Project Closure Report

Project Name: DATA PROJECT MOOVEO

Project Partner: Romain ETAY, Mooveo

Year *(4 or 5)* **and Team Number:** 4, 94

Prepared By

Document Owner(s)	Project Role (if defined)
Vincent DEBANDE	Project Manager
Clément LAJOUX	
Ludovic CHEVALLIER	
Simon PONGAN	

TABLE OF CONTENTS

1 PROJECT CLOSURE REPORT PURPOSE

The Project Closure Report contains key descriptive information about the project. As the last document written on the project it describes the satisfaction of the customer with the project. It analyzes the outcome of the project and the process by which that outcome was produced. Its purpose is twofold: to ensure that the closure activities are carried out properly and to facilitate the transfer of knowledge (or the transfer of experience if the project is not completed or if a follow-up to the project is envisaged) to the customer's organisation."

2 Background

The Project Closure Report is completed by extracting the data already produced throughout the project: the analysis, the document already written and the final results. The added value consists in the aggregation of the data within a single document, the analysis of the overall differences, the synthesis of customer satisfaction.

3 Project Descriptive Reminders

- Database clusterisation | Matching missions with talents
- Vincent DEBANDE | Clément LAJOUX | Ludovic CHEVALLIER | Simon PONGAN
- Mooveo
- Mooveo needed to implement database clusterisation to recommend skills to a talent based on the skills he already has. The second part is the matching of a mission and talents: scoring of a talent based on its location, skills, languages, and Availability

4 Objectives Set versus Results Achieved

4.1 Project Initial Objectives

Clusterisation

Based on the apparition of skills in all the jobs, define category of skills that appears in the same jobs. Computation of distance between skills (Dice distance because all the data is qualitative).

The clusterisation would have been used by Mooveo to recommend skills to talents.

Matching

Matching between mission and talents based on:

The location: How to determine how much it's easy to travel from point A to B. If a talent is:

- **in the same country** *within a given range*: use of Google Maps API
- **in the same country** *not in the range*: computation of the distance based on coordinates)
- **Not in the same country** use of the "power" of a passport and the needed Visa from country A to B plus the raw distance between the two locations

Skills: Score how much a talent has the needed skills based on the importance of a skill for a mission (importance of the skill on a mission, from 1 to 5)

Languages: Number of languages a talent has based on the needed languages

Availability: Boolean

The goal is to give indicators to a client, not to decide for him, so the objective is to send score without any decision (we don't decide that a talent available 2 days after the mission's start is better than another available 2 days before)

Deploy the code so Mooveo can use it.

4.2 Results Achieved

Clustering

The clustering was not successful because there wasn't any redundance of the skills in the jobs, so the computed distance between the skills was pretty high: All the skills are different. We found small "clusters" of skills representing a job (some set of skills appears one time in a job, so they are pretty similar)

To make a clusterisation we need to have redundance, skills must appear several times and a minority of skills can appear just one time.

You can find a detailed analysis in annex documents.

Matching

The matching was successful. The real work was to define a method of scoring, not overlap on the client's decision and just give indicators.

Important, the ratings must be between 0 and 1. 1 being a perfect match, 0 being the worse match.

The data is query directly on Mooveo's database thanks to a connection string and the use of an ORM (sqlalchemy)

All scorings have been done:

Skills

On the mission side, skills have a grade of importance (from 1 to 5)
On the talent size, skills have a rating of knowledge (from 1 to 5)

To compute a score, we make a ratio of: (the sum talent's skills rating of knowledge * grade of importance) by (the sum of grade of importance * maximum grade of importance (5))

Languages

Sum of the languages / number of required languages

Availability

Boolean: is the talent available on the given range of dates?

Distance

They are multiple cases:

If a talent is *in the same country and in a given range* (threshold at 80km):

Use Google maps API to compute a time travel (in minutes) in transport if distance is smaller than 50km, in car otherwise.

We scale the time with the function f(x) = 1 – (1/(1+x))

If a talent *is in a country not in the given range*:

We compute and scale (f(x) = 1 – (1/(1+x)) the distance based on coordinates

If a talent *is not in the same country*:

We compute the raw distance, scale it with 1 – x/20000, 20000 being the half of the circumference of the Earth (the greatest distance to travel from point A to B)

We apply a coefficient based on the type of Visa required between 2 countries (Passport Index): {'NA': 0, 'VR': 0.1, 'ETA': 0.1, 'VF': 1, 'VOA': 0.9, '15': 0.2, '21': 0.3, '90': 0.5, '120': 0.6, '180': 0.7, '360': 0.8}

The numbers are the number of days a person has the right to be in a country without a Visa.

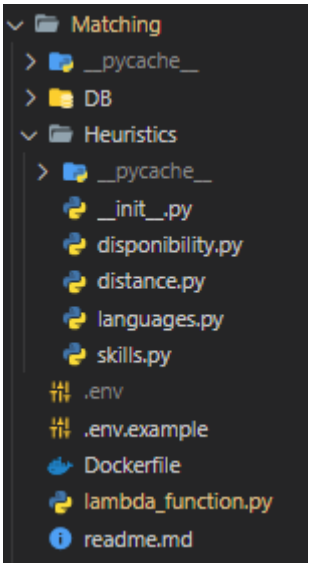
4.3 List of Deliverables

Clusterisation

Jupyter notebook containing the analysis made in the first part.
Data scrapped from Pole Emploi.

Matching

API deployed with AWS Lambda and Docker.
A guide to deploy and update the API on AWS servers.



We store our model in DB > model
In DB we can find db_loader which make a link with Mooveo's Database.
There is also a .csv with the required visa to travel from country A to B.

In Heuristics folder, we have all the heuristics for the different criteria.

Finally, in the base repository, we can find lambda_function, which is the main script. In there, we actually load the data and call the function to compute scores for a given offer.

All the code for both parts on GitHub (link in annexes).
The code has a good documentation, a readme is provided.

A user guide in PDF is also provided to deploy and update the API on AWS Lambda with Docker.

5 Technical Review

Clustering

The first part of the project being Data Analysis and Machine Learning (clusterization of a qualitative database), we chose to use Python with Jupyter Notebook. Our client had no constraint about the language.

Python is the main language for Data Science: it has a lot of documentation and tons of libraries designed especially for Data Science. Moreover, our courses were mainly in Python, so it was the opportunity to go further and develop our skills.

The data we used were stored in a .JSON document structured as follows:

A job is composed of 3 attributes:

- Name (string)
- Id (string)
- Skills (array of objects)
-

A skill is composed of 2 attributes:

- Name (string)
- Id (string)

All attributes for both objects are required.

Building the matrix of jobs:
In order to build a matrix of skills per jobs, we first, serialised the data so we could use objects. Then we structured the matrix all the skills as rows and the jobs as columns. With a “True” value if a skill is in a job, “False” otherwise.

How to determine if a skill is close to another one?
After research, we found that Dice Distance could give us the distance for categorical data. Since we don’t have any weight in the skills, we could use it.

The distance for two skills is [0,1]:

- 0: No jobs in common
- 1: Every value is common to both skills

We now had a square matrix with 0 in diagonal.

To compute this matrix, we used the pdist and squareform functions from scipy package.
To visualise the matrix, we built a HeatMap using seaborn package.

From the distance matrix previously built, we could start and see how much our data were linked. We used the AgglomerativeClustering algorithm from the sklearn package with the average linkage.

Basically, we saw that every skill represented a unique cluster (or that skills in the same job represented a cluster).

Why is that? We determined that 50% of our skills had an occurrence of 1 or 2. Therefore, there was not enough redundancy in the data to determine similarity between the skills.

When we applied filters, the results were better but not satisfying.

To see if we could get better results, we tried to get new data. We got new skills for Mooveo's jobs. Using Selenium, we search through Pole Emploi if the job existed or not. If so, we opened all "Fiche Métier" corresponding to that job and we added the skills into our job.

We applied the same method on our new data set. The results were better because we had a lot more skills per jobs and the skills appeared in different jobs. But still, we couldn't make clusters. Then we tried to replace the jobs per their category on PE (Pole Emploi) and build a quantitative matrix with the number of times a job appeared in a job of the given category. We computed the Euclidian distance; no results were usable.

We scrapped the whole database of Pole Emploi in order to try and cluster the data. But it was built in the same way as Mooveo's, there was no redundancy in the jobs. So, we couldn't determine any clusters. We did the same thing as before and replaced the jobs per their category. Now, the skills were all the same instead of being all different. No cluster was discovered.

Even if the clusterization of the database wasn't possible, the main goal was to recommend a skill based on a given skill or job.

So, we used the Pole Emploi database and built functions to give the top 5 skills from a job:
We investigated the category of a job and returned the 5 most used skills.
For a given skill, we looked the category where it appeared the most and returned the 5 most used skills in this category.

Sum up of technical details

We used:

- Data Analysis/Visualization (Seaborn/ ACM with R)
- Unsupervised Machine Learning (Agglomerative Hierarchical Clustering)
- Web Scrapping (Selenium)

The end of this part was around the mid of February.

Matching

Second part of the project that we started straight after the first part.
We decided not to use fancy techniques of machine learning for this part because it could be solved easily with basic heuristics moreover, we didn't had data to train our model.
We took a module approach app to make the maintenance easy and readable.
So, each heuristic has its own file and then all combined in one that the lambda entry point will then call.

The first part of this project was about thinking how to calculate a score over already defined criteria:

- Distance between user and offer
- Disponibility
- Languages
- Skills

The most challenging one was the distance heuristics since it takes a lot of multiple case.
If the talent is under a threshold distance (150km) we use Google Maps API to get the travel time in car and public transportation and then calculate a score between 0 and 1 based on the travel time.

You may ask why to use a threshold distance like this, it is because we want to limit the usage of the Google API since it is not free, also if a person is over 80km we assume that this person would prefer to move out of his place to get closer to his future job.

For people that do not live in the same country as the mission, the heuristic considers the easiness of moving to another country thanks to the passport index of the talent and the country of destination. So, we apply a coefficient on the distance based on the type of Visa we need to get to the country (Visa Required, Visa on Arrival, No Visa for X days).

For the other heuristics we look if the skills or languages are in the user skill set and apply a malus if not.

Mooveo wanted us to deploy our app on AWS Lambda (which is a server-less computing platform) and make our program communicating with their database. We produced a user guide to make them able to deploy it on their own with the good connection string.

We also provided clear documentation in the code and a readme on GitHub.

6 Planned Resources vs Used Resources

Since there were no deadlines and Mooveo trusted us with the realisation of the project, there was no overlapping and delay.

However, the first part on the clustering was too long and we should have ended it a month before it's actual end. It would have allowed us to go further in the development of the API. But we realised all the features Mooveo wanted and the API can be deploy in production.

7 Methodological review

We scheduled appointment on a weekly basis with Mooveo to make sure there was progress in the project.

During these meetings we could present our work, talk about the encountered problem and present the solutions.

Mooveo could approve or reject solutions, review their needs based on our work, the project was conducted with Agile method.

The role repartition within the team wasn't really defined, all the members were welcome to participate (with ideas, code…) but no roles were attributed because the project didn't require to divide the tasks and make them in parallel.

At the end of the clustering Vincent wanted to try some things while the team started to think about matching. The roles were attributed when needed. We worked with GitHub so all the members could add features and access to the last version of the code.

We scheduled meetings on our own at the end of each week to make a point on our advancement, the tasks we needed to finish before the next meeting and the points we wanted to talk about.

Our management of the project was good because the team had a good communication and all members were listened equally, no ideas rejected. The role definition wasn't an issue because the deadlines weren't imposed, and the structure of the project didn't require any work in parallel.

The weekly meetings were short and efficient, really useful to talk about the project, make a point on our work and move on with the issues.

8 Risk Management

The main project risks were during the clustering. We didn't know if the clusterization was possible or not.

We tried a lot of things to perform it and could have tried more things and run behind schedule. We stopped that part with the agreement of Mooveo to focus on the matching.

On the matching, the technical risk was to make an API not easy to modify, we structured our code with the idea of 1 part = 1 file, so every scoring method can be update easily and the code has a great readability.

9 "Post-project" tasks

A detail regarding the data in Mooveo's database needs to be evaluate in the last meeting. Based on the answer of Mooveo, we will have to make modification of the code and update our GitHub Repository.

We will perform the last tests regarding the modifications we have to make.

10 Satisfaction of customer and users

On our side of the project, we really liked the interaction with Mooveo and all the meetings. They didn't give us limits or didn't impose things to do in a given order: they trusted us to realise the project in the order we wanted.

They seem to appreciate our work.

11 Project Completion Recommendations

Best practices

The short meetings (10-20 minutes) on a weekly basis are really useful and efficient. It allows a great communication between the team and the partner.

Meetings within the team on a weekly basis allow the members to be updated on the project even if they couldn't work on it during the weeks. The meetings should cover at least: new features, errors, features to be implemented, tasks to do before the next meeting with the partner, new ideas.

Listen to all Team's members is really important even if some of them are less technical than others: All ideas are good to take into consideration.

Improvements

Even if we were efficient during the project, we would have done more things with a **better role definition**.

During the project we had goals like: Do a clustering, then do a matching without proper **deadlines** of each parts and subparts. To be more efficient and set limit (so, limit the risks) we should have set deadlines.

We also should have **defined subparts** of the project beforehand and create a better structure with deadlines for each subpart. It worked well for our case because the project didn’t require results on a deadline, it was more of an exploration project, the needs of the partner evolved over time. But with a better organisation and a better setup, we could have been even more efficient.

12 Annexes to the closure report.

Please list the annexes to the report

GitHub

https://github.com/vinsJ/mooveo-pi2

In the repo, you'll find two parts:

Clustering: Data scrapped from Pole Emploi and the Jupyter Notebook in HTML

Matching: All the code and a user guide in PDF