CENTRALESUPELEC

Introduction to Machine Learning

FINAL REPORT

Authors :
Aurélien Pasteau
Nicolas Gabrion
Marion Gobet
Vincent Bouget

Professor :
Fragkiskos Malliaros

CentraleSupélec

Abstract

This report presents an analysis and a comparison of four different algorithms (linear regression, tree and AdaBoost algorithm, neural network and nearest neighbours) which aim is to predict the results of French presidential elections. We used large databases from the French government, giving results of the 2017 elections per candidate and socio-economic data on a local scale, and pre-processed it to select the most pertinent features.

In this report, algorithms are compared so as to find the most performant one. Their strengths and weaknesses are spotlighted, and provide all the elements to decide which method is the most appropriated to predict the results of the election in 2022, and to quantify its reliability.

Introduction/Motivation

Our project aims to find a reliable way to predict the results of each candidate to the French presidential elections on a local scale. In fact, precise election results prediction has always been a challenge to solve because of the advantages it can provide to someone achieving it. Also, there are many organisations trying to predict those results by going ask people who they want to vote for, but this only works a short time before the election and it requires to go out and collect data from people one by one, which isn't efficient and can take a long time.

Here, the goal is to only use more general data about the socio-economic and demographic situation on a local scale, which doesn't require a long time of data collection since all this data can be found online and is free to use.

With this project, we want to answer the following question: is it possible to obtain a reliable prediction of each candidate's proportion of votes locally by working with data engineering methods and then applying machine learning algorithms to this open source socio-economic and demographic data?

If the answer to that question is true, one could imagine lots of applications to this tool : one could sell candidates information about their future results in the places where they are not sure whether they are going to win or not, or one could even bet on ranges in which the candidate's results are going to be and earn money with it. More simply, one could also provide the people with predictive information about the election.

Problem Definition

At first, we aimed to predict the result of 2022 presidential election based on the results of the 5 previous election. Indeed, we decided to predict the result of every candidate in every city/town. But as we started working on the project, we realized that our idea was definitely unrealistic because the candidates are different in every election so that was nearly impossible to predict scores as we had to define a political opinion for every candidates in order to know if he is from right or left etc…

Given these thoughts, we decided to focus on the 1st turn of 2017 election and to try to predict the score of every candidate based on socio-economic data. More precisely, we believe that the results are linked to the economic and social situation and that is the relation we are trying to show. In terms geographical splitting, election scores are given per cantons while social and economic data is given by cities/towns and that's why we have to aggregate data to get the same splitting for data and results.

The goal of this project it to be able to predict the results of some cantons given their social and economic data with algorithms trained on the other part of cantons. However, the ultimate goal could be the prediction of 2022 election. Indeed, if we could collect data for years 2021 and 2022 before 2022 election, we could train our algorithms on 2017 election and then predict the results for 2022.

Related Work

Nowadays most predictions use polling institute which means they make a survey among population and then they try to extrapolate those result to the entire population. However those institute have several breaks, for instance their policy of recruitment of surveyed people is biased: they select people thanks to questions via internet but they don't know the identity of the person so some people lie to be selected and then paid. Furthermore, the size of the sample is to be blamed too. Thus another approach of the problem is to be considered. We saw on Le Point [7] that students in Telecom were able to predict an election thanks to the analyze of tweets and google research. Thereby we decided to implement our own algorithm, however, instead of measuring "buzz" around each candidate, we decided to use governmental open data bases which give information on every town across France. We were then able to implement our predictive algorithm like we will see subsequently.

Methodology

1) Data Collection

The work first started with the data collection. All the data we used can be found on internet and is free to use (see the references at the end of the report). First, we collected some data about the first turn of the 2017 election[1] with each candidate's result in every "canton electoral", which is basically a big city or a zone grouping several villages and towns. We also needed some socio-economic and demographic data, which is why we collected two datasets[2],[3] containing a large number of such data for every town in France. Finally, we also needed to know what "canton" each town belonged to, so we also found a dataset about that[4].

All the data engineering code can be found in the python file *"data_engineering.py".*

The first thing we needed to do was to separate the election data concerning each candidate, because at the end, we wanted to train models separately for each candidate. We only chose to work with the main 5 candidates of the 2017 election, because they represent the main 5 political parties or movements in France. At the end of this step, we had 5 datasets (one for each candidate) containing for each canton (described by the 'Département' and 'Code Canton' fields) the proportion of votes for the candidate during the first turn of the 2017 election.

Unfortunately, the socio-economic and demographic data we had wasn't gathered in "cantons" but in towns, which is why we had to aggregate all those features into "cantons". First, we needed to know to which cantons all the towns belonged, so we used a dataset giving us that information. Then we could aggregate the two features datasets into "cantons", looking precisely for each feature if we had to take the sum, the average or the maximum of the values from the towns inside the "canton". We decided to fill the NaN values with the average value of the same column's values, and to normalize between 0 and 1 all the features' values. Finally, we could join these 2 "canton"-aggregated features datasets to the five candidates' datasets in order to have five ready-to-use datasets.

Note that in all the above described process, we had several problems because all the datasets don't describe "cantons" or towns or "Départements" the same way. To make our lifes easier, we used pandas[5], a library allowing to manipulate data in an easier way.

2) Feature Selection

In the final 5 datasets, we had approximately 100 features. If we kept them all, our models would overfit, so we had to use a feature selection method. We decided to use a linear filter approach called f_regression[6]. In *data_engineering.py*, we coded a function that, given an int k, finds a list with the union of the k best features for each candidate

according to that method (we take the union in order to have the same features for every candidate) and returns the 5 datasets with only those features.

The conclusion of all this data_engineering process is that it allows us to export 5 nice and proper csv files, one for each candidate, with the best selected features. Of course, one can change the number of features one wants to select.

3) Linear regression

Algorithm

The linear regression may be the most classic algorithm of machine learning, or at least the most relevant for a first approach. The data were already normalized in pre-processing, but not centered, so there is an intercept. Thus, we added a continuous current component, that is too say a parameter which doesn't depend on any feature. It appears in the matrix of our data.

We used the scikit-learn function "Linear Regression" in the module "linear_model" [8].

We used the cross validation to train the algorithm, thanks to the scikit-learn function "KFold" of the module "model_selection" [9]. The number of folds is a parameter of our algorithm, so it can be easily modified. The number chosen in often between 2 and 10, so we made a plot to find the better one. The better algorithm seemed to be with 4 folds, with no huge difference, so we choose 4 folds for the cross validation.

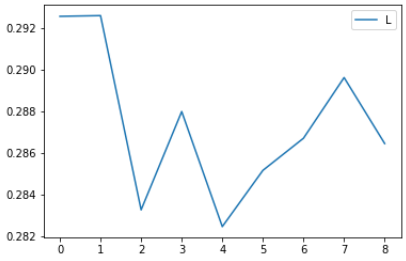


Figure 1: Precision for different number of folds in cross validation (data with Le Pen)

Evaluation

We decided to evaluate this regression with the average relative absolute difference between prediction and real value. Explicitly, for a data (x,y), let's note $\hat{y}$ the prediction made, the error is calculated with $d = |\hat{y}-y|/y$. Then we simply compute the average value of d over the whole test data (which is in fact the part of our data isolated by cross validation). So, our goal is to minimize this value.

Results

We get the following results for each candidate :

Candidate	Le Pen	Macron	Mélenchon	Fillon	Hamon
Avg relative error	28%	13%	17%	18%	23%

4) Neural network

Parameters

We implemented a neural network with the python module keras. We decided to use the neural network as a regressor. Then, we had an output layer with a single node giving the value of the regression. Besides, the input layer had 42 nodes : one for each feature.

We had many parameters to adjust such as the activation functions, the number of layers and the size of the layers.

We chose to use a single hidden layers. Indeed, we realized many experimentations and we finally understood that one hidden layer was enough. We also read a few articles that were suggesting such a choice.

Then most articles state that the size of the layer must be set between the number of output nodes and the number of input nodes. In our algorithm, we had to pick a number between 1 and 42. We tried different values and remarked that was not decisive so we finally picked 30. Indeed, our results were still very similar if that number dropped to 3.

For the activation functions, ML articles are unanimous to say that we better use ReLU function for the hidden layer and the linear function for the output layer.

Finally, we had to set the epochs, ie the number of iteration made on the data to train the algorithm. For this parameter, we couldn't find any real rule of the thumb on internet. So we tested many different values between 5 and 500. Unfortunately, there were no big difference and we chose 50 that seemed average.

Evaluation

As we picked a regression use of neural network, we decided to evaluate it the same way than the linear regression : the average relative absolute difference between prediction and real value. It enables to compare the performance of both algorithms.

Results

Then we had 5 different candidates to test our algorithm. We have to confess that the results are not very precise and we could imagine that because election scores don’t only depend on social and economic data but also on candidates speech, candidates charisma, etc…

In addition, as the optimizer function is the Stochastic Gradient Descent, the results we get can change a little for an iteration to another that's why we will give stats that compile 10 iteration of the algorithm for every candidate.

We obtained the following results :

Candidate	Le Pen	Macron	Mélenchon	Fillon	Hamon
Avg relative error	39%	17%	21%	21%	27%

It's clear that, even if it's more complex, our neural network algorithm is less performant than our linear regression, and it's the case for every candidate. The data may be less adapted for neural network. We also could have made different feature selection depending on the algorithm.

5) Nearest neighbours

Algorithm

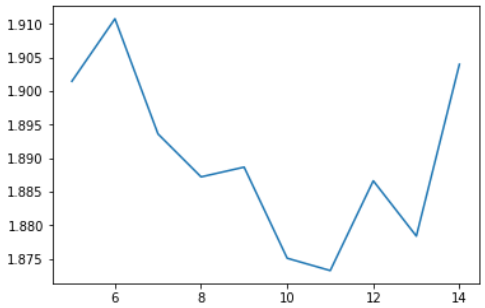
The nearest neighbours algorithm is a classification algorithm, so we need to partition the spectrum of the reachable percentages :

- under 30%, we made intervals of 3%
- between 30% and 50%, we made interval of 4%
- between 50% and 70%, we made interval of 5%
- between 70% and 100%, we made interval of 10%

The aim of the algorithm is to have as many results as possible which are in the same interval.

We used the scikit-learn function KNeighborsClassifier from the module neighbors [10]. The main parameter to choose is the number of neighbours, also called *k*. To choose it, we plot the scores with different parameters k from 2 to 100. By reducing the scale gradually, we found that the optimal number of neighbours was 12, but with no huge difference with the other numbers.

Figure 2 : Score for different numbers of neighbours (data with Macron)



We also used the same cross validation than before to train the algorithm.

Evaluation

To evaluate the algorithm, we used a special score adapted to classification : the absolute value of the difference of intervals. It means that if the result is in the correct interval the score is 0, if it's in the adjacent interval the score is 1, etc.

Results

Candidate	Le Pen	Macron	Mélenchon	Fillon	Hamon
Score	1,47	0,91	1,06	1,30	0,38

These scores are very good. We also determine the percentage of results which are in the good interval or in the adjacent interval, so as to have a better idea of the repartition :

Candidate	Le Pen	Macron	Mélenchon	Fillon	Hamon
Score	60%	79%	75%	68%	96%

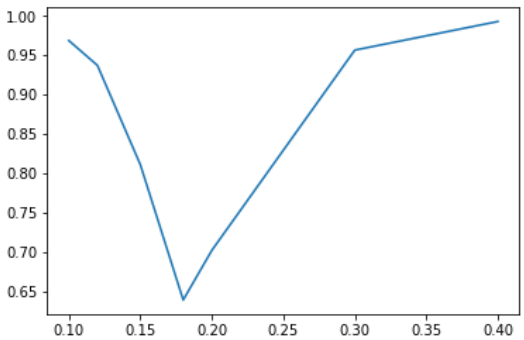
This algorithm is quite powerful. However, it's hard to compare it to other algorithms, since it's a classification and note a regression.

6) Tree and AdaBoost algorithms

The tree algorithm is not the most relevant algorithm to solve our problem, however it is simple to implement for a bidimensional issue. Thus, it is possible to apply it to solve the question "let $n \epsilon [0,1]$, will the percentage of people voting for candidate C in a given town be bigger or smaller than n".

Obviously, the accuracy heavily depends on the n number (it's not difficult to predict that less than 100% of the town will vote for the same candidate but far more difficult when n=0.2 for instance!). Indeed, when we apply the precedent algorithm for different values of n, we obtain the following curb with accuracy in function of n:

Figure 3: Accuracy=f(n)



Thus, it seems that the critical value of n which is the most difficult value to predict is around 0.18.

Tree algorithm is a weak learner (slightly better than random guessing) thus we decided to implement an AdaBoost algorithm to improve it. As we can see on the following curb, test error is decreasing when over the number of trees used. Thanks to this algorithm we have a first insight of possible results.

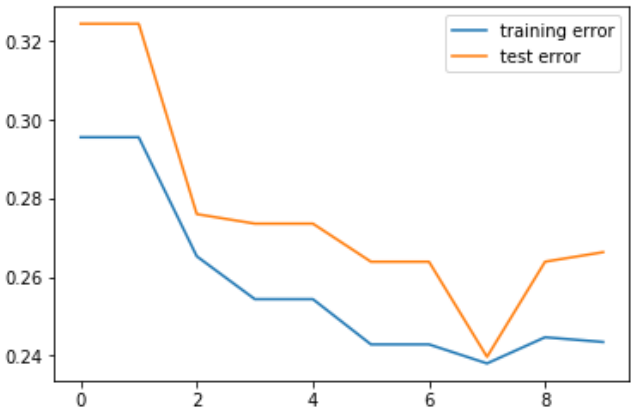


Figure 4: Training and test error for 10 estimators

Evaluation

We were able to implement algorithms with high accuracy and our predictions were in large part true however there is a lack of precision we weren't able to deal with. In fact the 2017 presidential election was particularly difficult to predict because of the “La république en marche” movement. This election was out of ordinary, the Macron movement that came as a new and young alternative to the binary system we had before and our datas don’t take into account things like this. Even if we are able thanks to government open data base to get access to a large quantities of data and thus to evaluate people’s political tendencies, it is very difficult to take into account the overall fed up that characterised this election which mostly explains the differences between predictions and reality. In addition, we were not able to cope with sudden opinion change such as Fillon judicial issues or Hamon sudden drop in the surveys a few weeks before the election.

Finally, we still succeed in predicting global trends based on social and economic data even if we understood that these fields were not completely determining the results of the election.

We could also have adapted the features to the different algorithms. Indeed, some features are more appropriated sometimes, and not other times. It could be a good first point to improve.

Conclusion

The evaluation leads us to select the linear regression as the most performant regression algorithm to predict results of the French election. The score is 0.13% for Macron, 0.18% for Fillon, 0.30% for Le Pen, 0.22% for Hamon, 0.17% for Mélenchon.

The discrepancies between the result of the candidates are often distant enough to be able to class them. For example, the two best candidates can be distinguished only if they have a gap of about 4 or 5% between their score. But it's more difficult to class 2 candidates who have very tight scores.

To use this algorithm, we have to quantify all the selected features describing the cantons in our database, at the year 2022. Sometimes it's impossible but computing them at the year 2021 can be a good approach: the more precisely we study every feature, the more precise our results will be.

Then, we have to run the algorithm with the new features, and we get results for each candidate. Of course, the candidates are not the same from one election to another, but they represent political parties or fields of the political spectrum. Mélenchon can be assimilated to the extreme left wing, Hamon to the left wing, Macron to the center wing, Fillon to the right wing, and Le Pen to the extreme right wing.

Knowing the results of an election is a huge advantage for a candidate, because he can adapt his election campaign, his budget, thus it's powerful information. Another challenge would be to predict the unexpected scandals or the facts which disturb results of the election. Of course, it's far more complex but it would perfectly complete our research and improve the performance.

References

[1]: Ministère de l'intérieur. Election présidentielle des 23 avril et 7 mai 2017 - Résultats définitifs du 1er tour. https://www.data.gouv.fr/fr/datasets/election-presidentielle-des-23-avril-et-7-mai-2017-resultats-definitifs-du-1er-tour-1/. Visited December 19 2018

[2]: Stéphane Chauvin. Data INSEE sur les communes. https://www.data.gouv.fr/fr/datasets/data-insee-sur-les-communes/. Visited December 21 2018

[3]: INSEE. Comparateur de territoire. https://www.insee.fr/fr/statistiques/1405599?geo=EPCI-249300088. Visited December 21 2018

[4]: Ministère de l'intérieur. Circonscriptions législatives : Table de correspondance des communes et des cantons pour les élections législatives de 2012 et sa mise à jour pour les élections législatives 2017. https://www.data.gouv.fr/fr/datasets/circonscriptions-legislatives-table-de-correspondance-des-communes-et-des-cantons-pour-les-elections-legislatives-de-2012-et-sa-mise-a-jour-pour-les-elections-legislatives-2017/. Visited December 22 2018

[5]: Pandas. Python Data Analysis Library. https://pandas.pydata.org/. Visited December 19 2018

[6]: Scikit Learn. Feature Selection. https://scikit-learn.org/stable/modules/feature_selection.html. Visited December 28 2018

[7] PREDICT THE PRESIDENT, prédiction d'une élection

http://hashtagmonde.com/2017/06/08/predict-the-parliament-prediction-des-resultats-des-elections-legislatives/

[8] Scikit Learn. Linear Regression. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression.fit Visited January 02 2018

[9] Scikit Learn. KFold Cross Validation. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

[10] Scikit Learn. KNeighborsClassifier

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html