Advanced Machine Learning - AML
Masters in Data Science

Final project

Adults income

Predicting whether the salary of an adult is greater or lower than 50.000 dollars

Clotilde Rodier
Sebastian Paglia

Adults income

January 16th, 2023

Problem

This project aims to analyze the impact of several features affecting the salary of people. We will work on a data set found in the census web page. In order to achieve our goal, we will follow the suggested pipeline including splitting data, performing preprocessing data techniques, univariate analysis, correlation among the variables, feature selection, modelling and comparison of the results obtained to establish the best classification method to our binary target. We have chosen this problem because it would allow us to understand a real case scenario in which we could apply Machine Learning techniques.

1 Dataset

Among the initial variables, we have fifteen columns that will be explained more detailed later on the reading, but just to give a brief description and understanding of them:

1. *age:* the age of an individual.
2. *workclass:* a term to represent the employment status of an individual.
3. *fnlwgt:* the final weight, this is the number of people the census believes the entry represents.
4. *education:* the highest level of education achieved.
5. *education-num:* an id given to a given highest level of education achieved.
6. *marital-status:* the marital status of an individual.
7. *occupation:* the general type of occupation of an individual (work).
8. *relationship:* a representation of what this individual is relative to others.
9. *race:* a description of an individuals race.
10. *sex:* the biological sex of the individual either male or female.
11. *capital-gain:* the capital gains for an individual.
12. *capital-loss:* the capital loss for an individual.
13. *hours-per-week:* the hours an individual has reported to work per week continuous.
14. *native-country:* the country of origin for an individual.

15. *salary:* the information that whether or not an individual makes more than 50.000 dollars annually (binary target).

The dataset is composed of 48.840 rows whose significance will be checked during the preprocessing step. Based on the first approach, we can split these variables into:

categorical (8 variables): workclass (9 categories), education (16 categories), marital-status (7 categories), occupation (15 categories), relationship (6 categories), race (5 categories), sex (2 categories), native-country (42 categories);

numerical (6 variables): age (74 unique values), fnlwgt (28.523 unique values), education-num (16 unique values), capital-gain (123 unique values), capital-loss (99 unique values), hours-per-week (96 unique values);

and binary target: salary.

We realised that our dataset is highly unbalanced in many variables but most importantly in our target as 76% people earn less than 50K per year and 24% people earn more than 50K, we will solve this issue during the preprocessing stage.

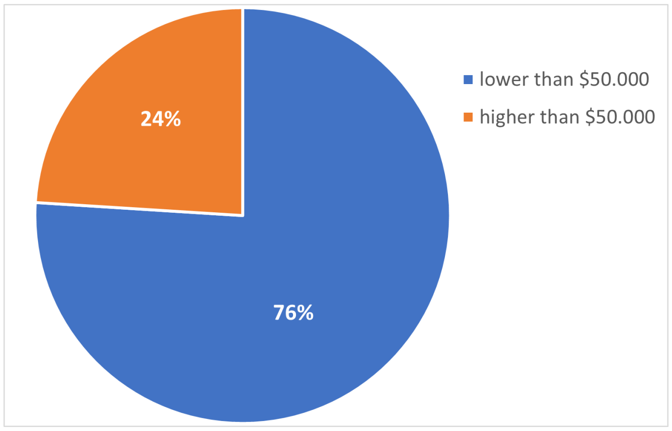


Figure 1: Unbalanced target

2 Methodology

To carry out this project, we have followed the illustrated methodology (2): we started with the data cleaning and preprocessing to refine the input data, including an exploratory data analysis of the variables, missing values imputation, recategorization of variables to model the target prediction, and data splitting for model validation. Afterwards, the modeling step included trials with different algorithms and parameters to get the best approximation to classify data. We approached this phase recursively until we reached the best model per algorithm based on the results obtained with training dataset. Once selected the best combination of parameters for a specific algorithm the final model was selected and we get the predicted values and the corresponding metrics. Finally, we made a final analysis with the results using the split test sample.

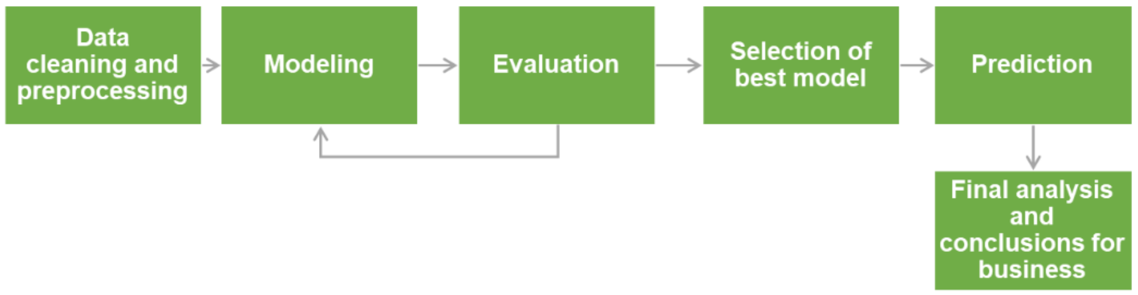

Data cleaning and preprocessing
Modeling
Evaluation
Selection of best model
Prediction
Final analysis and conclusions for business

Figure 2: Methodology used for this project

3 Data Cleaning and preprocessing

3.1 General cleaning

First of all, it was seen that many of the column names had an extra space character at the beginning of the string, so some functions were applied using pandas library to clean them by deleting those spaces. But this issue happens not only on columns, but also inside some of the categories so we did the same cleaning over those cases. After checking if there were any duplicate values, we found 29 of them that we decided to remove, taking into account that in this dataset should not appear any because of the characteristics of the columns where fnlwgt gives the weighted value for each row (meaning the times that exactly row should be multiplied by).

3.2 Feature Selection

Afterwards, we made some decisions regarding which features we want to keep and which ones to drop and why. As we want to predict the salary, we want to avoid having the same information in more than one column because it could change the weight of one feature on the model we define. The two columns: education and education-num have exactly the same information. Then, we delete the column education-num to keep the information just once. Another issue could be the relationship column as it echoes the marital-status column a lot.

Then, we should also pay attention to columns which aren't useful for our future analysis. This is the case for the 'fnlwgt' column. Indeed, as mentioned before, it represents a weight assigned by the US census bureau to each row. The literal meaning is that you will need to replicate each row, final weight times to get the full data. So, we consider this information not useful for our analysis.

3.3 Null Values

At first sight, we don't have any value set as Null. However, we want to make sure they are not appearing in a different way or typo, so we decided to check the unique values of each column just to make sure that we are not missing some. Indeed, some columns (workclass, occupation and native-country) had a quote mark (?) as a category that obviously represents missing data that we should deal with. Therefore, we set those quote marks as NAs.

3.4 Outliers

We only have 4 numerical columns: age, capital-gain, capital-loss and hours-per-week. We will study each of them to find potential outliers but first it is important to have a visualization of these variables to have an idea of their distribution and also a brief description of their min, max and quartiles, using also histograms: 3

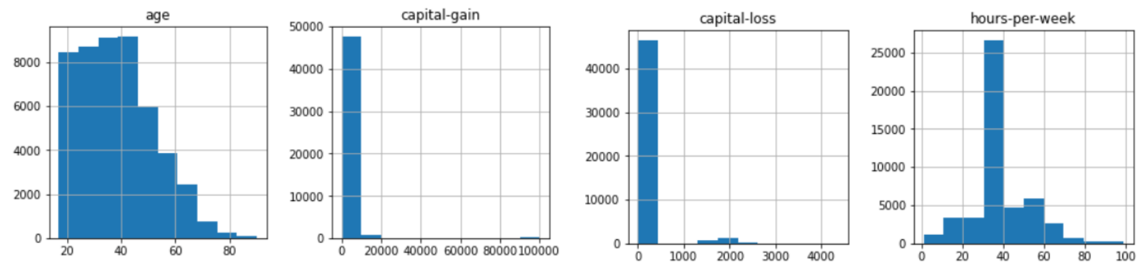


Figure 3: Histograms of numerical variables

After observing these histograms and all the occurring values for each column, we can already exclude the column age. Indeed, it seems to have coherent values (going from 17 to 99) and well distributed as we can see on the histogram.

At first, we decided to use the interquartile approach to find any outliers but as we can see for capital-gain and capital-loss, we have many values equal to zero which make the Q1 and Q3 equal to zero as well. To this end, every value higher than 0 is considered a strong outlier. However, these values still make sense in our study so we will study outliers differently for these two columns. Sorted from highest to lowest, in the capital-gain column we took a closer look at the first two values, being 99.999 the first one (appears 244 times) and 41.310 the second one (appears only 3 times).

This behavior does not appear on the capital loss, which its highest value is 4.356 followed by 3.900. Therefore, those with 99.999 were considered as errors (outliers) and taking into account that there are only 244 rows out of the 48.840 original ones (0,5%), we could have dropped them, but we don't want to miss important data, so we set them as NAs that will be imputed later.

Finally, observing the histogram of the last column, hours-per-week, and checking the quartiles (Q1 is 40 and Q3 is 45), 99 hours per week is the highest value. We don't have any profession where the number of hours of work per week is higher than 168 (the number of hours in one week 7*24=168) so no major outliers so far. Then, we take a closer look to the professions where the number of hours is higher than 70 (10 hours per day, working 7 days) to see if they were coherent and we found out that Farming-fishing (126 rows) and Prof-specialty (115 rows) were the top two professions using this threshold. We recognize that these values may be coherent in some cases, so we decided not to treat them as outliers and leave them as they are in our dataset.

3.5 Replacing Null values

After the preprocesses done so far, we have four variables with missing data: workclass (5,7%), occupation (5,8%), capital-gain (0,5%) and native-country (1,8%). The only numerical variable is the one that had outliers on it and we set them as missing values, so we will impute those using MICE algorithm. Once done, we still have the missing of the categorical variables but as they represent less than 6% of the dataset, we decided to drop them, so the remaining shape of the dataset are 45.194 rows and 12 columns.

3.6 Harmonising and grouping data

To finish the cleaning, we want to harmonize the variables of each column as much as possible. Therefore, we will add categorical columns, regrouping a large number of different values into groups. We can detail the grouping here:

- Age: young, adult and old people.
- Education: people ending their education before high-school, at high-school graduation, after high-school with some studies and after high-school with important studies.
- Marital status: people married, divorced and others.
- Native country: people from US and not from US (as we have more than 90% from US we can gather other countries).
- Capital gain: having 0 or more than 0 (as we have lots of 0).
- Capital loss: having 0 or more than 0 (for the same reason).
- Hours per week: people working less than 30 hours a week, between 30 and 60 and more than 60 hours.

We also harmonize the sex and the salary to obtain a binary output.

4 Exploratory

As a next step, visualising the data is essential to understand how it is structured and the relationship between them. First, we will observe the link between categorical variables and the target to have a preview on which variables influence positively or negatively the salary, by plotting histograms. The next figure 4 is a histogram between the sex (0 being men and 1 women) and the salary (0 being less than 50K and 1 being more than 50k) serving as an example.

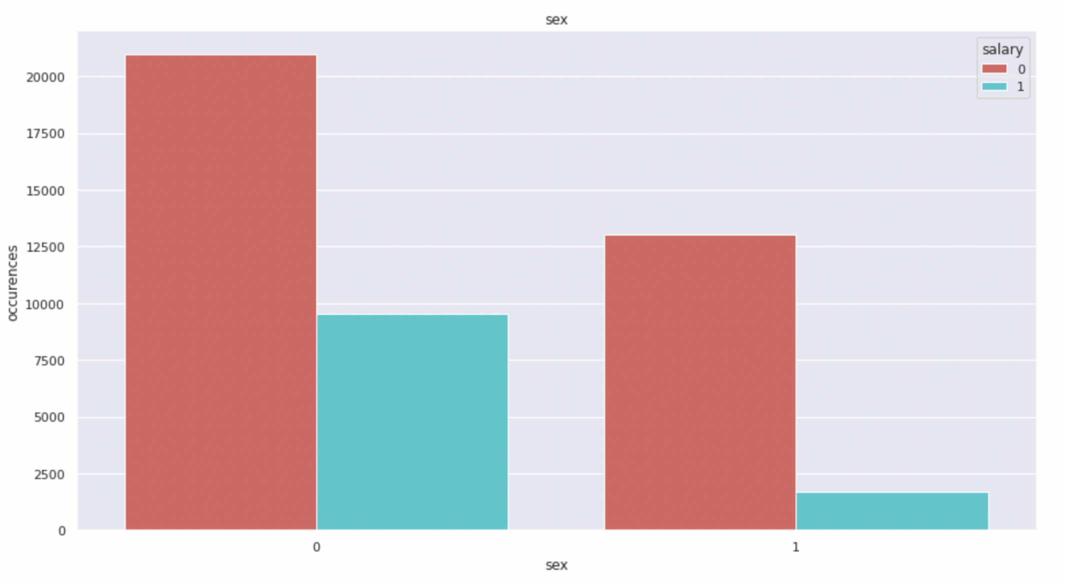
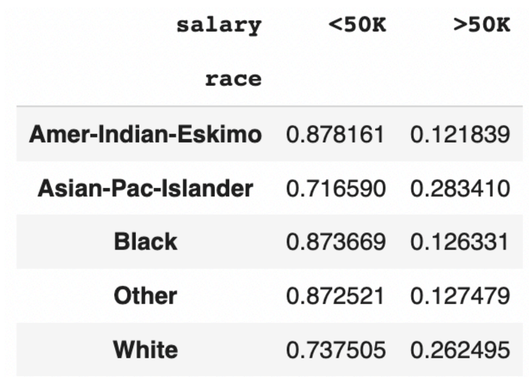


Figure 4: Histogram of sex and salary

We can clearly see that people earning more than 50k are mostly men. Also, a large majority of the women earn less than 50k while for the men it is more distributed. However, the fact that we have approximately 75% of people earning less than 50k in this dataset compared to only 25% people earning more than 50k could get us wrong. Also, we have more men than women in the dataset so it could influence our perception.

In the table below 5 We can see that almost 88% of people Amer Indian Eskimo earn less than 50k meaning more than the average (being 75%) so we can say that belonging to this category could be a prior major factor to have a small salary.

Then, we observed the link between the numerical variable "age" and the salary, taking differentiating also between gender using boxplots 6. We can see that the range of the quartiles for the higher salary category is more narrow than for the lower salaries, regardless the sex but increasing quartiles over the age.



salary	<50K	>50K
race		
Amer-Indian-Eskimo	0.878161	0.121839
Asian-Pac-Islander	0.716590	0.283410
Black	0.873669	0.126331
Other	0.872521	0.127479
White	0.737505	0.262495

Figure 5: Proportion of salary category for each race

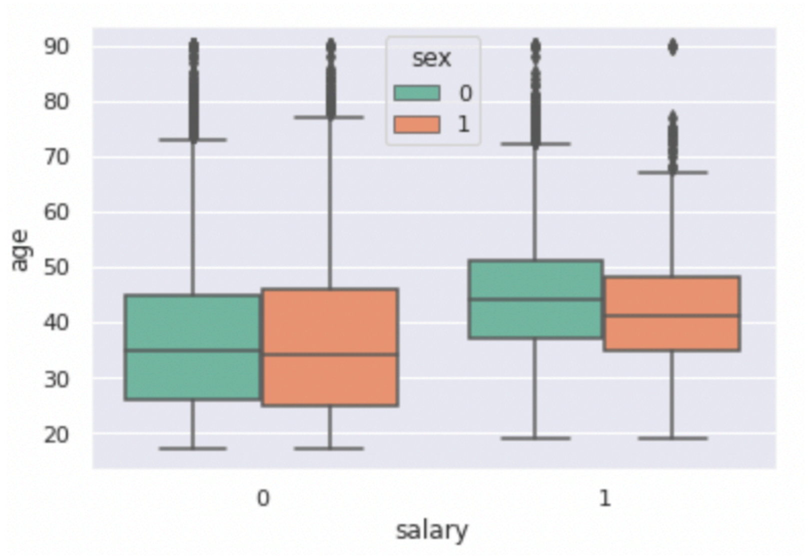


Figure 6: Boxplot of the age in function of the salary and sex

5 Cleaning before modelling

Before fitting a machine learning model, we need to prepare our data in specific ways to increase the chances of obtaining the best results possible.

5.1 Normalising

In the studied dataset, we have several variables all in different units and with also different distributions, which is why normalization could be a great solution. Indeed, the purpose of normalization is to transform data in a way that they appear dimensionless and have similar distributions. So, normalization gives equal weights to each variable so that no single variable distorts the model performance in one direction just because it has larger numbers.

Here, we apply and test three different types of normalization

- Standardisation: $\frac{X-\mu}{\sigma}$ will transform the data so it has a mean of 0 and standard deviation of 1.
- Min-Max scaling: $\frac{X-X_{min}}{X_{max}-X_{min}}$ will send the data to the range [0,1].
- Boxcox transformation: if $\lambda \neq 0$ then we have $\frac{X^{\lambda}-1}{\lambda}$, else we have ln(x). It transforms the data to try to fit a normal distribution.

We apply these normalization to the numerical data and observe the results in histogram 7 to choose the most suitable one.

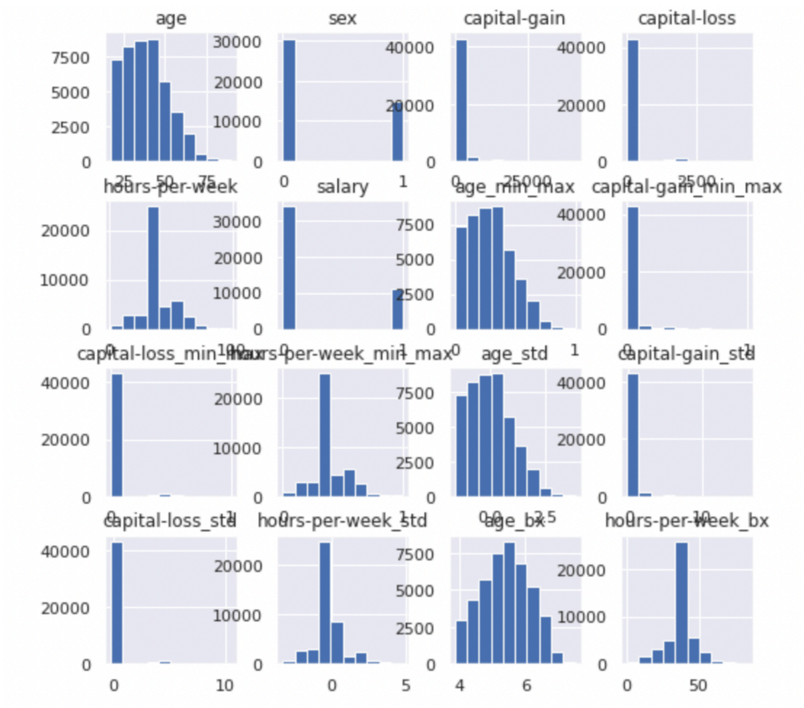


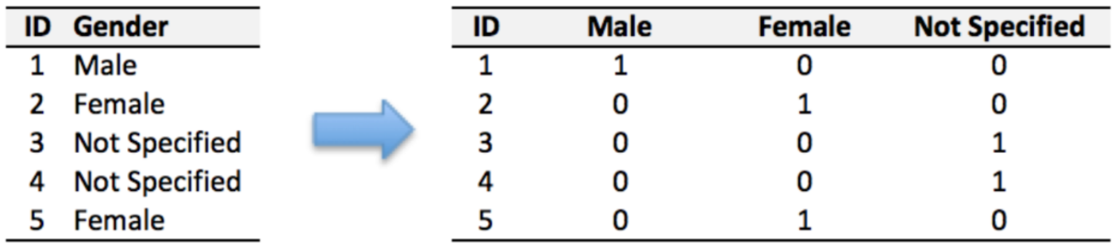
Figure 7: Normalisation histograms

Comparing the histograms, we can see that the best normalization process was obtained through the boxcox method. However, this can't be applied to the capital loss and capital gain as we many zero values. We are going to work with this method but only for the age and hours per week. We won't take into account the standardization and minmax because the results are not convincing enough.

5.2 Encoding

In our dataset we have several categorical variables, and we know that it is complicated to work with them as many machine learning algorithms cannot operate on categories directly. They require all input and output variables to be numeric. This means that categorical data must be converted to a numerical form which is why we are going to pass through encoding. We will apply two different methods.

The first one is the Hot Encoding (or dummy encoding) taking each distinct element in each categorical column and turning it into a binary vector (for k distinct elements in one column we will have k columns). Indeed, the new column is filled with 0 besides when it is equal to the initial element. We have the following example 8:



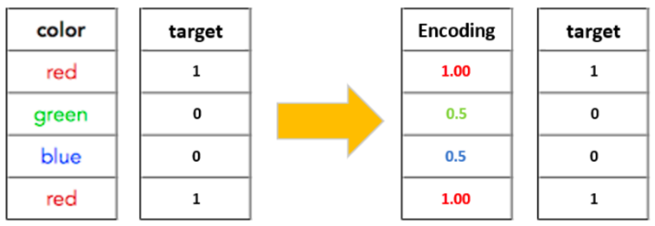
ID	Gender
1	Male
2	Female
3	Not Specified
4	Not Specified
5	Female

ID	Male	Female	Not Specified
1	1	0	0
2	0	1	0
3	0	0	1
4	0	0	1
5	0	1	0

Figure 8: Hot encoding example

To implement Hot Encoding, we use the function get_dummies() in the pandas library.

The second chosen encoding is the Target Encoding which is a numeration of categorical variables via the target column. In this method, we keep the same number of columns, but we replace the categorical variable with a numerical variable corresponding to the probability of the target. If we work in a binary case, the numerical variable is equal to the probability to have a target equal to 1, having the previous categorical variable. We have the following example 9:



color	target
red	1
green	0
blue	0
red	1

Encoding	target
1.00	1
0.5	0
0.5	0
1.00	1

Figure 9: Target encoding example

To implement this method, we use the TargetEncoder function from the category _encoders library.

5.3 Undersampling

As we saw at the beginning of the project, our dataset is highly unbalanced, which presents a common issue when dealing with real datasets. This could have an impact on the training models taking into account that most of the data we have for training, have zero for value on the target variable.

Therefore, it is advisable to deal with this problem before the modeling is applied. In this project, we will train the models using an unbalanced version and a balanced one that we will generate through the Undersampling method.

We could also had used oversampling to solve the unbalance, but as we have many rows in the dataset, we considered that it was better to lose some data than creating a lot of synthetic/fictitious new one.

6 Applying models

We are finally ready to apply various machine learning models, taking as an input several characteristics of one person and predicting its salary. The performance of each model will be measured with different metrics: the accuracy, the confusion matrix and the classification report for more information.

We will test each model on 12 distinct datasets having different specificity 10:

It enables us to analyse each model completely as not each specificity is great for every model: the target encoder could be beneficial for one model but not for the next one and same for all features. Therefore, we will fit each model and compute metrics for every one of this dataset, it will help us to distinguish which model offers the best results and for which type of data we should apply it to increase the chances of obtaining better predictions.

6.1 Splitting the data

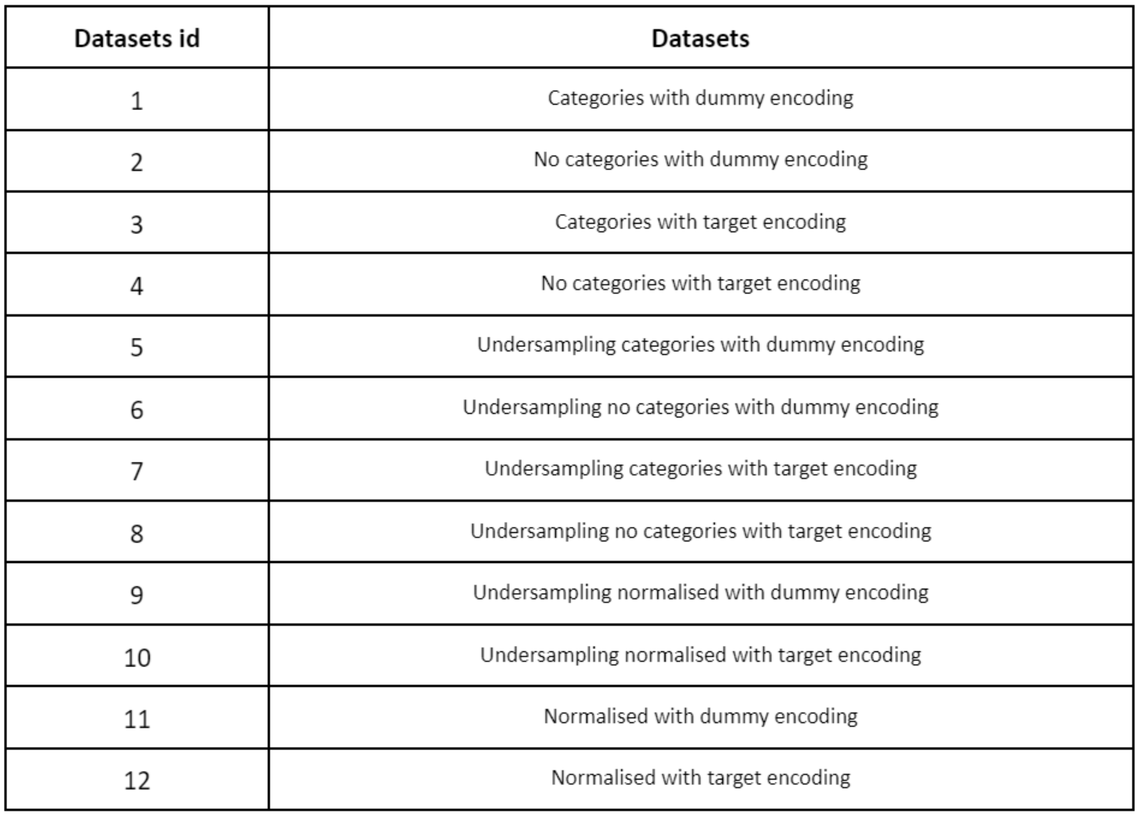
First, we split the data into two different datasets :

- Training set: the sample of data used to fit the model;
- Test set: the sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

The training set contains 75% of data and test set 25%.

After that, we resplitted the training set into two different datasets:

- Learning set: used to fit the models;
- Validation set: used to validate the results inside a training environment.



Datasets id	Datasets
1	Categories with dummy encoding
2	No categories with dummy encoding
3	Categories with target encoding
4	No categories with target encoding
5	Undersampling categories with dummy encoding
6	Undersampling no categories with dummy encoding
7	Undersampling categories with target encoding
8	Undersampling no categories with target encoding
9	Undersampling normalised with dummy encoding
10	Undersampling normalised with target encoding
11	Normalised with dummy encoding
12	Normalised with target encoding

Figure 10: The twelve datasets used to apply the machine learning models on

The learning set contains 75% of the training set and validation set 25%. Having the training set splitted into validation and learning datasets is important as it decreases the chances of overfitting.

6.2 Logistic regression

Logistic regression estimates the probability of an event occurring, such as voted or did not vote. Since the outcome is a probability, the dependent variable is bounded between 0 and 1 in this case as the target is binary.

First of all, we applied the fitting specifying only two fixed parameters:

- solver="liblinear", default 'lbfgs'. We decided to change it from default because we thought we could get a better score with liblinear. However we will test different solvers afterwards.
- random_state=42, default None. Same as before, we thought that shuffling the data we

could get a better result.

As a result, the highest accuracy (0.833019) was obtained with the eleventh model (normalized with dummies encoder). See Accuracy Table figure 13 in the annexes.

As we decided not to settle for that result, we applied hyperparameter tuning using GridSearchCV() function on this eleventh model.

grid_values =

- 'penalty': ['l2', 'l1', 'elasticnet'], default 'l2'. With 'elasticnet' both L1 and L2 penalty terms are added.
- 'C': [0.01,0.1,1,10,100], default 1. Inverse of regularization strength, like in support vector machines, smaller values specify stronger regularization.
- 'solver': ['lbfgs', 'liblinear', 'sag', 'saga', 'newton-cholesky'], default 'lbfgs'. Algorithm to use in the optimization problem.
- 'max_iter': [100,400], default 100. Maximum number of iterations taken for the solvers to converge.
- 'random_state': [0, 42], default 0. Used to shuffle the data.

After 50 minutes of executing it, we achieved a better model than the original, specifying the following parameters:

- C = 1,
- max_iter = 100,
- penalty = 'l1',
- random_state = 0,
- solver = 'liblinear'.

Obtaining an accuracy of 0.8346247622053726.

6.3 Gaussian Naive-Bayes

A GaussianNB model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters.

In this case, the only parameters that are available to change in the model are 'priors', 'var_smoothing' but we decided not to apply gridsearch and use only the default values for the twelve data sets (see the Accuracy Table figure 14 in annexes) and the highest result was 0.771629 using the twelveth data set (normalized without categories and with a target encoding).

6.4 Support Vector Machine

Support Vector Machine are a set of supervised learning methods used, in this case, for classification but can also be applied to regression and outliers detection.

As the fitting takes much more time to execute than the rest of the models, we maintained the default values of the parameters.

The results obtained can be seen in the Accuracy Table figure 15 in the annexes section. The highest value was 0.819212 using the first model with categories and dummies encoder. However, we understand that the selection of the parameters may not be the most suitable for the different data sets. Therefore, we decided to apply GridSearch hyperparameters tunning, testing the following changes and an explanation of why we decided to test different values:

- 'C': [0.1, 1, 10], the regularization parameter helps us when we have noisy observations so if we decrease C we would have more regularization. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly. In our case we cleaned the dataset before fitting the model but we decided to try these three values;
- 'degree': [1, 2, 3], this parameter only affects the 'poly' kernel and is ignored on the rest of them so, we considered necessary to check whether increasing to a second degree will improve or not the accuracy when the kernel is poly;
- 'gamma': [0.1, 1, 10, 'scale', 'auto'], defines how much influence a single training example has. The larger gamma is, the closer other examples must be to be affected. In other words, is the kernel coefficient that applies only for rbf, poly and sigmoid kernels. Quoting the scikit learn library web page "if gamma='scale' (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma. If auto, uses 1 / n_features".
- 'kernel': ['linear', 'poly', 'rbf', 'sigmoid'], quotting scikit learn linear: $\langle x, x'\rangle$; polynomial: $(\gamma\langle x, x'\rangle + r)^d$, where d is specified by parameter degree, r by coef0; rbf: $\exp(-\gamma\|x - x'\|^2)$, where γ is specified by parameter gamma, must be greater than 0; sigmoid: $\tanh(\gamma\langle x, x'\rangle + r)$, where r is specified by coef0;
- 'cache_size': [1000], to improve performance;
- 'class_weight': ['balanced', 'None'], as we are not using it in the unbalanced dataset, we probably should set this parameter to get a better result but we will try it on both options and see the results.
- 'max_iter': [1000], as we tried using the value -1 (default with no restriction on maximum iterations) but the running time was taking more than an entire day, we decided to set a threshold being aware that this could affect the results.

We applied this GridSearch on the first dataset, as it is the one having the highest accuracy score using the default values for SVM.

The results over the first dataset were:

- 'C': 10,
- 'cache_size': 1000,
- 'class_weight': 'balanced',
- 'degree': 1,
- 'gamma': 10,
- 'kernel': 'rbf',
- 'max_iter': 1000.

Achieving an accuracy of 0.771064, the results are worse than the original taking into account that we set the max_iter at 1000 and as we mentioned before, this could deteriorate our results. However, we can deduce from the previous analysis that the kernel that best fits our data is the Radial Basis Function (rbf, same as default). Also, in this case is better to change both the C and gamma parameters from the default ones, to 10 both. Additionally, as we expected, the 'balanced' class_weight give us better results than the default because we are using an unbalanced dataset.

Then, we decided to apply Custom Kernels. First, we redefined the linear kernel with the maximum number of iterations being 1: we obtained an accuracy of approximately 0.4994100.

After that, we decided to directly to apply the Radial Basis Function (RBF) kernel as it was the one giving the best results according to the Grid Search. With 1 as a maximum number of iterations, we obtained an accuracy of 0.827000. The results for this kernel are much better as expected.

6.5 Random Forest Classifier

Another approach that we decided to try is the Random Forest Classifier, which is a set of decision trees from a randomly selected subset of the training set. It aggregates the votes from different decision trees to decide the final class of the test object.

Same as before, we did not change the default values of the parameters and test it over the twelve different datasets, achieving 0.821336 of accuracy using the second dataset (no categories with dummy encoding). The results obtained can be seen in the Accuracy Table figure 16 in the annexes section.

Once we obtained the results using the default parameters, we will use again GridSearch to improve them:

- 'criterion': ['gini', 'entropy', 'log_loss'], default 'gini'. Used to measure the quality of a split.
- 'max_features': ['sqrt', 'log2', None], default 'sqrt'. Represents the size of the random subsets of features to consider when splitting a node. The lower the greater reduction of variance but also increase in bias.

- 'oob_score': [True, False], default False. Whether to use out-of-bag samples to estimate the generalization score.
- 'n_jobs': [None, -1], default None. The number of jobs to run in parallel. fit, predict, decision_path and apply are all parallelized over the trees. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors.
- 'class_weight': [None, 'balanced', 'balanced_subsample'], default None. Weights associated with classes in the form class_label: weight. If not given, all classes are supposed to have weight one.

After 47 minutes, we achieved a set of parameters that improve our accuracy score (remaining the rest as default), using:

- 'criterion': 'entropy',
- 'max_features': None,
- 'oob_score': True,
- 'n_jobs': -1,
- 'class_weight': None,

The resulting accuracy was 0.8247510650955782 (not a huge improvement, but still a better model).

6.6 XGBoostClassifier

XGBoost is currently the dominant algorithm for building accurate models with standard tabular data. It is an implementation of the Gradient Boosted Decision Trees algorithm. Basically, we go through cycles which repeatedly builds new models and then we combine them into an ensemble of model.

First, we applied the model with the default values, to set the ground flour and check the initial accuracy for each data set. We achieve 0.840807 for the second and eventh data sets (no categories with dummy encoding and normalised with dummy encoding). The results obtained can be seen in the Accuracy Table figure 17 in the annexes section.

After obtaining those results, in this case, we performed Bayesian Optimization with HYPEROPT to get the parameters that work best with our dataset (we selected the eleventh dataset because it is the one that worked best on the other models and with the highest score using the default parameters). We define as space, the parameters we wanted to test (some we remained as default values), which could be divided into three main categories:

- General Parameters:
  - 'verbosity': 0, default 1.
- Parameters for Tree Booster
  - 'gamma': (0, 10), default 0. Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative

the algorithm will be.

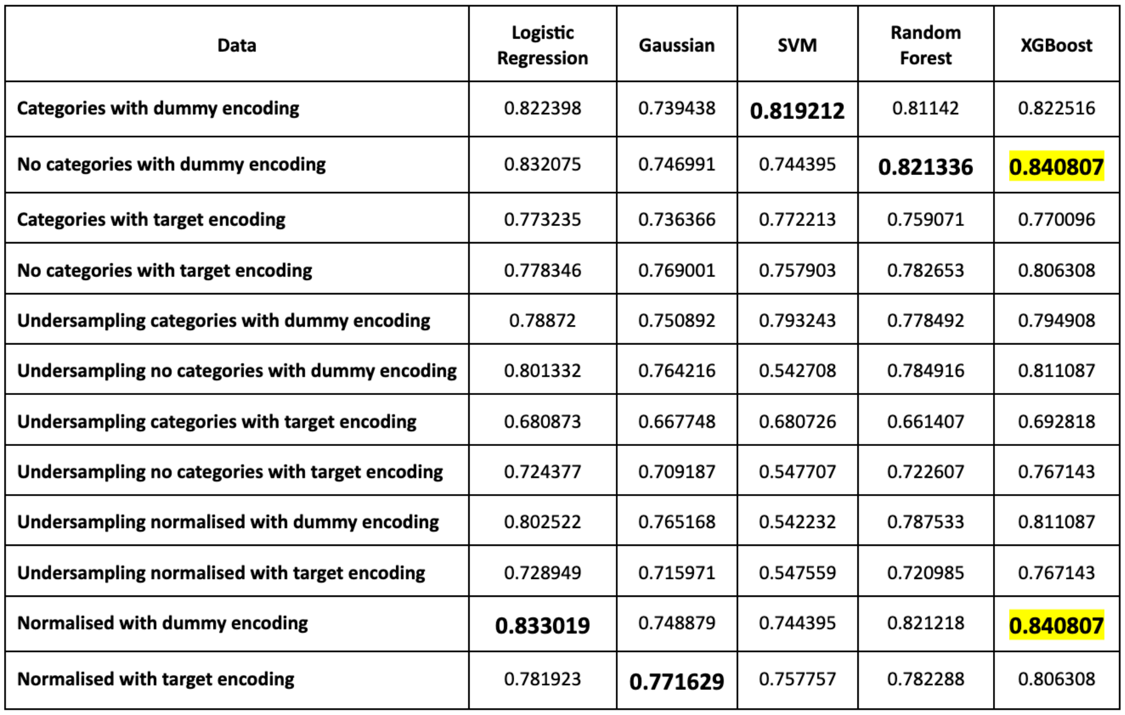
  - 'max_depth': (2, 6), default 6. Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. 0 indicates no limit on depth.
  - 'min_child_weight': (1, 5), default 1. Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. In linear regression task, this simply corresponds to minimum number of instances needed to be in each node. The larger min_child_weight is, the more conservative the algorithm will be.
  - 'max_delta_step': (0, 3), default 0. It refers to the maximum delta step we allow each leaf output to be. If the value is set to 0, it means there is no constraint. If it is set to a positive value, it can help making the update step more conservative.
  - 'subsample': (0.5, 1), default 1. Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees. and this will prevent overfitting.
  - 'lambda': (1, 2), default 1. L2 regularization term on weights. Increasing this value will make model more conservative.
  - 'alpha': (0, 1), default 0. L1 regularization term on weights. Increasing this value will make model more conservative.
  - 'num_parallel_tree': (1, 2), default 1. Number of parallel trees constructed during each iteration. This option is used to support boosted random forest.
- Learning Task Parameters
  - 'objective': 'binary:logistic',
  - 'seed': 123 default 0. Random number seed.

The higher accuracy obtained with this model is 0.842813 after the Grid Search (as well not a huge improvement, but here it remains important as it is the highest accuracy score we obtained).

7 Conclusion

In this project, the main objective was to obtain the best prediction of one person's salary having several of his/her characteristics. In order to achieve this goal, we really tried to maximize the number of models computed by applying several machine learning models: Logistic Regression, Gaussian, Support Vector Machine, Random Forest Classifier and XGBoost, on various datasets: some normalized, with different encoders, with categorical values and even with under sampling.
In total, 60 different models were computed.

The first approach when executing each machine learning model was to set basic parameters to have a general idea of which models could work and on which datasets. We obtain the results sum up in the following table 11.



Data	Logistic Regression	Gaussian	SVM	Random Forest	XGBoost
Categories with dummy encoding	0.822398	0.739438	**0.819212**	0.81142	0.822516
No categories with dummy encoding	0.832075	0.746991	0.744395	**0.821336**	**0.840807**
Categories with target encoding	0.773235	0.736366	0.772213	0.759071	0.770096
No categories with target encoding	0.778346	0.769001	0.757903	0.782653	0.806308
Undersampling categories with dummy encoding	0.78872	0.750892	0.793243	0.778492	0.794908
Undersampling no categories with dummy encoding	0.801332	0.764216	0.542708	0.784916	0.811087
Undersampling categories with target encoding	0.680873	0.667748	0.680726	0.661407	0.692818
Undersampling no categories with target encoding	0.724377	0.709187	0.547707	0.722607	0.767143
Undersampling normalised with dummy encoding	0.802522	0.765168	0.542232	0.787533	0.811087
Undersampling normalised with target encoding	0.728949	0.715971	0.547559	0.720985	0.767143
Normalised with dummy encoding	**0.833019**	0.748879	0.744395	0.821218	**0.840807**
Normalised with target encoding	0.781923	**0.771629**	0.757757	0.782288	0.806308

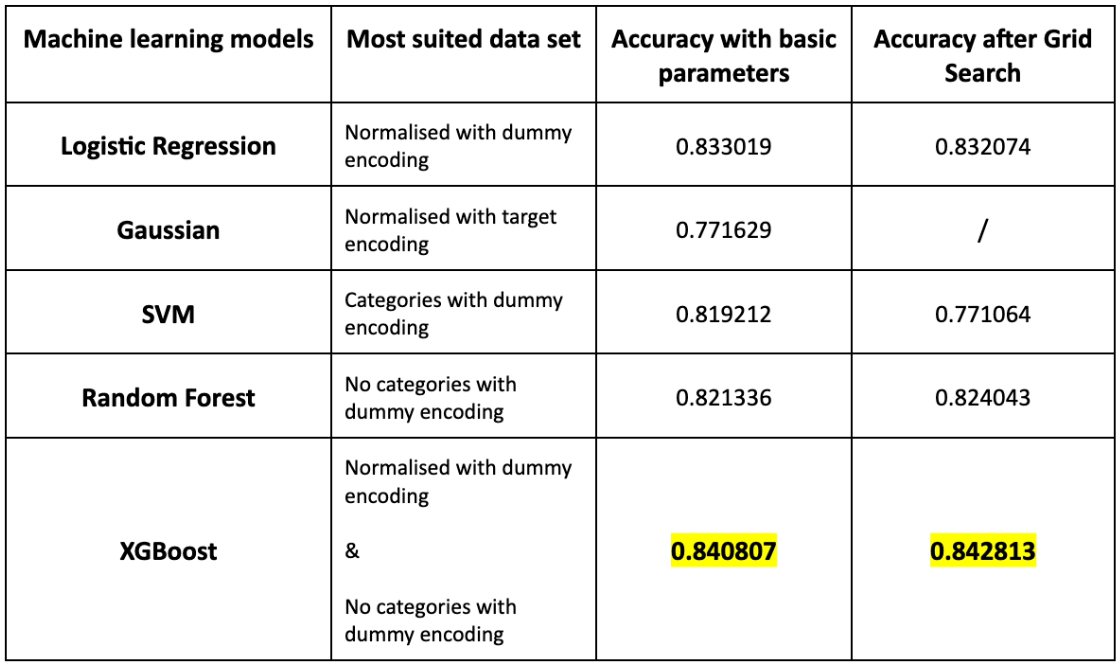
Figure 11: Accuracy Matrix for each model without hyperparameter tunning

This shows that the best model was obtained with XGBoost for the two datasets: without under sampling, but with normalization and dummies encoder and without categories and dummies encoder.

However, we decided to go further by applying Grid Search to find the best parameters for each machine learning model (as some results are very tight, changing one parameter could completely change our conclusion) and make sure that XGBoost is the best choice.
We obtained the results in the table 12.

Finally, XGBoost was still the best machine learning model when applying it on the fourth and eleventh data sets (same as before) with the parameters found with grid search.

So, we apply XGBoost to the eleventh data set and do the prediction on its test data set by fitting its training data set.
The results are coherent as we obtained an accuracy of approximately 0.834, close to the one computed when applying the XGBoost model with only the learn and validation dataset.



Machine learning models	Most suited data set	Accuracy with basic parameters	Accuracy after Grid Search
Logistic Regression	Normalised with dummy encoding	0.833019	0.832074
Gaussian	Normalised with target encoding	0.771629	/
SVM	Categories with dummy encoding	0.819212	0.771064
Random Forest	No categories with dummy encoding	0.821336	0.824043
XGBoost	Normalised with dummy encoding & No categories with dummy encoding	**0.840807**	**0.842813**

Figure 12: Accuracy comparison before and after grid

Therefore we can conclude that the model is not over/underfitting the new data given.

We can keep in mind an axe of improvement which could be to run the Grid Search for SVM without a maximum number of iterations, but in order to do this we need a more efficient software.

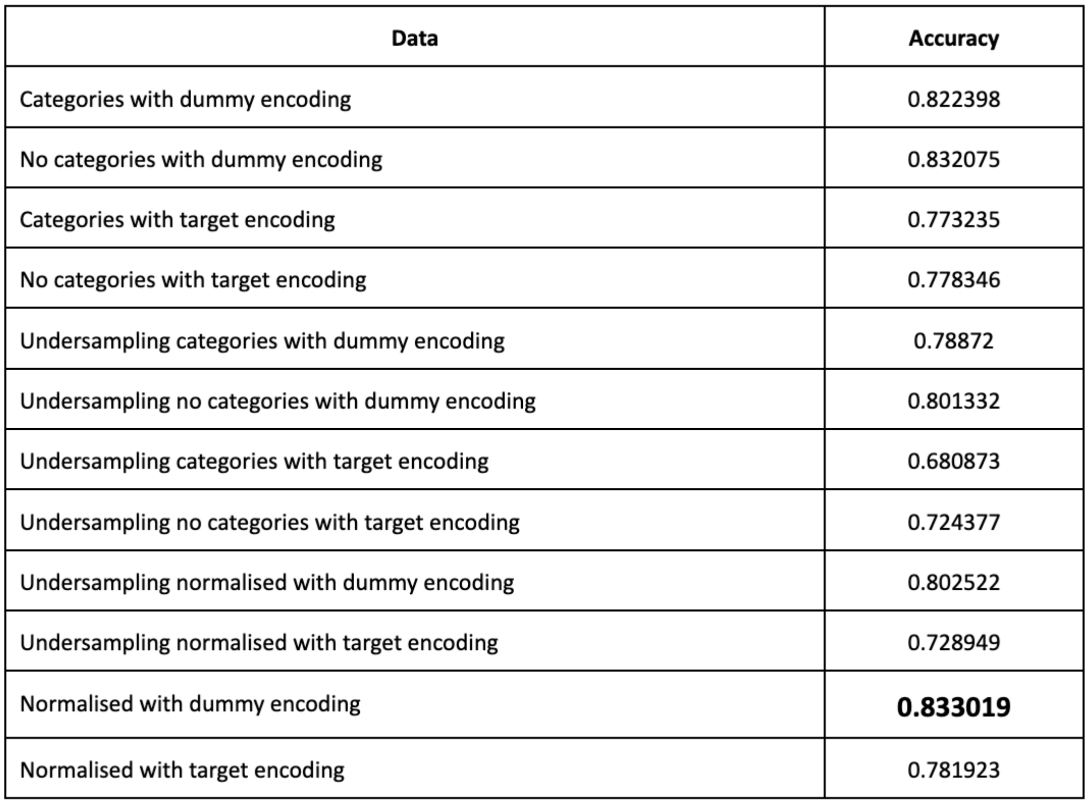
8 References

[1] Used dataset: https://www.kaggle.com/datasets/ddmasterdon/income-adult
Extracted by https://www.census.gov/en.html

[2] https://scikit-learn.org/stable/index.html

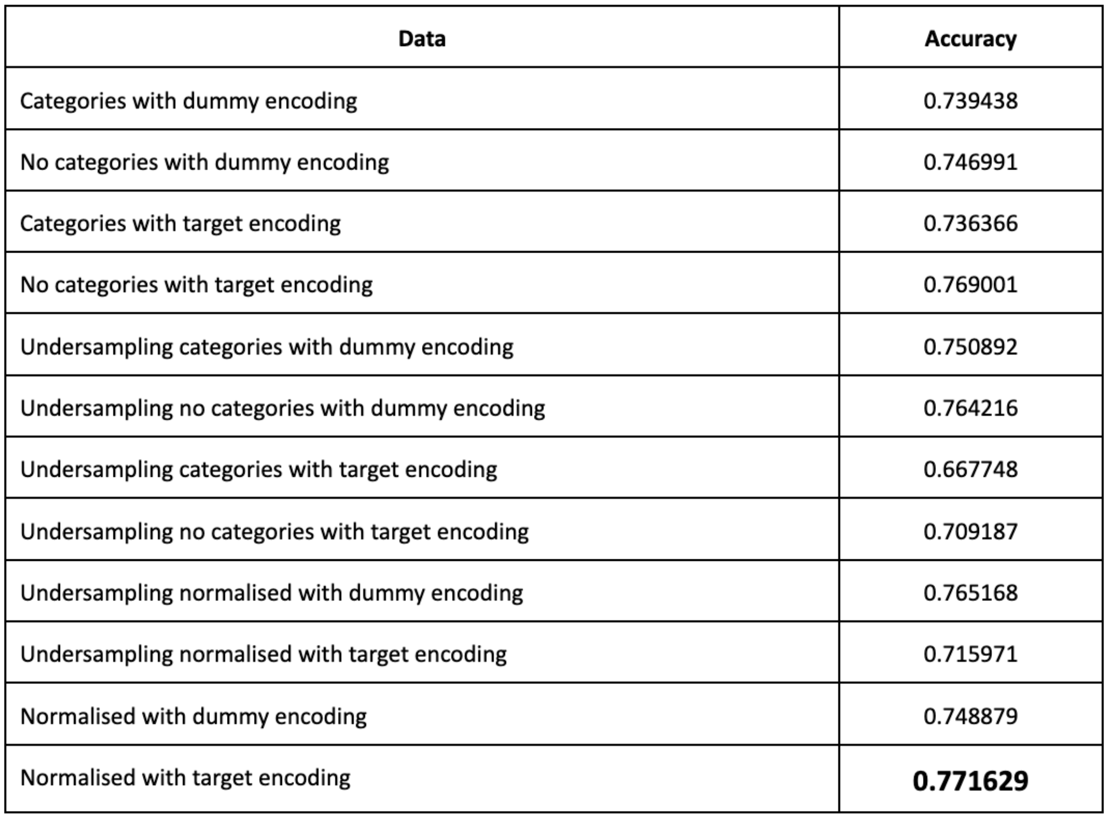
[3] https://stackoverflow.com/

9 Annexes



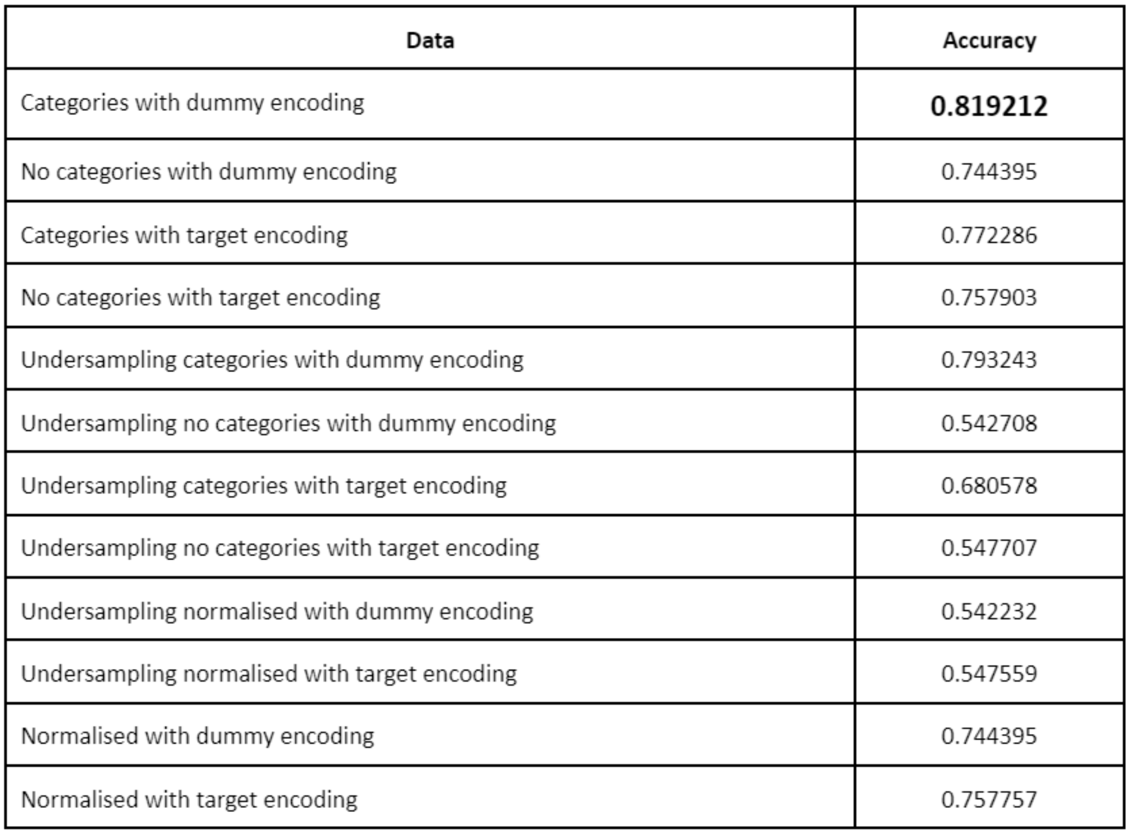
Data	Accuracy
Categories with dummy encoding	0.822398
No categories with dummy encoding	0.832075
Categories with target encoding	0.773235
No categories with target encoding	0.778346
Undersampling categories with dummy encoding	0.78872
Undersampling no categories with dummy encoding	0.801332
Undersampling categories with target encoding	0.680873
Undersampling no categories with target encoding	0.724377
Undersampling normalised with dummy encoding	0.802522
Undersampling normalised with target encoding	0.728949
Normalised with dummy encoding	**0.833019**
Normalised with target encoding	0.781923

Figure 13: Logistic Regression Accuracy



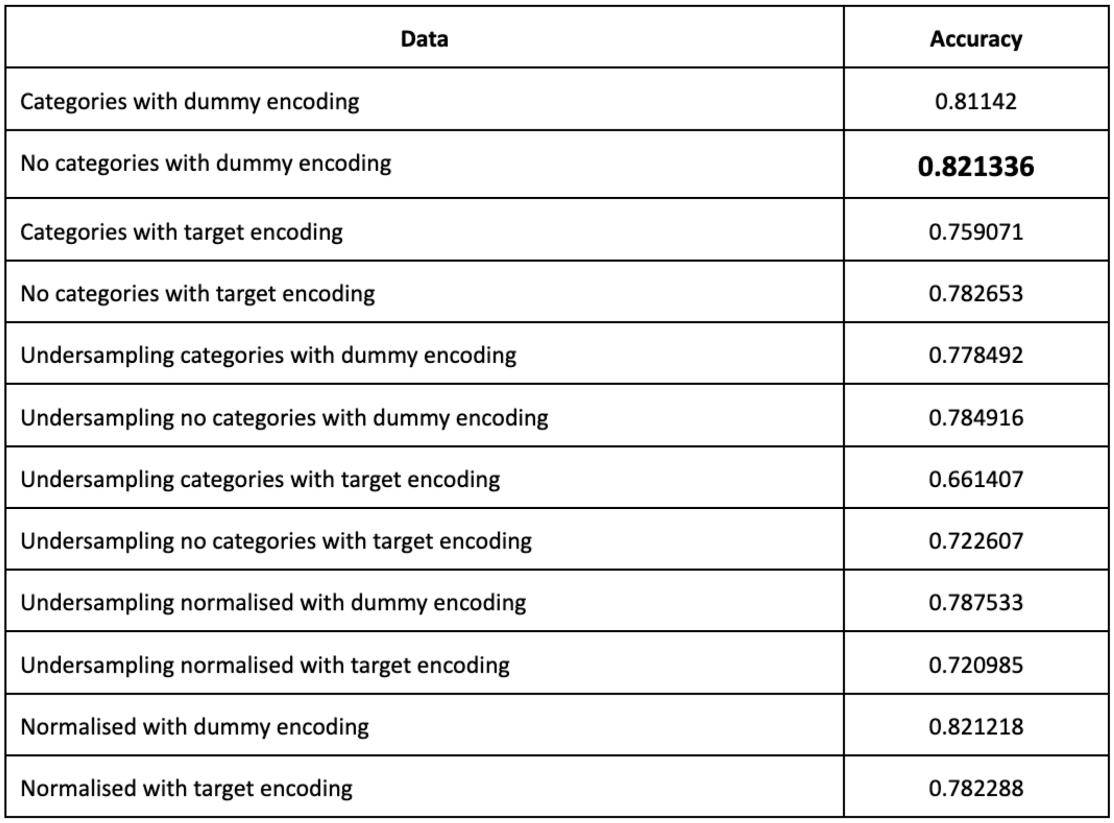
Data	Accuracy
Categories with dummy encoding	0.739438
No categories with dummy encoding	0.746991
Categories with target encoding	0.736366
No categories with target encoding	0.769001
Undersampling categories with dummy encoding	0.750892
Undersampling no categories with dummy encoding	0.764216
Undersampling categories with target encoding	0.667748
Undersampling no categories with target encoding	0.709187
Undersampling normalised with dummy encoding	0.765168
Undersampling normalised with target encoding	0.715971
Normalised with dummy encoding	0.748879
Normalised with target encoding	**0.771629**

Figure 14: Gaussian Accuracy



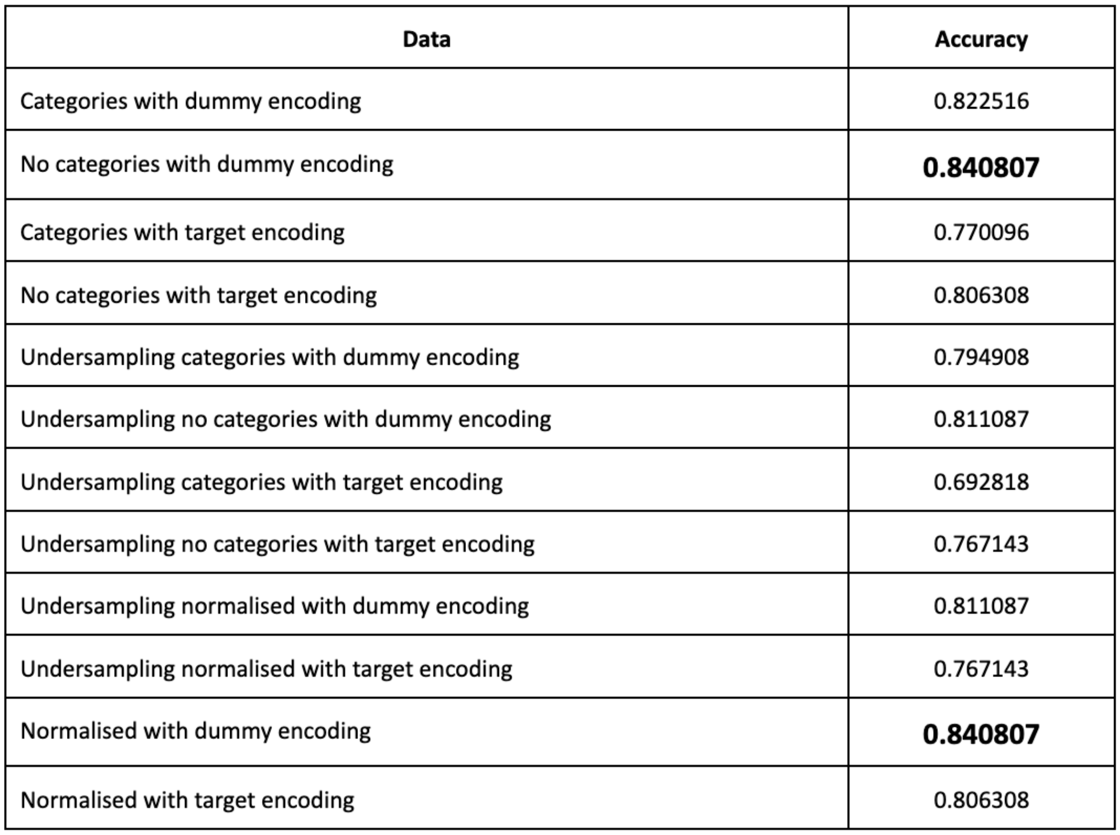
Data	Accuracy
Categories with dummy encoding	**0.819212**
No categories with dummy encoding	0.744395
Categories with target encoding	0.772286
No categories with target encoding	0.757903
Undersampling categories with dummy encoding	0.793243
Undersampling no categories with dummy encoding	0.542708
Undersampling categories with target encoding	0.680578
Undersampling no categories with target encoding	0.547707
Undersampling normalised with dummy encoding	0.542232
Undersampling normalised with target encoding	0.547559
Normalised with dummy encoding	0.744395
Normalised with target encoding	0.757757

Figure 15: Support Vector Machine Accuracy



Data	Accuracy
Categories with dummy encoding	0.81142
No categories with dummy encoding	**0.821336**
Categories with target encoding	0.759071
No categories with target encoding	0.782653
Undersampling categories with dummy encoding	0.778492
Undersampling no categories with dummy encoding	0.784916
Undersampling categories with target encoding	0.661407
Undersampling no categories with target encoding	0.722607
Undersampling normalised with dummy encoding	0.787533
Undersampling normalised with target encoding	0.720985
Normalised with dummy encoding	0.821218
Normalised with target encoding	0.782288

Figure 16: Random Forest Classifier Accuracy



Data	Accuracy
Categories with dummy encoding	0.822516
No categories with dummy encoding	**0.840807**
Categories with target encoding	0.770096
No categories with target encoding	0.806308
Undersampling categories with dummy encoding	0.794908
Undersampling no categories with dummy encoding	0.811087
Undersampling categories with target encoding	0.692818
Undersampling no categories with target encoding	0.767143
Undersampling normalised with dummy encoding	0.811087
Undersampling normalised with target encoding	0.767143
Normalised with dummy encoding	**0.840807**
Normalised with target encoding	0.806308

Figure 17: XGBoost Accuracy