Modeling 2017 french presidential election results using machine learning on INSEE databases
-
Project Report

Gaël de Léséleuc de Kérouara
gael.de-leseleuc@student-cs.fr

Antonin Duval
antonin.duval@student-cs.fr

Louis Martin
louis.martin2@student-cs.fr

ABSTRACT

We aim at modelling 2017 french presidential election by using machine learning algorithms on different subsets of the INSEE databases. Our goal is to answer the following questions : how accurately can we predict the election results in a particular town, based on the several statistics the INSEE have produced ? Which features have the greater impact on the results or on the turnout ? Can we infer the features our model lacks by studying places where our prediction is less accurate ? In a more general way, we want to know if socio-economic factors are by themselves sufficient to predict the result of an election.

1 INTRODUCTION

Context. The 2017 french presidential election took place in a very complicated context. In June 2016, happened one of the most unexpected situation in European politics, 51.9 % of the British people decided that leaving the EU would be a beneficial option for their country. Nobody would have thought this could happen but 5 months later another news shook our world. Despite every polls being against him, Donald John Trump was elected President of the US on November 8. Moreover, France itself is facing its share of the challenges : Europe is enduring a major migratory crisis, the unemployment rate is still high, as is the public debt, the state of emergency that was declared following the November 2015 Paris attacks is still ongoing, ecological issues are still to be discussed...Considering the problem's scope, french decided that a reshuffle of the political class was necessary, and what we saw was exactly as imagined. The main french favoured candidates disappeared from the lists, for the first time the sitting president Francois Hollande decided not to run as candidate, Nicolas Sarkozy, Manuel Valls, Alain Juppé, Arnaud Montebourg stepped down in favor of newcomers such as Francois Fillon, Marine Le Pen and Emmanuel Macron. In spite of all the unknowns, some people had foreseen the election of Emmanuel Macron such as Jaques Attali a famous french economist [**?**]

Problem. The problem we are trying to solve in the context of this election is to understand if it is possible for us to have a prediction as precise as the polls we can see in figure 1 above using machine learning techniques.[**?**]
To achieve this goal, we have formulated the following questions that our work will try to answer:

- What is the best R^2 accuracy we can get on the first round of the election and what model can provide it?
- Which are the most relevant features for a first round prediction ?
- What information can we infer from our statistics' outliers ?
- How many cities is it necessary to study in order to get the best possible prediction for the rest of them ?

Tools. Our work will rely on the following data sources.

- The first one comes from the DataGouv website [3] it provides us the election results for each town.
- The second dataset contains all the statistical data on each french cities. All the data were queried on a specific INSEE website[4].

This second database can be decomposed in different sub-categories :

- Demography : it contains information about how many people are in each city, how the population fans out into the different age categories, how many people live in a household...
- Education : it contains information about what the level of education is in the city, what is the share of people who have a bachelor degree, how many don't have any diploma...
- Enterprise : it contains information about how many enterprises have been created, how the enterprises are distributed into the different industrial or tertiary sectors...
- Equipment : it contains information about all the different businesses in the city, for example the number of bakeries or doctors...
- Real Estate : it contains information about whether people own or rent their home and if it is a principal residence or not...
- Work : it contains information about the employment rate in a city and what kind of work people mostly practice in the town.

For this work we have a sufficient amount of data with more than 34 000 towns at our disposal.

2 DEFINITION

2.1 Dataset notation

Let us consider all the cities in France (roughly 34 000). Each city will be written as c_i. Those cities are characterized by a set of m numerical features.
For a city c, we define the city's features as $\theta_1(c), \theta_2(c), ..., \theta_m(c)$.

For the first round of the election, the result of candidate $cand_i$ in city c is denoted as $r^{1rst}_{cand_i}(c)$ where i is between 1 and 11 (there

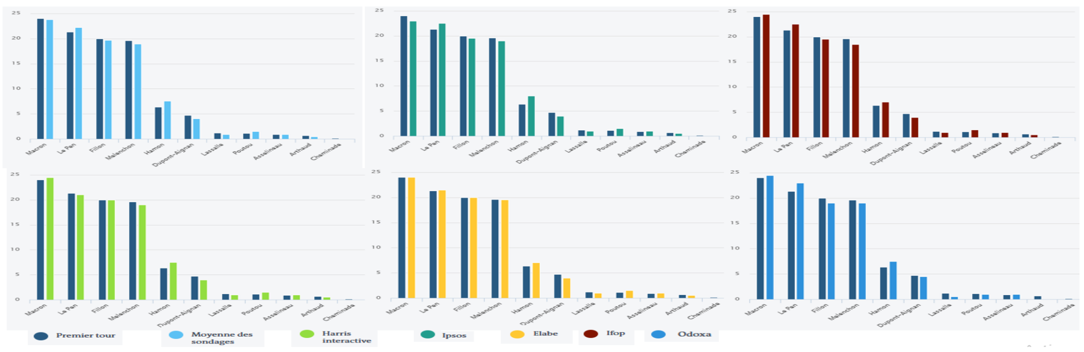


Figure 1: Results of 5 polls compared to the real results
source : Commission des sondages & Ministère de l'intérieur

was 11 candidates in the first round). Candidate's result are conventionally described by a percentage, so $r^{1rst}_{cand_i}(c) \in [0,1]$ and $\sum_{cand} r^{1rst}_{cand}(c) = 1$.

For the second round, the result of a candidate is denoted as $r^{2nd}_{cand_i}(c)$ with now i = 1 (LE PEN) or 2 (MACRON). Let's also introduce a second notation : let $w(c)$ be the winner of second round in city c.

2.2 Problem formulation

We will train our different algorithms on a subset of the cities (typically 50%) and try to make a prediction on the rest of the cities. For that we will try to do two separate tasks:

- Classification
- Discrete probability distribution regression

Classification. Considering that a majority of candidates did not win in any cities in the first round, we will only use this classification method on the second round.
Considering a city c, we want to predicts the winner $w(c)$. To do so, we will predict the probability that "Le PEN" win knowing $\theta_1(c)$, $\theta_2(c)$, ..., $\theta_m(c)$, then we will use the cross-entropy loss $\mathcal{L}$described as such :

$$\mathcal{L}(p, winner) = \begin{cases} log(p) & \text{if } winner = \text{LE PEN} \\ log(1-p), & \text{otherwise} \end{cases} \quad (1)$$

And in order to evaluate the model, we will simply compute the accuracy on the test set.

Discrete probability distribution regression. This method can be implemented for both first and second round. For each city c, we compute the resulting distributions as :

- $r^{1rst}_{cand_1}(c)$, .., $r^{1rst}_{cand_{11}}(c)$ for the first round
- $r^{2nd}_{cand_1}(c)$, $r^{2nd}_{cand_2}(c)$ for the second round.

To train the model, using the regular "hard" cross-enctropy loss was no longer an option, we had to define another loss function. Let *pred* be the predicted result distribution ans *true* the true result distribution. With this notation, in first turn, the loss is :

$$\mathcal{L}(pred, true) = \sum_{i=1}^{11} true_{cand_i} * log(pred_{cand_i}) \quad (2)$$

This loss function is sometime called the "soft" cross-entropy loss in the literature but it is the formal definition of the cross-entropy between the "true result" distribution and "predicted result" distribution.
For our algorithms, we had to program it ourselves since machine learning frameworks such as scikit-learn only computes the "hard" cross-entropy loss.

To evaluate the model, we have used coefficient of determination. for instance, to evaluate model predictions for second turn, we will simply compute $R^2_{\text{LE PEN}}$.
For the first turn, the situation is more delicate because we have a prediction with multiple outputs. So to end up with a single numeric metrics, we have decided to simply take the mean of each candidate's coefficient of determination :

$$R^2 = \frac{1}{11} \sum_{i=1}^{11} R^2_{cand_i} \quad (3)$$

2.3 Further discussion on loss functions and evaluation metrics

Note that in our two evaluation metrics : accuracy and mean of coefficients of determination do not take into account the population of each city. This is reflected in both loss functions : the population feature do not appear in none of these two losses. As we shall see later, this choice is undoubtedly not without consequences. To push things a little bit further, let explicit and compare two type of predictions close but still different :

(1) taking randomly a city from all the cities, predict the result of a candidate
(2) taking randomly a french citizen and only knowing in which city he lives, predict the probability that he votes for a particular candidate.

The first type of prediction is what we are doing with losses 1 and 2. As for the second type of prediction which is very similar to the first one, we have to take into account a prior probability distribution corresponding to the probability for a random citizen to be picked from a particular town. This prior distribution forces the algorithm to take better notice of the population of each city. Another solution would have been to put a weight on each cities depending on its population into the loss.

3 RELATED WORKS

Trying to predict the outcome of an election is a decades-old tradition. However, these predictions are the results of opinion polls made on a sample of individuals as seen on figure 1. What we want to try instead is focusing only on socio-economic factors to predict the outcome.

Different people have already tried to implement machine learning on this subject. In 2017, a student tried to predict the result by counties in the 2016 American election using machine learning techniques[5]. However, his analysis was only based on demographic data and was focusing on the last turn of the election.

Michel Blum, a Data Scientist at the CNR, published an article on how he used machine learning with socio-economic data to find outlier cities[1]. Outlier cities are cities where results are a real no-match with the prediction, meaning those conurbations don't follow common trend and are interesting to be looked at. He focused on the result of Marine Le Pen in the second round of the election. His work is close from what we want to achieve, since he used the same source of datasets. However, our focus is on the first round of the election, and we want to predict the total distribution of vote in every town for every candidate.

One interesting approach was made by researchers at the Stanford Universtiy[2] where they made prediction at the individual level. Using personal public information (gender, age...), they tried to predict the probability that an individual person would vote for Clinton in the 2016 U.S. elections. They managed to achieve pretty good accuracy. This might be due to the fact that they relied on covariates such as "is a member of the Republicain party", etc. Since we can't access to these type of personal information, we decided to work at a superior level, using aggregated data at the city level.

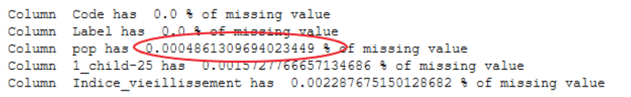
4 METHODOLOGY

4.1 Data Cleaning

We first imported a lot of data from the INSEE database and we worked on removing all the "Nan" values. We studied how much removing them would influence our database. For example, when checking for missing value in our database we reached the conclusion that either this town belonged to Mayotte where the social situation might sometimes be a little difficult to evaluate or this town was destroyed during the Second World War. In both case, we took the liberty of removing these cities from the database. Furthermore, we also had the situation where the problematic city was neither in Mayotte nor destroyed but was only a small town (< 10 inhabitants) in this case we verified that the number of occurrences was marginal compared to the size of the database and proceeded to remove these cities as well.

The second strategy we implemented is regrouping the data in category, in particular for population.

Besides, in order to merge all the data in a single dataframe, we produce for each town a INSEE code which results from the merge



```
Column  Code has  0.0 % of missing value
Column  Label has  0.0 % of missing value
Column  pop has  0.0004861309694023449 % of missing value
Column  1_child-25 has  0.0015727766657134686 % of missing value
Column  Indice_vieillissement has  0.002287675150128682 % of missing value
```

Figure 2: We can see in this figure that a really low proportion of our data is missing and that removing them is not important. In this case, it sums up to 65 cities

population	category
$0 < p < 2000$	Village
$2000 < p < 5000$	Borough
$5000 < p < 20000$	Small Town
$20000 < p < 50000$	Middle Town
$50000 < p < 200000$	Big Town
$200000 < p$	Metropolis

Table 1: Categorize cities based on their population

of the communal and departmental code.
After cleaning all our data we tried several algorithms to make our predictions.

4.2 Prediction techniques

For a first approach on this dataset, we decided to try to predict the winner in each town in the second round of the election. This is a classification task where our target vector is composed of 0 and 1, where 1 indicates that Marine Le Pen had more voters than Emmanuel Macron in this town. What we wanted to see is if it's possible for an algorithm to learn how the socio-economical variables play a role in the final turn of the election. What we expected to see is that it will give many importance in features like the size of the population, or the ratio of diploma. Indeed, many analysis after the presidential campaign showed how Mr Macron and Mrs Le Pen have a very different basis of electors. The one big downside of this approach is that, by converting the number of voters in 0 and 1, we lose a lot of granularity. For example, a town that is very undecided between Macron and Le Pen, which had a final score of 51% vs 49% will pose problem for the algorithm, since this indecision will not be transcripted in the binary vector.

We decided for this task to use different type of classifier :

- A Logistic regression
- A Random Forrest
- A Gradient Boosting
- A SVM

To compare the result of those different algorithms, we shall use the global accuracy.

4.3 Candidate result prediction

To predict the distribution of results in an particular city, we have to figure out our own working pipeline. Indeed, this problem is neither a classification problem, nor a lambda regression problem. Let us analyze why we can not apply directly this method to our problems.

(1) **Classical classification algorithms** (as implemented in scikit for instance) learned by comparing predicted output to one-hot distribution by using cross-entropy loss. We could have used such methods with soft-label : a vector of probability instead of one-hot. Unfortunately, machine learning frameworks does not accept such label for classification
(2) **Classical regression algorithms** can predict a numerical vectors. However, it can't ensure that we end up with a normalized vector (which is mandatory for our cases).

So to overcome those limitations, we try two approaches.

Neural networks. In a first approach, we try to directly predict a discrete probability distribution by using a neural network with a number of output neurons equal to number of candidates and then normalize the output results before apply loss functions (soft-cross-entropy loss described in **equation 2**). This pipeline is describe in **figure 3**.

- **Pro** : directly produce a discrete probability distribution as an output
- **Con** : with this pipeline, will the neural network be able to learn anything from the small candidates (which always have low result).

To implement this neural network we used pytorch.
We still have to discuss the way we apply normalization. Usually, in classification algorithms, we apply a softmax to the output : it differentiable and produce a probability distribution. In that case, each output neuron is normalized as :

$$pred_i = \frac{e^{output_i}}{\sum_{i=j}^{11} e^{output_j}} \tag{4}$$

However, it exponentially amplified the neuron output which has the largest values. This is good for classification, but question is : is it good for our purpose ? Because in our case, different index of the true distribution result may have similar values. So will our neural network be able to produce this values when using a softmax layer ? To study this effect, we will compare softmax normalization with simple normalization :

$$pred_i = \frac{|output_i|}{\sum_{i=j}^{11} |output_j|} \tag{5}$$

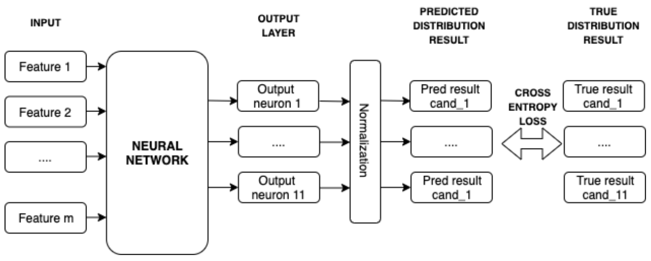


Figure 3: Neural network to predict a discrete probability distribution

One versus rest. In a second approach, we try to use multiples model (as much as there are candidates). Each model predict for each city the score of only one candidate. Then we merge the results of the 11 models and normalize it to have a true distribution results.

- **Pro** : we can used many existing regression algorithms and normally we should be able to do good predictions even for "small" candidates
- **Con** : now, we don't produce a probability distribution : we have to normalize it at the end. So if the output vector norm is far from one, we will probably lose a lot of prediction quality in normalization step. The pipeline for this method is describe in **figure 4**.

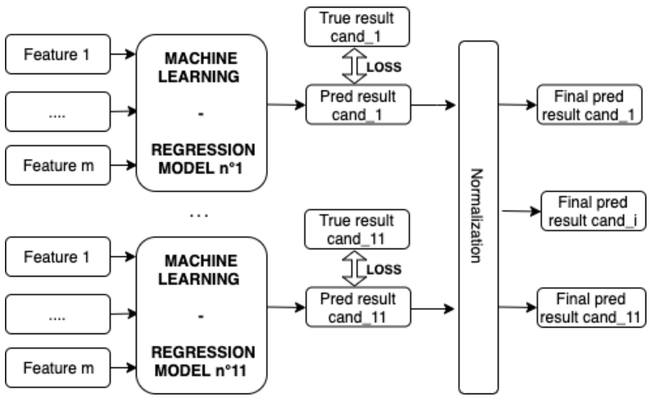


Figure 4: One versus rest approach to predict a discrete probability distribution

4.4 Analysis of feature's importance for each candidates

Good thing of using simple model like logistic regression is that we can easily look at the weight of the model to understand why it made a particular decision. So our idea was that once we trained logistic regression to predict the score of a particular candidate, we can retrieve all the weight, with each weight being the importance a model give to a specific feature when it have to predict the score of a particular candidate. Once we have retrieve all the weights, we can easily sort them by their absolute value and print the largest one.

4.5 Candidates embedding

Our final idea was to study if, when using machine learning to model election results, the algorithm learn anything about the candidates themselves. For instance, is it possible to compute a similarity measure between the candidates.

To do so, we look for a way to embedded the 11 candidates from first round into as many vectors such as the distance between candidates reflect their political affinities. We used the following method to create such an embedding :

(1) for each $cand_i$, train a logistic regression $model_i$ to predict $r^{1rst}_{cand_i}(c)$ for each city c

(2) retrieve $w^i = (w_1^i, .., w_j^i, .., w_m^i)$ the weight vector of each model. This mean that w_j^i is the weight that $model_i$ give to features θ_j.
(3) associate $cand_i$ to vector w^i so that $w = \overrightarrow{cand_i}$

Once we have embedded the candidates in a m (= number of initial features) dimension vector, we will be able to visualize the candidates embedding in a 2D-space and check if distance between candidate embedding reflect our common representation of political space. For instance, we would like to check if we have $\overrightarrow{FILLON}$ closer $\overrightarrow{MACRON}$ than to $\overrightarrow{HAMON}$) because in french political left-right axes, FILLON is more right, HAMON more left and MACRON in the middle. If the logistic regression model we trained are good enough, meaning than we can predict result each candidates will get depending on the city features with enough accuracy, this is something we expect to get as a result because this means than the weight of the models has succeed to learn the important features of each candidate.

5 EVALUATION

As our motivations were multiple, we aimed at firstly targeting the simpler objectives. As so, we will begin by predict the election winner, then try to obtain the election distribution results. In amount of this two works, we will study the different features.

5.1 Winner prediction

We tried predicting the winner of the election in the second round for each town. Using different classification algorithms, we used a gridsearch on multiple folds to tune our models. We selected a subset of our dataset to only have cities with more than 500 inhabitants. By doing so, we remove outliers that are very often small villages : their behaviour is very hard to predict, so we focused on larger cities. After selecting the best hyperparameters for each models, this is all the ROC curve we get.

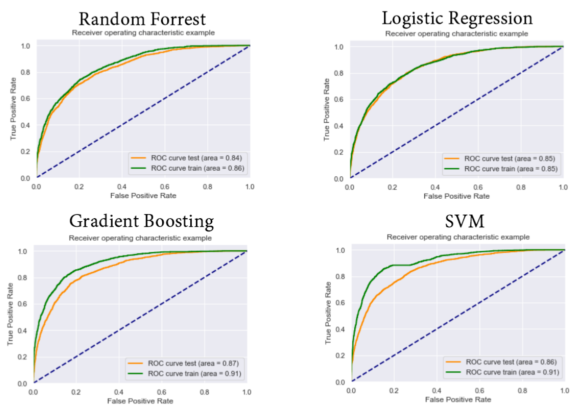


Figure 5: ROC curve for the task of predicting the winner in the second round

Overall, it seems all 4 algorithms we tried gives similar result with gradient boosting being the one with the most accuracy.

Model	**Accuracy**
Logistic Regression	0.835
Random Forrest	0.801
Gradient Boosting Trees	0.847
SVM	0.840

5.2 Distribution regression

Second round. To predict second round results, we first implement a very simple algorithm but logistic regression with soft-cross-entropy loss. To visualize what our algorithm has learned, we plot the true score of Marine Le PEN vs the score our logistic regression. To make the plot easier to interpret, we used two tricks :

(1) colorize and choose the size of each city dot depending on their population (see table 1).
(2) first plot the small city and after plot above the biggest city

By using this strategy, we obtain figure **??** and can do the following analysis.

(1) this simple model succeed to learn something ($R^2 = 0.372$)
(2) the smallest cities are more spread. This was excepted because, some city only have few inhabitants. As a consequence the variance of INSEE statistics on these cityes are really high and not trustful, and it hard to produce any predictions on it.
(3) the model over-estimate Marine LE PEN score for big cities. This phenomena is probably due to two things : biggest cities has voted more MACRON than smallest cities and as we mention in section 2.3, we do not take into account the population in the loss function.

If we remove, the small city (let say less than 1000 habs) and retrain our model, we obtain the plot **??**. We directly see a great improvement of our model ($R^2 = 0.53$) and now the prediction on metropolis are way better.

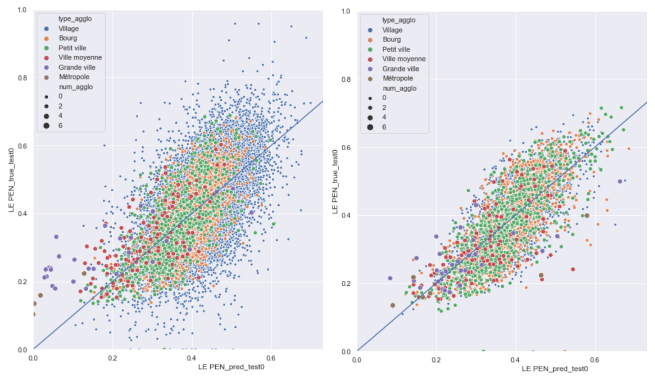


Figure 6: Use of logistic regression to predict LE PEN score.
Left : on all the cities
Right : by keeping only cities which have at least 1000 inhabitants

First round. We first try to predict the first round result by using neural networks with soft-cross entropy loss (pipeline describe in **figure 3**). We compare three models :

(1) *2Layers_no_softmax* : a 2 layers network with 200 neurons in the hidden layer with custom normalization at the end (see **equation 5**)
(2) *2Layers_w_softmax* : a 2 layers network with 200 neurons in the hidden layer with softmax at the end)
(3) *1Layer_w_softmax* : a single layer network with softmax at the end - so simply generalization of logistic regression to multiple dimensions.

We train this model in parallel, on the same batch (size = 36) and with the same loss function on 50% of the citys which contains at least 500 inhabitants. The models were trained over 100 epochs. To compare these models, we used our custom metrics - R^2 mean define in **equation 3**. All the results are compiled in **table 2**

MODEL	2Layers no_softmax		2Layers w_softmax		1Layer w_softmax	
R2 score	**train**	**test**	**train**	**test**	**train**	**test**
mean	0.02	-0.01	0.39	0.16	0.19	0.19
HAMON	0.12	0.08	0.48	0.18	0.2	0.21
DUPONT-AIGNAN	0.13	0.08	0.39	0.18	0.24	0.23
LASSALLE	0.13	0.09	0.63	0.22	0.21	0.25
ARTHAUD	-0.03	-0.06	0.2	0.07	0.15	0.15
POUTOU	-0.01	-0.04	0.22	0.03	0.1	0.09
ASSELINEAU	-0.02	-0.05	0.1	-0.04	0.03	0.02
MACRON	0.24	0.2	0.63	0.44	0.43	0.43
MÉLENCHON	-0.24	-0.27	0.57	0.19	0.22	0.22
FILLON	-0.07	-0.08	0.66	0.39	0.33	0.33
CHEMINADE	-0.38	-0.36	-0.32	-0.38	-0.29	-0.29
LE PEN	0.32	0.28	0.76	0.51	0.5	0.5

Table 2: Comparison of 3 neural networks model to predict first turn result. For each candidates, the R^2 is calculate on test set

By analysing this table, we can formulate several observations:

(1) surprisingly, it is better to use sotfmax normalization. It is seems that more train epochs are needed when using our custom normalization function
(2) by using 2Layers network, we observe a clear over-fitting because we have a big gap between our metrics values on train set (0.39) and on test set (0.16)
(3) as we excepted, model do not learn much from small candidates (it under-fit).

Because, when we try to increase the complexity of our model (when going from 1 layer to 2 layer) we quickly have a ovefitting effect from the "big" candidates but still observe a underfitting effect for the "small" candidates, it seems very unlikely to obtain better results with this pipeline.

Now, let evaluate the prediction results when using the one-versus-rest pipeline describe in **figure 4**. We first train 11 logistic regressions in parallel and concatenate their output to form the discrete probability/results distribution. Then we plot the distribution of these vector norm before normalization. The idea is that we are far from one, the normalization will strongly transform the vector and we will loss a lot of precision. This can for instance occurred in a model strongly over-estimate the score of one particular candidate.

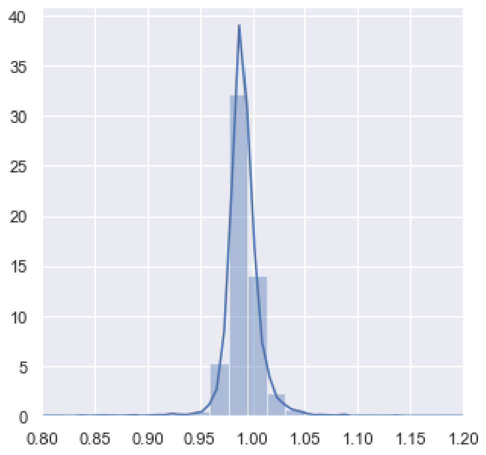


Figure 7: Distribution plot of output vector's norm before normalization in one-versus-rest approach

As you can see in **figure 7**, the norm of the output vectors of one-vs-rest approach are narrow to one.

Since this approach seemed quite successful, we decided to try another model. We wanted to use decision trees to try to predict this distribution, so we used a Gradient Boosting Regressor, which is an ensemble model proposed by the the scikit library. We trained 11 models in parallel, one for each candidate, using different set of parameters. As before, we wanted to maximise the mean R2 score. The result were better than the neural network as we can see in **figure 9**. Parameter D stands for max depth of a tree and T for the total number of estimators. Our Gradient models seemed to overfit quite a lot. However, we achieved a better R2 score on the majority of candidate comparing to the neural network we trained before. After some trials hypertuning the parameters, we didn't not achieved to reduce the variance without reducing the global R2 score.

In **figure 6**, we show the regression plot we obtain with gradient boosting on each candidates (we plot only 10 candidates for visualization purpose). These plot demonstrate that our model succeed to at least something for the 6 biggest candidates. How when it is about small candidates (which has obtain less 2% of national suffrage), we observe a sort of vertical cloud dot in the regression plot. This mean that the only thing our algorithm learn is to predict for each city the mean result that a particular candidate obtain at a national level.

5.3 Features analysis

Obtaining meaningful information about candidates by looking at features weight as explain in **section 4.4** did not work as well as expected. If we looked at the 5 most important features for a particular candidates, we observed is that we had a great variation

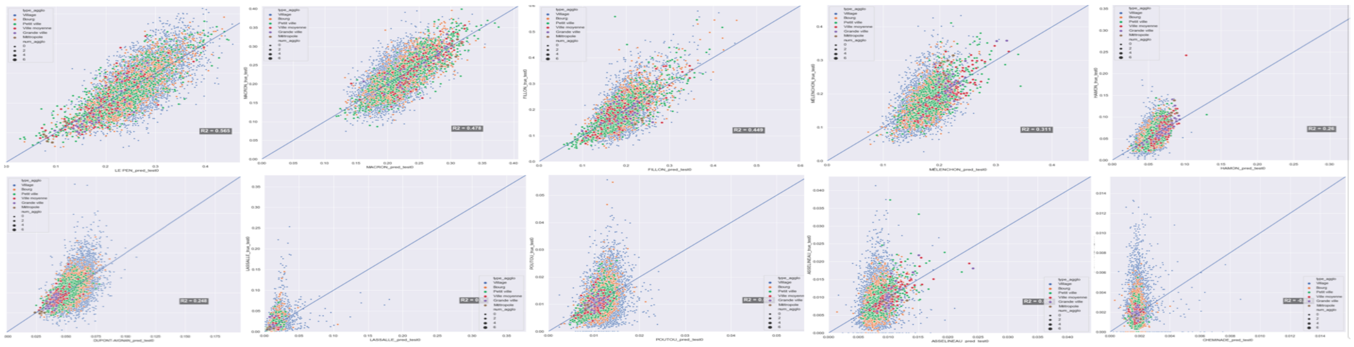


Figure 8: Regression plot for each candidate prediction on test set. From left to right, first row then second row : Le Pen, Macron, Fillon, Mélenchon, Hamon, Dupont-Aignan, Lassalle, Poutou, Chemineau, Asselineau

Model	1Layer w_softmax		Gradient Boosting D=3 T=500	
R2 score	**train**	**test**	**train**	**test**
HAMON	0.2	0.21	0.39	0.26
DUPONT-AIGNAN	0.24	0.23	0.43	0.247
LASSALLE	0.21	0.25	0.51	0.18
ARTHAUD	0.15	0.15	0.38	0.18
POUTOU	0.03	0.02	0.31	0.10
ASSELINEAU	0.43	0.43	0.27	0.05
MACRON	0.43	0.43	0.591	0.477
MÉLENCHON	0.22	0.22	0.46	0.30
FILLON	0.33	0.33	0.57	0.44
CHEMINADE	-0.29	-0.29	0.21	-0.02
LE PEN	0.5	0.5	0.65	0.561
Mean R2	0.18	0.19	0.43	0.253

Table 3: Comparison of the Gradient Boosting approach vs neural network. For each candidates, the R^2 is calculate on test set

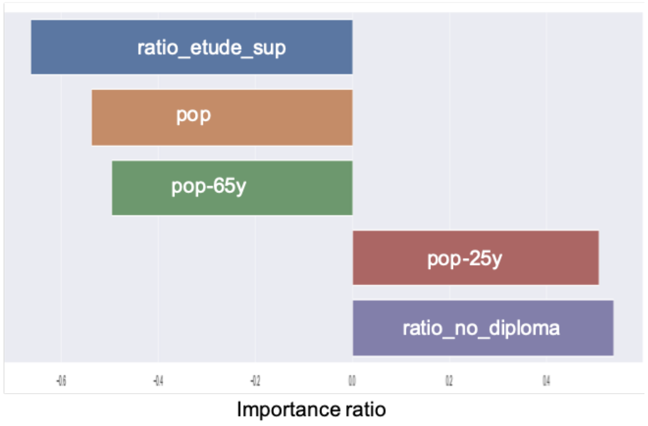


Figure 9: Histogram plot of the 5 largest feature weights in a logistic regression trained to predict Marine Le Pen results in secund turn

depending on the way we train the model or of the number of initial features. However, in secund turn, by retrieve the weight of our classification model, we obtain stable result (show in **figure 9**).

Our conclusion is that even if we can make use of the model we have trained to undestand feature importancy, we could have obtain similar results and more easily by computing simple correlation between the feature and a candidate vote.

5.4 Candidates embedding

To embed into vectors as described in **section 4.5**, we first have to reduce the number of features for two reasons.

First at the end, we want to compare distance between different pairs of candidates embedding so if the dimension of embedding spaces is too large, the distance between each vector will more likely be the same.

Second, some of our initial features are very correlated (for instance "population number" and "number of entreprise" of the cities), this is a big issue for our embedding task because we are training 11 models separately. Let take an example to illustrate this problem. Suppose that we have two candidates which are equally sensitive to the "size" of the city and suppose that "pop" and "nb-ent" are perfectly correlated. Maybe first model will arbitrary give large weight to "pop" features and low weight to "nb-ent" whereas the second model will give the opposite weight distribution to achieve the same effect. This is not the problem when the focus is to predict the candidate results but here it is the just a way to achieve our true goal. To continue with our example, we will then obtain these two embedding vectors :

$$\overrightarrow{cand_1} = (nb\_ent = \alpha, pop = \beta, \ldots)$$

$$\overrightarrow{cand_2} = (nb\_ent = \beta, pop = \alpha, \ldots)$$

Then, if we simply consider the first two dimensions (corresponding to our theorical hypothesis that this two features are perfectly correlated), our two vector will not be aligned as we which.

To address this two problems, we reduce embedding dimensions and remove linear correlation by performing PCA on initial features spaces. Starting from almost 50 features for each city, we use a 20-components PCA which allow us to keep almost 90% of the

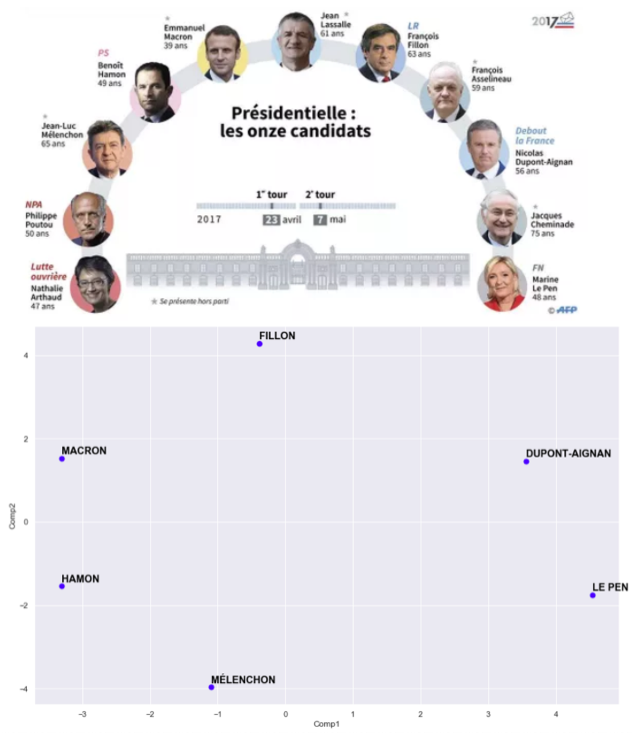


Figure 10:

- **On the top : positioning of each of the eleven candidates on the french political space according to AFP. Here we have a representation goind from extrem-left wing (Arthaud) to extrem-right wing (Le Pen)**
- **On the bottom : 2D visualization of candidates embedding (only for candidates which has succeed to obtain at least 1.5% of the suffrage**

variance ratio. Then we follow our initial protocol describe in **section 4.5** to obtain a 20-dimension vector for each of the 6 "biggest" candidates. We did not keep the other candidates because as we saw in **figure 8**, the algorithms did not succeed to learn anything for these candidates due to the very low amount of vote they succeed to get. As a final step, we re-scale each dimension and perform a final PCA to project our six 20-D vectors in 2D space and be able to visualize them. The 2D scatter plot of this six candidates embedding is show on **figure 10**.

As a comparison, we print above the common representation of the eleven candidates spread on the political space. As you can see, our 2D representation nicely confirm the common representation. As excepted we have FILLON is closer to MACRON than to HAMON, DUPONT-AIGAN is clore to LE PEN than MACRON, etc.

CONCLUSION

REFERENCES

[1] Michael Blum. 2018. Outlier detection using machine learning approaches: application to French politics. https://towardsdatascience.com/machine-learning-approaches-detect-outlier-values-that-do-not-follow-a-common-trend-detecting-cc0252f637bd

[2] Rosenman Evan and Viswanathan Vitin. 2018. Using Poisson Binomial GLMs to Reveal Voter Preferences. *arXiv* 1802.01053 (2018).

[3] Data Gouv. 2019. Les données des élections. https://www.data.gouv.fr/fr/posts/les-donnees-des-elections/

[4] INSEE. [n. d.]. Les statistiques locales. https://statistiques-locales.insee.fr

[5] Ryan Peach. 2017. Describing the 2016 Election with Machine Learning. http://www.ryan-peach.com/school-projects/2017/5/22/describing-the-2016-election-with-machine-learning