Paula Rivière

1 The return of the Data Miner

1.1 Cleaning with bash

In order to build a matrix with all the features that may determine the different planet's affinity, we need to merge the data contained in the following archives:

- list-planets.csv
- data-planets1.csv
- data-planets2.csv

A rapid inspection with bash shows that **data-planets1.csv** contains 91 delimiters per line:

```
cat data-planets1.csv | awk -F ";" '{print NF-1}'
```

However, there is an incomplete line, corresponding to planet 555:

```
cat ../data-planets1.csv | awk -F ";" '{print $1,NF-1}' | grep " 90"
```

Result:

```
00555 90
```

Since we are not sure that the missing column is the last one, and given that it corresponds to a single example of our original 2500 planet set, we will remove this line from the file before generating the table.

Besides, there are empty values, as can be seen with

```
grep ";;" data-planets1.csv |wc -l
```

We could remove then now,

```
sed '/;;/d' data-planets1.csv > data-planets1-clean.csv
```

but we prefer to do the cleaning at the database. In this way, we will only do with bash the cleaning needed to be able to import the data at the database.

Alternatively, some extrapolation technique could be used to estimate those missing values and not removing the planets. However, since it is a small sample of our set, we will just remove them.

1.2 Creating the database with Postgres

We create a database, and connect to it. Once inside we need to create 4 tables, corresponding to the following files:

- list-planets.csv: Table **LP**: id, names and coordinates (x,y).
- data-planets1.csv: Table **NP1**: id and 91 features (dp1_f1 to dp1_f91)
- data-planets1.csv: Table **NP2**: name and 73 features (dp1_f1 to dp1_f73)

The scripts to create and load each of these tables are attached at folder SCRIPTS:

- DP1_create.csv, DP1_load.csv
- DP2_create.csv, DP2_load.csv
- LP_create.csv, LP_load.csv

For each table, the different variable types of the columns are taken into account.

Besides, there are two scripts that join these data to generate the features matrix X and the target array y:

- join_tables_X.csv
- prepare_y.csv

The different steps are explained at the scripts.

2 The Machine Learner strikes back

We need to determine the affinity of a new planet based on the data we have: features X and affinities y. For that we use two methods of linear regression:

- Ordinary least squares, in which the cost is calculated as

$$L(y, \hat{y}) = \sum_i |y^{(i)} - \hat{y}^{(i)}|^2 \tag{1}$$

- Ridge regression, in which we introduce a regularization term to penalize high coefficients

$$L(y, \hat{y}) = \sum_i |y^{(i)} - \hat{y}^{(i)}|^2 + \alpha \sum_i |\theta_i|^2 \tag{2}$$

The models are calculated with program affinity_model.py.

2.1 P-scores

In order to determine which features are not relevant for the determination of the planet's affinity, we calculate the p-scores: probability of the null-hyphothesis, in which the coefficient's value is 0. Results show that there are several features that do not contribute much to the affinity:

Features from data-planets1.csv: 22, 33, 50, 57, 58, 59, 62, 63, 65, 66, 72, 73, 74, 79.
Features from data-planets2.csv: 3, 7, 11, 31, 36, 37, 38, 42, 43, 44, 45, 46, 62, 63, 67, 68, 69, 72.

If the computational cost becomes too high (i.e. when the number of planets increases considerably), we could consider removing the columns with the highest p-scores from the feature matrix for speeding the computation while maintaining a reasonable accuracy.

2.2 Ordinary least squares

For both regression models we will use a division into train set (89%) and test set (11%). The training is done with the train set, and the score calculated with the test set.

The obtained score for the test set with this model is 0.923.

2.3 Ridge regression

After trying with a set of alphas, it seems that the regularization parameter tends to zero. The obtained score for the test set is 0.923 as well.

What this means is that there is no need for regularization: there is no high variance, but rather high bias. We would need to try a different model (polynomial regression) to improve our model.

3 A spatial hope

3.1 Evil outposts in the vecinity

In this case we need to calculate the number of enemy outposts within a radius of 15k GUs of others. For this, we load the data for the evil outposts from file evil-outposts.csv. The program near-planets.py calculates these numbers and stores into table NEO (near evil outposts).

The scripts for create and load these tables are:

- ES_create.csv, ES_load.csv
- NEO_create.csv, NEO_load.csv

The EO with the maximum number of EOs in the vecinity is that with id=320532, which is near other 20 outposts.

3.2 Optimal location

For calculating the optimal location of new rebel outposts, we would need a new model whose target is (x,y), the location of the new rebel outpost.

We should define the feature matrix based on the data that we want to maximize/minimize. Some features that will determine the success of the new rebel outposts will be:

- 1/D, where D is the distance to the nearest enemy outpost.
- We would like to maximize the presence in the vicinity of planets with high affinity to the rebellion and high population. Therefore, a possible feature would be affinity×population for closer planets, for example, the sum of affinity*population for every planet inside a radius of 15k GUs.

The feature matrix could then have the columns: (x,y,1/D, $\sum_{nearplanets}$affinity×population).