Predicting the percentage of people speaking Catalan in municipalities in Catalonia

Enric Reverter
Pim Schoolkate

June 2022

1 Introduction

Catalan language has gone through a lot during the last century, specially during the persecution after the Spanish Civil War. As such, Catalan speakers have been oscillating in numbers thorough generations. The aim of this project is to analyze the number of speakers taking into account contextual variables such as the level of education attained or the number of immigrants at a municipality level. To do so, data from Idescat [3] is used.

There do not seem to be previous studies analyzing the correlation between number of Catalan speakers and other socioeconomic indicators. However, there are other kind of studies such as the usage of Catalan by the context of different areas of personal relations [1]. Other studies can be found at Gencat [2].

2 Data

2.1 Data Description

The data that was used in this project was retrieved from the idescat API, which provides 213 different indicators for each municipality in Catalunya. The indicators are separated in different topics which are "Population", "Culture - Language", "Education", "Elections", "Labour", "Quality of Life", "Main aggregates - Public sector finance", "Economic Sectors", and "Environment. Population includes data on basic demographics like the population size, the age distributions, deaths and births, place of birth, and external and internal migrations. The Culture - Language indicators report the knowledge of Catalan, the existence of libraries, and the amount of sports facilities by type. In Education the percentage of people having attained a certain education can be found. The Elections include information about the turnout (in percentage) of different elections (like municipal, or European). The Labour category denotes information about the unemployment rate, self-employment rate, and the contributors to the Social Security in Spain. Quality of Life contains information about the dwellings in the municipality. Main aggregates reports on the Urban property tax and the personal income tax. The Economic Sectors include information about the type of cultivated land (like fruit trees, olives, etc.), fleet of vehicles, or the amount of tourist accommodations. Lastly, the Environment indicators report municipal and industrial waste.

2.2 Data Retrieval

The data was retrieved by calling the idescat API for each municipality. Because idescat does not handle data structure well, a custom build parser was build, which allowed to parse and store all the data in a `csv` file. To quickly sum up the problem, within the `json` structure provided, sometimes a list and other times a dictionary was present, which required many conditional handling of each separate indicator. Likely the homogeneity of the structure isn't consistent for all indicators because the data is not supposed to be retrieved the way it was done in this project. A second problem with retrieving the data had to do with being kicked out of the API for requesting too much data, suggesting even more that the API was only meant for retrieval of a few indicators. This problem was fixed by batching the requests and introducing a waiting time between each request session.

2.3 Problems of the data

One of the main problems of the data is that the different indicators are from different years. Because many of the indicators relate to the total population, some indicators would report that 140% of the population speak Catalan, as the measurement of Catalan speakers was done in 2011, but the population size was of 2021. This was true for many more indicators, each having their own year of origin. In order to fix this problem, data on the population sizes of each of the municipalities for the year of 2011 was joined with the main data and used to get a percentage of Catalan speakers. Unfortunately, this was not done for the other indicators, due to finding the problem late in the project, and the limited amount of time. Therefore, only the target variable was treated as it is the most important variable. This shortcoming will be discussed in the limitations.

3 Pre-processing

The first thing to notice in the data is that multiple columns have duplicate values although their names are different. This is due to how the API retrieved the data. For example, for the indicator of population by sex it retrieves it for male, female, and total. The same happens with the indicator of population by age, where apart from the different intervals of age it retrieves the total. As such, all columns with duplicate values have been removed. This results in deleting 11 indicators.

Another relevant aspect is that most of the indicators are measured in terms of population. That is, small municipalities have smaller values for most of the indicators while in larger municipalities the opposite is seen. For that reason, most indicators need to be normalized by the municipality total population, which is done in this step.

The data contains many missing values in some of the indicators, which can be depicted in figure 3.1. It is logical that some indicators are not being that much tracked in some municipalities, since some are quite small. It has been considered that features with more than a 30% of missing values are not robust and have been directly dropped from the data. This results in deleting 64 indicators. The number of missing values are counted per row and stored in a feature called `missing_count` to see if outliers are affected by the imputation to be done.

Then, two methods from `scikit-learn` are applied to impute the missing values from numerical variables. First, the data is fit and transformed with `SimpleImputer`, which strategy is set to impute all the null values with the mean of the non-null ones. Second, the data is pipe-lined through `KNNImputer`, which imputes the values with the mean value of the `k` nearest neighbors. It has been set so that closer neighbors are more weighted and thus contribute more to the imputed value. Multiple `k`'s have been tried, as depicted in table 1.

In order to check the validity of the imputation a Kolmogorov-Smirnov test is applied to compare the distributions before and after imputation takes place. Every feature is tested, so if the p-value

Figure 3.1: Histogram of the percentage of missing values.

Method	Average Score	Different Distributions
SimpleImputer	0.952	6
KNNImputer (k=2)	0.975	2
KNNImputer (k=6)	0.971	3
KNNImputer (k=10)	0.967	3
KNNImputer (k=14)	0.965	2
KNNImputer (k=18)	0.962	2

Table 1: Imputation results.

yielded by the test is inferior to the threshold of 0.05 it gets annotated. The number of columns with different distributions can be observed in table 1. It is decided that the best method is `KNNImputer` with 2 neighbors. The columns that are not properly imputed are *elections_municipal_elections_electors* and *quality_of_life_type_of_dwellings_secondary*, which are removed from the data.

Regarding the categorical variables, since the number of missing values are low (38 at most), they are simply labeled as *Unknown*.

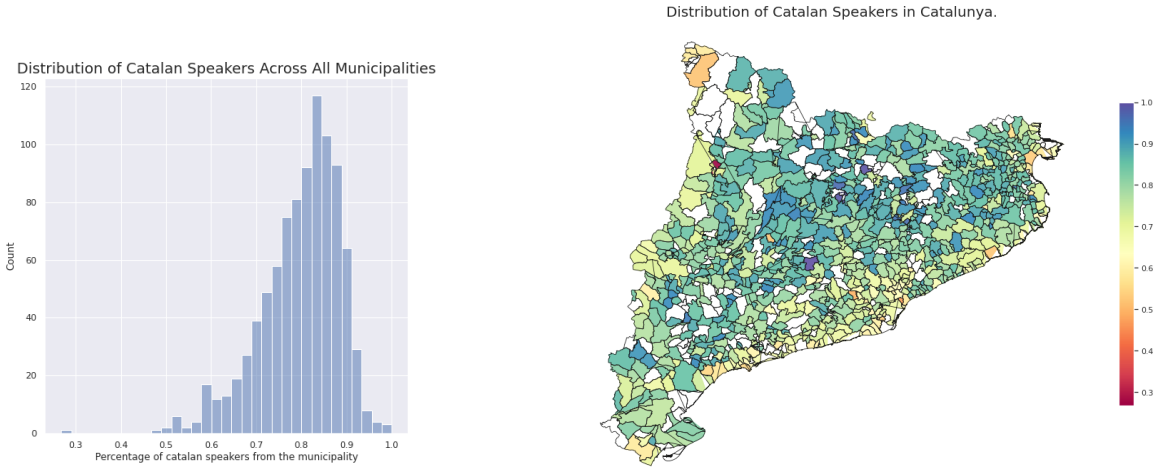
With relation to the outliers, two multivaraite approaches are considered, Mahalanobis distance and `IsolationForest`. Univariate approaches are not applied due to the large number of features. The former method computes the Mahalanobis distance for each data point and then compares it with a Chi-squared distribution to check if it complies with the threshold. The latter detects anomalies by computing recursive partitioning, which produces shortest paths for anomalies. The issue with Mahalanobis distance is that it needs the covariance matrix, which due to the high correlations between the current features yields results very susceptible to the input. As such, it is decided to remove the outliers based in the isolation method. It detects 28 outliers, which have a higher number of missing values per row as depicted in table 2, so imputation might have slightly effected such results. These observations are removed from the data, but they are stored in case a model is trained specifically for them.

Henceforth, data is already pre-processed, but is still needed to choose whether to treat the label as

	Count	Average (NaN's)	STD (NaN's)
Outliers	28	6.8	4
Non-outliers	919	1.5	1.7

Table 2: Outlier detection.

a numerical or a categorical variable. The distribution of the percentage of people that speak Catalan is depicted in figure 3.2. It can be seen how it is skewed to the right as the mean percentage is 0.79. The same distribution can be depicted over Catalunya in the map on the right. It is interesting to see how municipalities near the coast and Aragó tend to have less percentage of Catalan speakers, while municipalities near the heart of Catalunya and the Pirineus tend to concentrate more Catalan speakers. This may be due to the fact that people near Aragó are more exposed to Spanish and also the coast tends to attract more foreigners. Looking at the label to be predicted it has been decided to treat the problem as a classification one. As such, the label is split into three categories: low (left tail based on the first quantile), medium (second and third quantiles), and high (right tail based on the fourth qantile).



(a) I1

(b) I2

Figure 3.2: Dist

4 PCA and clustering

Further exploring the data, it can be observed how most variables are highly correlated as depicted in figure 4.1. For that reason, dimensionality reduction is taken into account. The approach chosen has been principal components analysis (PCA) because most of the variables are numerical. The categorical variables have been encoded into dummies using the `OneHotEncoder` method so that they can be directly used for the PCA. In case of a large amount of categorical features a multiple correspondence analysis could have been applied. Also, in order to use the data in the PCA it needs to be previously scaled and centered using the `StandardScaler` method, otherwise the results could be misleading.

In figure 4.2 the results of the PCA are depicted. Note how after certain amount of components the explained variance does not increase significantly. In the end, 25 components are chosen to reduce the data, which explain roughly 80% of the variance. The reason being that many components were needed to increase the variance from this point onwards. In order to grasp how each component is computed from the original features, they have been grouped by the fields of the indicators: population, territory, culture, etc. This can be observed in figure 4.3. It can be seen how the first component receives the most information from features in: population, elections, and economic. The second describes variables from: population, education, and labour. And so on. It is also worth noting how features from some groups are taken into account by most of the components. That is, population,

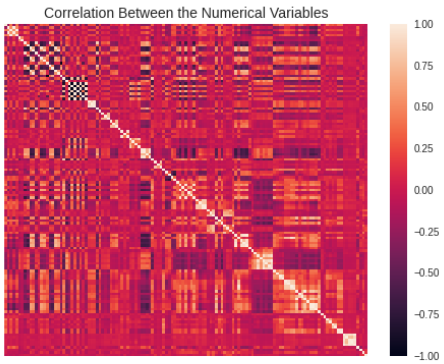


Figure 4.1: Correlations

economic, and labour indicators seem to capture a large amount of variance. The components are looked into more detail on the conclusions.

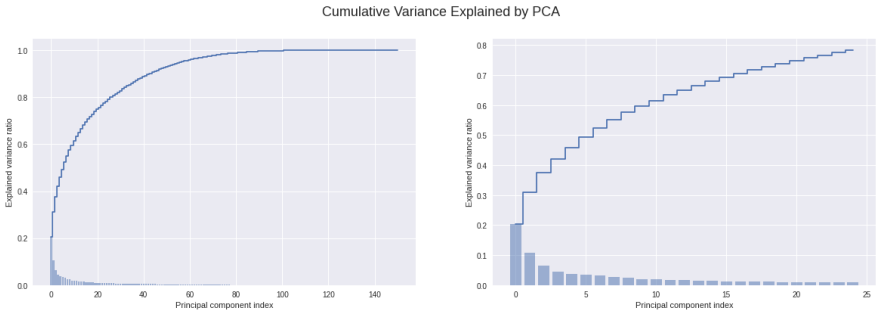


Figure 4.2: Histogram of the percentage of missing values.

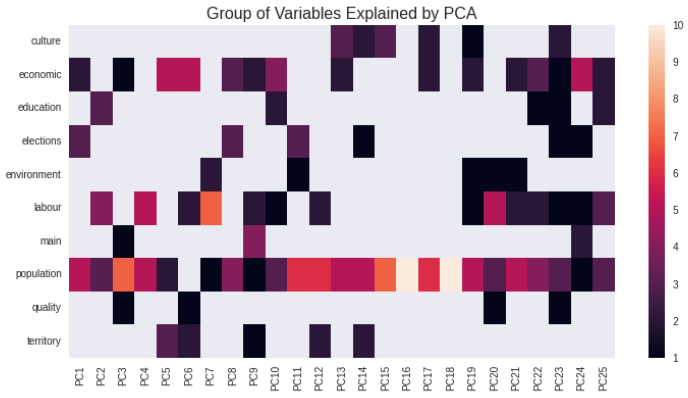


Figure 4.3: Histogram of the percentage of missing values.

After the PCA is computed, different clustering strategies are applied to check how differentiable is the data. First, `KMean` (KM) method is applied with different number of clusters. It is a

Model	Parameter (K / d)	CH score	Silhouette score	DB score
KM	2.0	212.694	0.224	1.91
KM	5.0	120.165	0.126	1.966
KM	10.0	96.82	0.132	1.734
KM	15.0	94.6	0.178	1.701
KM	20.0	100.186	0.187	1.54
AP	0.5	196.117	0.265	1.046
AP	0.6	193.142	0.294	1.048
AP	0.7	202.856	0.27	1.044
AP	0.8	205.407	0.295	1.023
AP	0.9	207.811	0.287	1.06

Table 3: Clustering results

reliable general-purpose algorithm that computes the clusters based on the distances between observations, minimizing inertia. The number of clusters needs to be given before initialization. Second, `AffinityPropagation` (AP) method is fitted with different damping values. Contrary to KM, the number of clusters is computed by the algorithm itself, so the important parameter to tune is the damping. It computes the distances based on the graph of nearest neighbors. The results of both methods have been assessed with the Calinski-Harabasz index, Silhouette coefficient, and Davies-Bouldin score, depicted in table 3 as seen in class. It can be observed how the seemingly best method is the AP with a damping value of 0.9. Such algorithm results in the large amount of 80 clusters, so for illustration purposes the results obtained with KM and 4 clusters are depicted in figure 4.4. It can be clearly seen how clusters are easily differentiable in the first components, but they get more entangled in the ones that do not capture that much variance. Top-right cluster in the first plot represents municipalities with a higher representation of old people, good elections turnout, and large amount of land ownership, while the bottom-left captures the opposite.

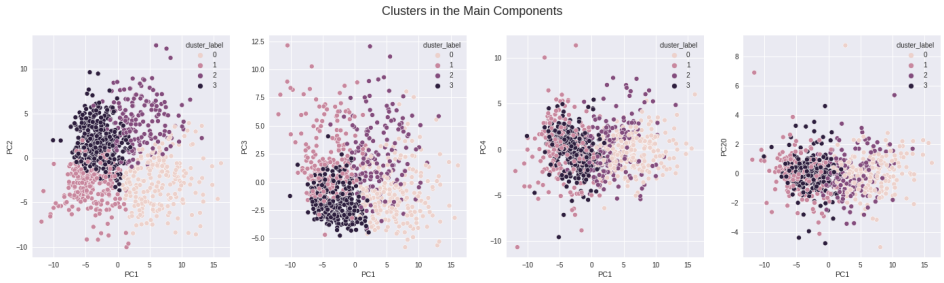


Figure 4.4: Histogram of the percentage of missing values.

5 Validation and Sampling Strategy

The considered validation and sampling strategy has been the leave-one-out cross-validation (loocv). First, the data is split between train (77%) and test (33%) sets. The latter will only be used to assess the generalization capabilities of the final model. Since the labels are slightly imbalanced because the *medium* has more data than *low* and *high*, the split is stratified to ensure the same proportions among both sets. Then all the models are fit and assessed over the train test using loocv with 10 folds. Due to the characteristics of the data, which does not contain a large number of observations, 10 folds have been preferred over 5. Such method is applied for each iteration of the hyperparameter tuning

that is tried using the `RandomizedSearchCV` method. This method differs from `GridSearchCV` in the fact that it does not try all the possible combinations, it stops after a user-specified threshold. This results in a better efficiency during the time of optimization at the expense of ensuring the optimum.

6 Modeling

The pre-processed data has been fitted into different models, which have all been fine-tuned using the `RandomizedSearchCV` method. All the hyperparameters tuned as well as the best iteration are shown in the subsections describing the different models. Results accomplished with the best combinations are depicted in table 6.

6.1 Linear Discriminant Analysis

The Linear Discriminant Analysis (LDA) assumes that the class-conditional distributions $p(x|y)$ all follow a Gaussian distribution, thus $p(x|y) \sim \mathcal{N}(\mu_k, \Sigma_k)$. In other words if the data is separated by the target variable, the explanatory variables should all follow a Gaussian distribution. Because of this, we can model discriminating functions for each class k, using the probability density function of a Gaussian distribution and the assumption that the prior distributions are $p(y = c_k) = \pi_k$, which corresponds to the frequency of appearance of class k.

$$g_k(x) = \log P(y = c_k)P(x|y = c_k) \tag{1}$$

$$= \log \pi_k \cdot \frac{1}{\Sigma_k \sqrt{2\pi}} \exp^{-\frac{1}{2}(\frac{x-\mu_k}{\sqrt{\Sigma_k}})^2} \tag{2}$$

$$= \log \pi_k - \log(|2\pi\Sigma_k|^{\frac{1}{2}}) - \frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1}(x - \mu_k) + C \tag{3}$$

The functions $g_k(x)$ are called the Quadratic Discriminant functions which are central to the Quadratic Discriminant Analysis (QDA) discussed in the next section. In order to arrive at the LDA, the assumption that the class conditional covariances σ_k^2 are equal is made, which allows to simplify $g_k(x)$ to

$$g_k(x) = \log \pi_k + \mu_k^T \Sigma^{-1} x - \frac{1}{2}\mu_k^T \Sigma^{-1} \mu_k + C \tag{4}$$

The implementation of LDA in `sklearn` follows this equation. In this function, $(x - \mu_k)^T \Sigma_k^{-1}(x - \mu_k)$ corresponds to the Mahalanobis distance between observation x and μ_k, the mean of class k.

In order to predict values, `sklearn` computes the log-posterior g_k as

$$\log P(y = c_k|x) = \omega_k^t x + \omega_{k0} + C \tag{5}$$

where $\omega_k = \Sigma^{-1}\mu_k$ and $\omega_{k0} = -\frac{1}{2}\mu_k^t \Sigma^{-1}\mu_k + \log \pi_k$, of which the former corresponds to the coefficients per class for each explanatory variable and the latter corresponds to the intercept for each of the classes. Thus, the estimation of ω_k and $_{k0}$ can be done solely by considering the observations X, since $\Sigma = \frac{1}{n-1}X^t X$, which, using the singular value decomposition $X = USV^t$, is equal to $\Sigma = \frac{1}{n-1}VS^2V^t$. Thus LDA is not expensive to compute and could be implemented easily in `python`. For this project, the LDA implementation of `sklearn` was used.

Because LDA assumes that the covariances of the explanatory variables for all of the classes are equal, it is only able to divide the decision space in a linear fashion (as the name suggests). Thus, in

the case that the data takes more complex forms shapes, which can't be divided with linear planes, LDA will not perform well.

The parameters tuned in the search have been: `solver` and `shrinkage`. The best combination of parameters is accomplished using *svd* solver, which does not allow shrinkage.

6.2 Quadratic Discriminant Analysis

Because QDA does not assume the covariance to be equal for different classes k, it needs to estimate Σ_k instead of Σ. It does so simply by doing the same estimation for Σ, but for each separate class k: $\Sigma_k = \frac{1}{n-1} X_k^t X_k$, which, with $X_k = USV^t$, becomes $\Sigma_k = \frac{1}{n-1} VS^2V^t$. This has the advantage that more complex relation can be captured with the decision boundaries, since they are not linear anymore.

It is likely that QDA outperforms LDA when each of the classes has different covariances in their explanatory variables. An example of such a case could be predicting the type of bird based on wingspan, beak length, flight time, pitch, etc. In the case of the Catalan Language, no such case can be made, since the data all come from the same class, namely municipalities. However, it could be that a distinct difference exists between the different degrees of Catalan speaking people and thus, weighing QDA vs LDA makes sense.

The only parameter tuned in the search has been: `reg_param`. The best value for this parameter is 0.91.

6.3 Gaussian Naive Bayes

The Gaussian Naive Bayes (GNB) works much the same as the QDA and LDA as it uses the same class priors π_k and class-conditional densities $P(x|y = c_k)$. Just like in QDA, the GNB assumes that there is class specific covariance matrices Σ_k for each class k, however, it GNB also assumes that all explanatory variables are independent, implying that the covariance matrices are diagonal matrices.

In the standard Naive Bayes formulation, it is assumed that the class-conditional densities follow a normal distribution, thus $p(x|y = c_k) \sim N(x|\mu_k, \Sigma_k)$. Then, the posterior predictor would be

$$g_k(x) = \log \pi_k + \sum_{j=1}^{d} \log p(x_j|y = c_k) \tag{6}$$

It is easy to see that for the GNB, $p(x_j|y = c_k) \sim N(x|\mu_k, \Sigma_k)$ can be replaced by $p(x_j|y = c_k) \sim \mathscr{N}(x|\mu_k, \Sigma_k)$.

In this project, `sklearn` was used to implement the GNB. `sklearn` uses MLE to estimate parameters μ_k and Σ_k.

The only parameter tuned in the search has been: `var_smoothing`. The best value for this parameter is 0.001.

6.4 Linear Regression

A linear regression model aims to minimize the mean squared error defined as

$$J(\theta_0, \theta_1) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{7}$$

$$= \sum_{i=1}^{N} (y_i - \theta_0 + \theta_1 x_i)^2 \tag{8}$$

where θ_0 are the intercepts, θ_1 are the coefficients, and N is the amount of observations. In order to minimize this loss function, `sklearn` approaches the problem like an Ordinary Least Squared problem for which it can be shown that the hat function

$$\theta_{lse} = (X^T X)^{-1} X^T y \tag{9}$$

finds the optimal coefficients and intercept θ_{lse}. By substituting singular value decomposition $X = U\Sigma V^{-1}$ for X, $\theta_{lse} = V\Sigma^{-1}U^T y$ can be found, which is much more efficient computation. The hat function also works for the multivariate case. In this case, linear regression essentially attempts to divide the space into d-dimensional planes (d being the amount of features). Notice that, when comparing the equation for linear regression with LDA, what is being computed is essentially the same, just the method differs. It would thus be expected that linear regression and LDA perform equally as good.

The parameters tuned in the search have been: `solver`, `penalty`, `C`, and `class_weight`. The best configuration for this parameters is *newton-cg*, *l2*, 0.1, and *None*, respectively.

6.5 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) approaches modelling the data different that the before described methods such as LDA, QDA, or LR, in that, instead of analytically solving for θ_0 and θ_1, it solves it by performing what is called gradient descent. Given the loss function $J(\theta_0, \theta_1)$, gradient descent updates θ_0 and θ_1 each iteration by computing the gradient of the loss function and updating θ each iteration i based on learning rate α:

$$\theta_{i+1} = \theta_i - \alpha \times \nabla_\theta J(\theta) \tag{10}$$

The loss function $J(\theta)$ can be of all sorts. As an example, when the loss function is the squared mean error $(y - \hat{y})^2$, SGD is computing linear regression in yet another way. Since, in the case of this project, a multivariate classification is done, it makes more sense to instead use a logistic loss function, which is defined as

$$h_\theta(x) = \frac{1}{1 + \exp^{(-\theta^T x)}} \tag{11}$$

$$J(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases} \tag{12}$$

$$J(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(h_\theta(x)) \tag{13}$$

The `sklearn` implementation of SGD allows for many hyper parameters and thus tuning is necessary. One option is to apply a regularization, either Ridge (L2) and Lasso (L1), by adding a penalizing term to the loss function, in order to prevent the coefficients from exploding. Another hyperparameter regards the amount of iterations needed, which, when set to high, might result in overfitting and, when set too low, might result in underfitting.

It is expected that SGD does not outperform any of the other before mentioned methods.

The parameters tuned in the search have been: `loss`, `penalty`, `alpha`, `l1_ratio`, and `class_weight`. The best configuration for this parameters is *log*, *elasticnet*, 0.01, 0.2, and *None*, respectively.

6.6 K-Nearest Neighbours

The K-Nearest Neighbour classifier (KNN) is a supervised, non-parametric method that performs the k-nearest neighbors vote. During the classification, k is defined by the user, and the algorithm classifies the label based on the values of the most near observations. The most common measure to compute the distance is the Euclidean.

The issue with this algorithm is that when the class to be predicted is highly skewed, most observations might end being labeled as the majority one. For that, the distance is multiplied by a weight that depends on the distance to the other points.

The parameters tuned in the search have been: `n_neighbors`, `weights`, and `algorithm`. The best configuration for this parameters is 24, *distance*, and *auto*, respectively.

6.7 Decision Trees

Decision Trees (DT) are a supervised, non-parametric, and binary segmentation learning technique. Recursive binary splitting is used to grow the tree, which follows a top-down and greedy approach. That is, the algorithm begins with all the observations belonging to a single region and the binary splits are chosen at each step rather than looking at which will be the best combination in the future. The splits are decided based on either the Gini impurity [14] or the Entropy [15], which are defined as follows:

$$G_m = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}) \tag{14}$$

$$H_m = \sum_{k=1}^{K} p_{mk} log(p_{mk}) \tag{15}$$

where p_{mk} is represents the proportion of observations in the mth region that are from the kth class.

Both strategies are measures of the likelihood of a random observation in the set being missclassified. As such, it is said they measure the purity of a node.

Decision Trees are easy to interpret, as they can be displayed visually. However, trees fit with different subsets from the same data can yield completely different results. They suffer from high variance and its predictions are not as accurate as other methods.

The parameters tuned in the search have been: `criterion` and `max_depth`. The best configuration for such parameters is *gini* and 4.

6.8 Random Forests

Random Forests (RF) are an improvement over the bagging method, where multiple trees are grown from the same set and then averaged. Such trees are not pruned, which means they suffer from high variance, but their bias is lower. However, the issue with the variance is solved by averaging them (in the case of classification, voting is the actual approach). RF do the same, but with a small tweak. Unlike bagging, where all the predictors are considered while growing each tree, only a random sample of m predictors is considered from the full set of p predictors, where usually $m = \sqrt{p}$.

Thanks to bootstrapping, for each bagged tree (from 1 to n grown trees) the remaining observations not participating in the actual fit are used for predictions, which lead to the out of bag error (OOB). This error decreases as more trees are averaged until it converges, in other words, there is a point where growing more trees does not improve the accuracy. This estimate is practically equivalent to the leave-one-out cross-validation. Another relevant aspect from RF is that it provides a measure for variable importance by adding up the Gini index of its trees.

Contrary to DT, RF are similar to a black-box, so even if its results are much more reliable, their interpretation is too complex.

The parameters tuned in the search have been: `n_estimators`, `max_features`, `max_depth`, `min_samples_split`, `min_samples_leaf`, and `bootstrap`. The best configuration for this parameters is 1000, *sqrt*, 60, 2, 2, and *True*, respectively.

6.9 Support Vector Machines

Support Vector Machines (SVM) attempt to define a d-dimensional hyperplane in order to separate and distinguish different classes in the data. It does so by maximizing the so called margin m. Intuitively in 2-D, this means that the SVM draws a line between two groups of data points in such a way that the orthogonal distance from the closest point (called a support vector) of one class to the line is equal to the orthogonal distance of the support vector of the other class. The distance between the support vectors and the line is called the margin. As this method is highly sensitive to outliers (imagine a point of one class laying far away from other points of the same class, but close to the points of another class), the soft margin is a better metric to be maximized, as it allows for some outliers to be present on the other side of the line.

With the margin being defined as $m = \frac{2}{||w||}$, the maximization of the soft margin is defined as

$$\min_{w,b} \frac{1}{2}||w||^2 + T\sum_{i=1}^{n} \xi_i \tag{16}$$

where w are the coefficients of the hyperplance, b are the intercepts for the hyperplane, and ξ_i is the upper bound of the number of errors for class i is the with the constraints such that

$$w^T x_1 + b \geq \quad 1 - \xi_i \quad \forall x_1 \in C_1 \tag{17}$$

$$w^T x_2 + b \leq -1 + \xi_i \quad \forall x_2 \in C_2 \tag{18}$$

$$\xi_i \geq \ 0 \tag{19}$$

where C_1 are all the explanatory variables that belong to class y_1 and x_2 those that belong to class C_2. The coefficients w can then be found by

$$w = \sum_{j=1}^{s} a_{t_j} y_{t_j} x_{t_j} \tag{20}$$

where t_j are the indices of the s support vectors, and α_{t_j} are the Langrange multipliers.

Because, not all data is separable with a linear hyperplane (or line), SVM employ what are called kernels. Kernels are sets of functions that transform the data to a higher dimension in order to be able to separate classes with a linear hyperplane. For example, the polynomial kernel for degree-d polynomials is defined as

$$K(x, y) = (x^T y + c)^d \tag{21}$$

When fitting a SVM with a specific kernel, the SVM will not only compute w but also the optimal parameters for the kernel, which, in the case of the polynomial kernel, would be the amount of degrees d.

The parameters tuned in the search have been: `kernel`, `gamma`, `C`, `class_weight`, and `decision_function_shape`. The best configuration for this parameters is *rbf*, 0.001, 10, *ovo*, and *None*, respectively.

6.10 Voting Classifier

The idea behind this ensemble is to combine multiple classifiers and use their predictions to perform a majority vote or the average predicted probabilities for each observation. The former strategy decides the label by taking the class more represented across the results. The latter strategy requires the probabilities computed by each model and decides based on the maximum value obtained from the sum of probabilities. Both strategies have been tried. The results can be depicted in 6. The soft strategy seems to be the best option. Moreover, if DT, RF, and SVM are not used during the voting it is capable of increase its score. It is not clear why leaving the SVM out results in an improved performance. Another method, called `StackingClassifier` has been tested but its results were not robust enough to consider.

6.11 Multiple Layer Perceptron

The idea behind the Multiple Layer Perceptron (MLP) is the foundation of all deep learning neural networks. Officially, a MLP should have 3 hidden layers, however in the upcoming elaboration this is not taken too seriously. Also, it is very difficult to explain everything that should be considered when describing neural networks, and thus a simple overview is given, together with the hyperparameters that were important to this project.

A MLP works by using so called neurons. A neuron can be seen as a node in a graph which takes some input, applies some weights W, adds an intercept β and the transforms the data using an activation function σ. For example, a simply binary logistic regression can be modelled with MLP, one neuron, and the sigmoid activation function

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{22}$$

$$y = \sigma(WX + \beta) \tag{23}$$

Where the W and B can be learned using gradient descent as explained above in SDG.

It is possible to add multiple nodes in a single layer all receiving input X and having their own weights $W_i^{(1)}$. The output of all of these neurons combined can be forwarded into another set of neurons, which again multiple some weights $W_i^{(2)}$ with the input, add an intercept $B^{(2)}$ and apply some specified transformation σ. When adding multiple layers, the weights in all the layers can be still be learned by gradient descent, by means of the chain rule.

All MLPs have an output layer, which has a specific activation function that corresponds to the task at hand. For example, in a 4 class classification task, 4 neurons can be chosen to have the softmax activation function, which makes the MLP choose the most likely class.

Activation functions play an important role in the MLP as they are able to transform the data into different shapes in the first layers, which allows the latter layers to clearly distinguish between classes. Therefore, choosing the right activation function for the hidden layers is essential. `sklearn` provides 4 different activation functions, namely the "logistic" (or sigmoid), "tanh", "identity", and "relu" functions. The sigmoid function has already been described above, and outputs a value between 0 and 1. The "tanh" function outputs a value between 1 and -1, the "identity" function simply returns z, and the "relu" function returns $\max 0, z$. Although the sigmoid function was used in the early

	Accuracy	F1-score (Micro)	F1-score (Macro)	F1-score (Weighted)
MLP	0.668	0.668	0.656	0.666

Table 4: mlp

stages of research in neural nets, it is widely considered that the "relu" and "tanh" are more effective activation functions.

Next, the amount of hidden layers, as well as their sizes play a large role in the effectiveness of the MLP. Multiple layers might allow for complex transformations to find nuanced patterns in the data, however, care should be taken as it can often quickly lead to overfitting on the training data. The size and amount of these layers also determine the training time that is needed in order to train the model. The more and larger layers, the more weights have to be updated, which takes more time.

Another hyperparameter to consider in the training of a MLP is the solver, which is the algorithm that performs the gradient descent. Currently, the "adam" optimizer is considered to be the most optimal optimizer, however, `sklearn` also provides the SDG (as described above) and "lbfgs" solvers.

As with SGD, regularization is automatically implemented in the MLP version of `sklearn` and is tuned by the hyperparameter α, which can be set to 0 in order to deactivate regularization.

The MLP has been trained apart from the other models because it has been fitted using the data without the PCA transformation. This was done as the MLP should be able to realize these transformations itself. The parameters tuned in the search have been: `alpha`, `hidden_layer_size`, `activation`, and `solver`. The best configuration for this parameters is 0.1, *(10,)*, *relu*, and *adam*, respectively. Its results are depicted in 5. It can be seen how it performs very similar to the models fitted with the data transformed by the PCA. It is even outperformed by most of them, but his might be due to the fact that a MLP is not suited for this kind of data. It was hypothesized that the MLP would outperform other models, since it did not use the PCA data (which explains 80% of the data) and thus might be able find a more complex relation, not found by PCA. However, more or less the same results were found, implying that this higher complex relation might not exist, and that the models are only basing their prediction on the most important components.

6.12 Multiple Layer Perceptron in Tensorflow

The MLP was also built in `tensorflow` in order to compare the "relu" activation function with the "LeakyReLU", which instead of return $\max 0, x$, returns x for all $x > 0$ and αx for $x < 0$. The advantage of this activation function, is that it is able to backpropagate the gradient better as for the case of relu, when $x < 0$ the gradient would be 0. Next to this, using `tensorflow` allows to plot the history of the iterations so that conclusions can be drawn on whether the model overfits or not.

For this, the best MLP from `sklearn` was built in `tensorflow`, which as described above had an L2 regularization term of 0.1, *(10,)* hidden layers, *relu* activation function, and the *adam* optimizer. The *relu* was replaced with a $LeakyReLU$ function with α set to 0.1 (which nearly mimics *relu*.

Because the cross validation and sampling methods from `sklearn` do not easily integrate with tensorflow, a different but similar sampling and validation method was used to obtain the result. The method was as follows. Twenty runs were performed in which each time, the data was randomly shuffled (changing the seed) into a training and validation set, a new model was build, trained, and used for prediction with the validation set, to get a weighted and macro F1-score, and an accuracy score after 20 runs. Each model was trained on 20 epochs, with 59 iterations each, accounting for a total of 1180 iterations. Also the training history was saved for each of the models.

As can be seen, it seems that the $LeakyReLU$ performs slightly better than the *relu* activation function. Furthermore, the training history was plotted as a mean and standard deviation for each epoch to get a good idea of whether early stopping might improve the MLP, which can be seen in figures

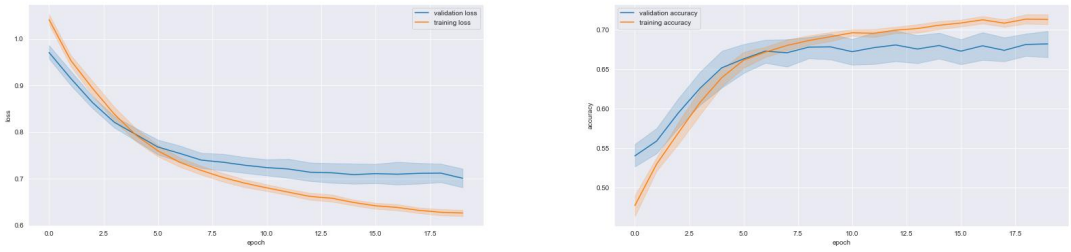
	Mean Accuracy	Mean F1-score (Macro)	Mean F1-score (Weighted)
MLP in Tensorflow	0.682	0.646	0.675

Table 5: mlp

Model	Accuracy	F1-score (Micro)	F1-score (Macro)	F1-score (Weighted)
DTC	0.61	0.61	0.537	0.576
GNB	0.641	0.641	0.62	0.638
KNC	0.677	0.677	0.617	0.647
RFC	0.66	0.66	0.625	0.647
LDA	0.659	0.659	0.638	0.653
QDA	0.655	0.655	0.659	0.654
VCs	0.667	0.667	0.639	0.657
LR	0.664	0.664	0.648	0.66
SGDC	0.668	0.668	0.646	0.661
VCh	0.678	0.678	0.656	0.671
SVC	0.683	0.683	0.661	0.676
VCb	0.686	0.686	0.681	0.686

Table 6: Results

6.1. These figures show that the validation accuracy, and with that the F1-scores heavily depend per data sample, and thus using a random sampling method is crucial for building an accurate model. Second, it seems that although the validation accuracy is increasing slightly, not much improvement is made after 10 epochs, suggesting that 500 iterations for MLP is enough to fit the data.



(a) I1 (b) I2

Figure 6.1: Training history of MLP

7 Final Results and Conclusions

As stated in the data section of this report, some heavy limitation existed in the data used. The main problem of this limitation, is that the models might be able to infer a relation between variables of different years, which ideally should be corrected for. Even worse, since all models almost reach the same F1-score and accuracy, there might be some hidden pattern connected to this fault that determines the outcome of the models, instead of the data itself. Obviously, in any professional setting, this mistake would not be allowed, and would be corrected for. However and unfortunately, this shortcoming was realized to late in the project to completely account for and thus it was deemed

True \Predicted	Low	Medium	High
Low	52	22	2
Medium	12	108	32
High	0	25	51

Table 7: confusion

Generalization	VCb
Accuracy	0.694
F1-score (Micro)	0.694
F1-score (Macro)	0.693
F1-score (Weighted)	0.696

Table 8: testset

sufficient to fix it partly, and discuss it in the report. The model discussion should thus be taken with a grain of salt.

The results depicted in table 6 show how the best performance is accomplished the `VotingClassifier` with a soft configuration and fewer models in its core (LDA, QDA, KNN, LR, and SGD). As such, the best F1-score (weighted) obtained is 0.686, which compared to the LR supposes a marginal improvement. It is worth to mention that since only 80% of the variance is captures by the data used to fit the models, the reachable scores might be capped, although training a NN with all the data does not seem to improve much further. The performance of the best model in the test set can be depicted in tables 7 and 8. It can be observed how the majority of labels are correctly classified, although the model tends to predict more labels as medium than it should. Regarding the generalization, it can be observed how the weighted F1-score is very similar, even slightly higher, to the one obtained through the cross-validation in the training set, which indicates that the model is capable to generalize, although it might be slightly underfitted.

Since the best model is an ensemble, properly analyzing the results is not possible. However, some light can be found looking at the variance importance provided by the RF, which can be depicted in 7.1 and linking the principal components to the original data. It can be observed how the most important variables seem to be PC1 and PC2, followed by PC3, PC9, PC4, PC13, and surprisingly PC25. The first four components imply that the age of the population, the turnout in the elections, land ownership, education level, registered unemployment, and nationality of the people are very relevant to predict if a municipality has a low or high degree of Catalan speakers. It is logical to think that municipalities with more elders, more land ownership, high turnout, and low degree of immigration might be small areas where the weight of Spanish is not relevant. The remaining components also take into account this indicators, but also take into account if the municipality has permanent pastures, ownership of livestock, and tourist accommodations. Again, it seems that rural municipalities are highly correlated to the percentage of Catalan speakers, which is logical.

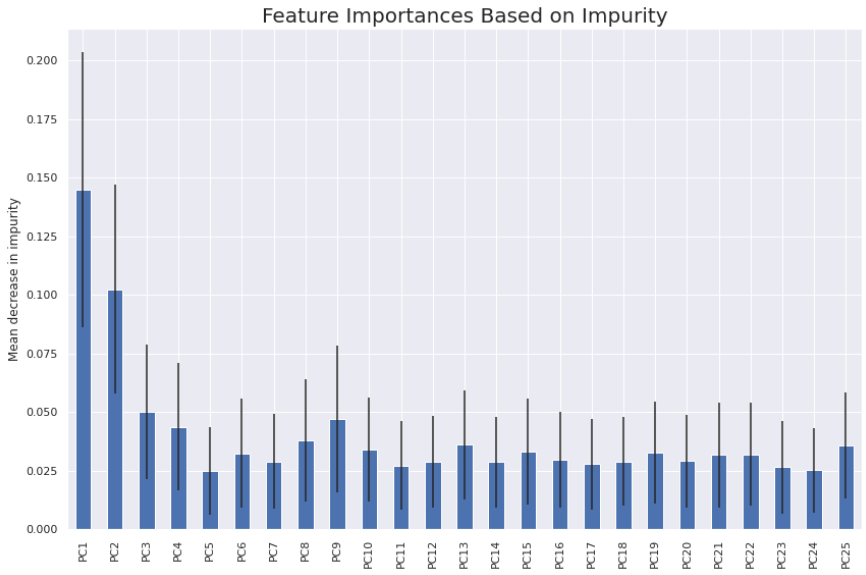


Figure 7.1: Histogram of the percentage of missing values.

References

[1] Els usos lingüístics de la població de catalunya. https://llengua.gencat.cat/web/.content/documents/publicacions/publ *opuscle.pdf*. *Accessed* : 2022.

[2] Gencat, llengua catalana, dades i estudis. https://llengua.gencat.cat/ca/serveis/dades$_{ie}$*studis*/. *Accessed* : 2022.

[3] Idescat. https://www.idescat.cat/. Accessed: 2022.