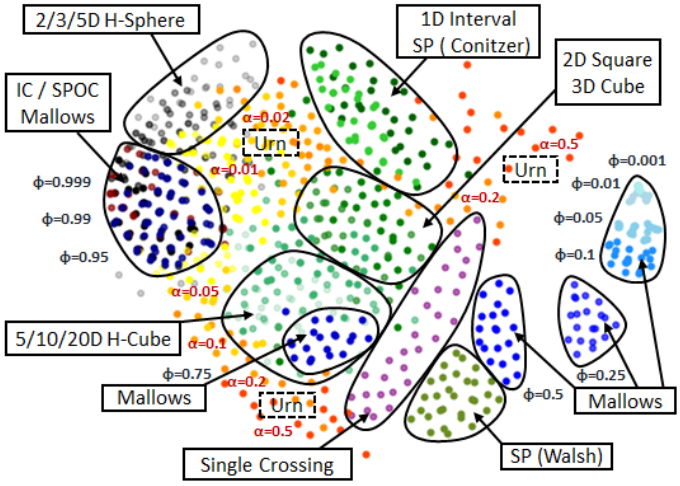

Figure 3: A map for the 10x100 dataset of Szufa et al. [2020].

3.4 Create Map of Approval Elections

Creating a map of approval elections in principle works the same way as creating the map of ordinal elections. The only differences are that we use different statistical cultures and different distances.

3.5 Coloring Map of Elections

It is interesting to color the map according to certain statistics, which we refer to as features.

3.5.1 Basic

We offer several preimplemented features. If you for example would like to compute the highest plurality score for all elections you can write:

```
experiment.compute_feature(feature_id='highest_plurality_score')
```

and then to print it use `feature` argument:

```
experiment.print_map_2d_colored_by_feature(feature_id='highest_plurality_score')
```

and if you want to access the computed values, type:

```
experiment.features['highest_plurality_score']
```

List of currently available features (that work for matrices as well):

Name	Argument
Highest Borda Score	highest_borda_score
Highest Plurality Score	highest_plurality_score

3.5.2 Printing

Basic arguments for `print_map_2d` function, are the following:

```
saveas=str    # save file as xyz.png
title=str     # title of the image
legend=bool   # (by default True) if False then hide the legend
ms=int        # (by default 20) size of the marker
show=bool     # (by default True) if False then hide the map
cmap          # cmap (only for printing features)
```

For example:

```
experiment.print_map_2d(title='My First Map', saveas='tmp', ms=30)
```

3.6 Offline Experiment

Offline experiment is very similar to online experiment, however, it gives the possibility to export/import files with elections/distances/coordinates/features etc.

3.6.1 Prepare Experiment

In order to prepare an offline experiment, run:

```
experiment = mapel.prepare_offline_ordinal_experiment(
                            experiment_id='name_of_the_experiment')
```

The function above will create the experiment structure, which looks as follows:

```
experiment_id
├── coordinates
├── distances
├── elections
├── features
├── matrices
└── map.csv
```

3.6.2 Prepare Elections

In order to prepare elections, run:

```
experiment.prepare_elections()
```

Elections are being generated accordingly to map.csv file. The map.csv will be described in details in the next section. While preparing the experiment, an exemplary map.csv file is being created automatically.

All prepared elections are being stored in `elections` folder, in a `soc` format. Definition of the `soc` format can be found at https://www.preflib.org/data/types#soc

3.6.3 map.csv

The controlling map.csv file usually consists of:

- size – number of elections to be generated from a given culture
- num_candidates – number of candidates
- num_voters – number of voters/votes
- culture_id – code of the culture; all cultures are described in detail in the next section
- params – dictionary with parameters of a given culture
- color – color of the point(s) on the map
- alpha – transparency of the point(s)
- marker – marker of the point(s)
- ms – marker size
- label – label that will be printed in the legend
- family_id – family id
- path – dictionary with parameters for generating a path of elections

3.6.4 Imports

If you have some parts of your already experiment precomputed, you can import them while preparing the experiment. However, note that they should be put in proper files. If you were precomputing those things using mapel, then you do not need to worry about it.

If you want to import particular things (different from default), you should specify them while preparing the experiment. For transparency, we suggest to always define them.

```
experiment = mapel.prepare_offline_ordinal_experiment(
                          experiment_id='name_of_the_experiment',
                          distance_id="emd-positionwise",
                          embedding_id="kk")
```

As to the features, if they are precomputed, the program will import them while printing the map.

3.7 Other

3.7.1 Own Cultures

If you want to add your own culture you can do this by using the `add_culture()` function.

```
experiment.add_culture("my_name", my_func)
```

The function is taking two argument, first one is the name of new culture, second is the function that generates the votes. The function that generates the votes can take any number of arguments, but among others it has to take `num_candidates` and `num_voters` parameters. Moreover, the function should return a numpy array with votes.

3.7.2 Own Features

If you want to add your own feature you can do this by using the `add_feature()` function.

```
experiment.add_feature("my_name", my_func)
```

The function is taking two argument, first one is the name of the new feature, second is the function that computes that feature. The function that computes the feature can take any number of arguments, but the first have to be an election. Moreover, the function should return a dict (usually it looks like this: `{'value': value}`, but it can contain several keys, for example, `{'value': value, 'std': std}`).

3.7.3 Own Distances

If you want to add your own distance you can do this by using the `add_distance()` function.

```
experiment.add_distance("my_name", my_func)
```

The function is taking two argument, first one is the name of the new distance, second is the function that computes that distance. The function that computes the distance can take any number of arguments, but the first two have to be elections. Moreover, the function should return a pair (a float and a list). The first returned value is the distance, and the second is the mapping witnessing that distance, if the distance is not using a mapping then it should return `None` instead.

3.7.4 Remark

Functions that store things in files, when rerun overwrite the previous data. For example, if the elections are already created but the command `mapel.prepare_elections()` will being executed, the old elections will be overwritten. The same is true for, for example, `compute_distances()` or `embed_2d()` functions.

3.8 Maps of Preferences (aka Microscope)

4 Reference Manual

4.1 Ordinal Election Cultures

To create an online experiment with ordinal elections use:

```
experiment = mapel.prepare_online_ordinal_experiment(...)
```

Similarly, to create an offline experiment with ordinal elections use:

```
experiment = mapel.prepare_offline_ordinal_experiment(...)
```

Below, we present the list of available ordinal election cultures.

Name	culture_id	Params
Impartial Culture	ic	–
Impartial Anonymous Culture	iac	–
Single-Peaked by Conitzer	conitzer	–
Single-Peaked by Walsh	walsh	–
Single-Peaked on a Circle	spoc_conitzer	–
Single-Crossing	single-crossing	–
Group-Separable	group-separable	tree
Euclidean	euclidean	dim, space
Pólya-Eggenberger Urn	urn	alpha
Mallows	mallows	phi
Normalized Mallows	norm-mallows	normphi

Brief parameters' explanations:

- tree (str) – name of the tree (available names: 'caterpillar', 'balanced'), if unspecified then tree is generated uniformly at random.
- dim (int) – dimensionality of euclidean space
- space (str) – name of the space (available names: 'uniform', 'gaussian')
- phi (float from [0,1] interval) – dispersion parameter
- norm_phi (float from [0,1] interval) – normalized dispersion parameter

4.2 Approval Election Cultures

To create an online experiment with approval elections use:

```
experiment = mapel.prepare_online_approval_experiment(...)
```

Similarly, to create an offline experiment with approval elections use:

```
experiment = mapel.prepare_offline_approval_experiment(...)
```

Below, we present the list of available approval election cultures.

Name	culture_id	Params
Full	full	–
Empty	empty	–
Impartial Culture	ic	p
Identity	id	p
Resampling	resampling	p, phi
Disjoint	disjoint_resampling	p, phi, g
Moving	moving_resampling	p, phi, g
Noise	noise	p, phi, type
Partylist	urn_partylist	g
Euclidean	euclidean	
Truncated Mallows	truncated_mallows	
Truncated Urn	truncated_urn	

Brief parameters' explanations:

- p (float from [0,1] interval) – probability that a voter approves a candidate
- phi (float from [0,1] interval) – noise
- g (int) – number of groups

More details about the cultures are available in the *How to Sample Approval Elections?* paper.

4.3 Distances

Below, we list the implemented distances:

Type	Name	Distance ID
Ordinal	Discrete	discrete
Ordinal	Swap	swap
Ordinal	Spearman	spearman
Ordinal	ℓ_1-Pairwise	l1-pairwise
Ordinal	EMD-Positionwise	emd-positionwise
Ordinal	ℓ_1-Positionwise	l1-positionwise
Ordinal	EMD-Bordawise	emd-bordawise
Ordinal	ℓ_1-Bordawise	l1-bordawise
Approval	ℓ_1-Approvalwise	l1-approvalwise

More details about the ordinal distances are available in *Drawing a Map of Elections in the Space of Statistical Cultures* and *Understanding Distance Measures Among Elections*, and about approval distances in *How to Sample Approval Elections?*.

4.4 Embeddings

Below, we list the available embedding algorithms:

Name	Algorithm	Params
Fruchterman-Reingold	fr	num_iterations, num_neighbors, radius
Multidimensional scaling	mds	-
t-distributed stochastic neighbor	tsne	-
SpectralEmbedding	se	-
LocallyLinearEmbedding	lle	num_neighbors
Isomap	isomap	num_iterations, num_neighbors
Kamada-Kawai (by Sapała)	kk	-