École Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise
Université de Paris-Saclay

Fraud detection in financial transactions

Comparison of classification methods

Théophane Gué Antoine Pascalet–Gradolatto Lauriane Schell Richard Szeberin

17/03/2021

Contents

0.1 Introduction

Data Analysis is a trending branch of maths nowadays even though it has been around since decades. But with the explosive growth of technologies and the computing abilities of the newest computer, it's something to be reckon with. The shear amount of data make them improcessible by a human, especially when it comes to look for a needle in a haystack such as looking for anomalies or -in our case - frauds.

That's where data science takes the lead, proposing several tools, methods and other algorithms to process those data. Our project is on point with that purpose as this project was to build an algorithm capable of sorting out frauds from datas with the highest accuracy possible, datas that were overwhelmingly huge, on key sector which e-transaction is.

The data where provided by Vesta Corporation via the website http://kaggle.com for the benefits of IEEE Computational Intelligence Society and the site were the people evaluating our performance.The datas were logs of e-transaction, each labelled as either fraudulent or or clean.

The "boosting" method was prohibited for diversity purpose. So, all things considered, how did we conceive our algorithm and which model did we use? This rapport will first detail the data analysis,then different transformations used to process columns then the differents predictions methods considered and used. The language we use is python.

Here are our kaggle pseudos :
- laurianelolo35
- antoinepasto
- richrdszeberin
- theophanegue

You will find our codes in the same repertory as this report :
- Data_analysis.ipynb : for the analysis of data
- cleaned_data.ipynb : this one regroups all the transformations we made on our datasets in order to have proper data.
- methodes.ipynb : here you will find the code related to the method we performed on our cleaned datasets.
- Feature_Engineering_and_RF.ipynb : the second way to transform and clean data with our best result.

0.2 Analyse des données

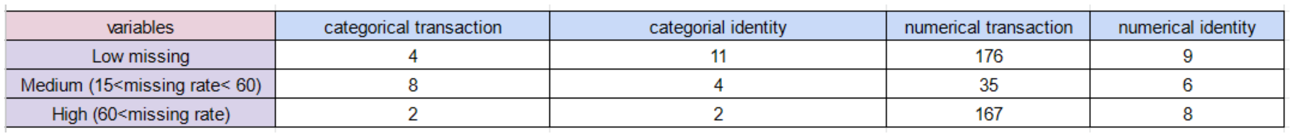
0.2.1 First impressions

The whole point of the project was to detect fraud, ie. to detect anomalies. That's why we decided to used different methods of approach, including SVM,KNN,Random Forest and a more classic Logistic Regression. To have our way more conveniently with the data, we had to process them.

The Fraud detection data consists of transaction and identity datasets, both of them divided into train and test sets. Transaction and identity data is linked by TransactionID, a variable which uniquely identifies each transaction.We first had nearly 400 hundred variables over 500 000 entries for the sole test transactions datas and about the same amount for the train transaction datas.The train identity and test identity are both around 140k example for 40 variables.So we first tried to thinned out those huge blocks. As we started the basic exploration of the dataset, we found several interesting properties:

1. First, we found that most transactions do not have corresponding identity data.

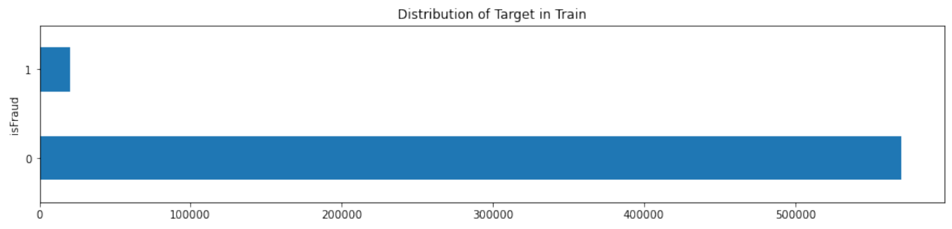
2. Furthermore, we looked for the proportion of missing values in the dataset. We discovered a significant amount of NaNs even before merging data, which increased further due to the fact that identity data is much sparser than transaction data.In our train data, we identified 41.2% of transaction data and 36.5% of identity data as missing. In our merged data, we have found that over 47% of features have more than 70% missing values. As these proportions are very significant, we decided to look for the best imputation strategy, which will be discussed in a later chapter.



variables	categorical transaction	categorial identity	numerical transaction	numerical identity
Low missing	4	11	176	9
Medium (15<missing rate< 60)	8	4	35	6
High (60<missing rate)	2	2	167	8

Figure 1: Rate of missing value

3. Regarding the distribution of the target variable, we noticed that the dataset is heavily skewed. Only 3.5% of transactions are labeled as fraud, thus, it is important to focus on avoiding overfitting during model training.



0.2.2 Variable analysis

We observed and identified true numerical and categorical features. The description of these features was provided by Vesta, which helped us to truly understand the variables.

Information of Variables

According to the Data Description given by the data provider Vesta: https://www.kaggle.com/c/ieee-fraud-detection/discussion/101203

Variable Type	Numerical	Categorical
Transaction	TransactionDT TransactionAMT dist1,dist2 C1-C14 D1-D15 V1-V339	ProductCD card1-card6 addr1,addr2 Pemaildomain Remaildomain M1-M9
Identity	id01-id11	id12-id38 DeviceType, DeviceInfo

Transaction features

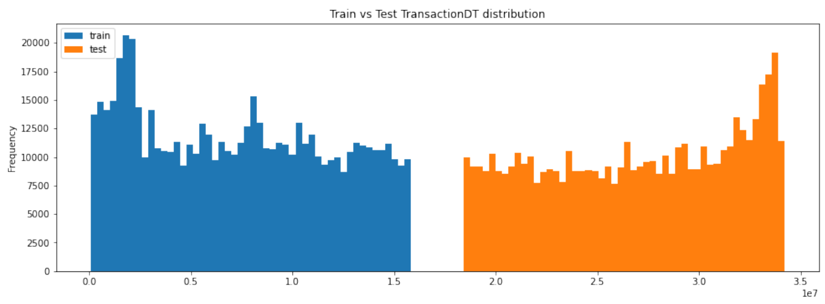
TransactionDT

The TransactionDT feature is a timedelta from a given reference datetime (not an actual timestamp). One early discovery about the data is that the train and test appear to be split by time. There is a slight gap inbetween, but otherwise the training set is from an earlier period of time and test is from a later period of time. This will impact which cross validation techniques should be used.

The unit of timedelta is unknown, but if we assume that it is in seconds, we can compute that

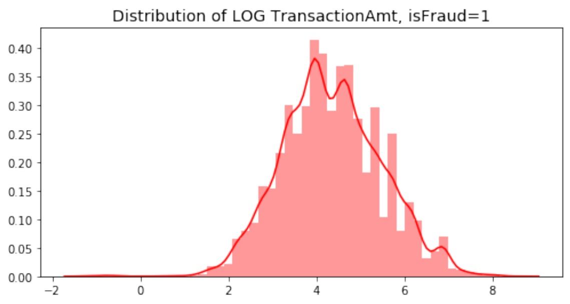
- Time span of the total dataset is 394.9993634259259 days
- Time span of Train dataset is 181.99920138888888 days
- Time span of Test dataset is 182.99908564814814 days
- The gap between train and test is 30.00107638888889 days,

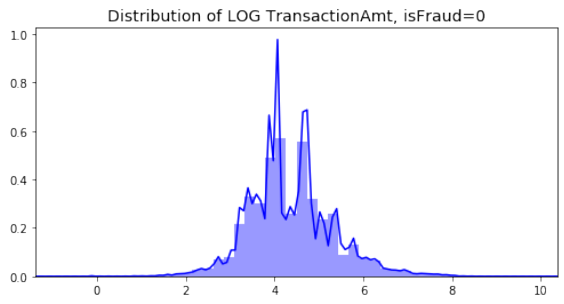
which seems reasonable.



TransactionAMT

This variable represents the amount of transactions in USD dollars. We have taken a log transform in some of these plots to better show the distribution - otherwise a few, very large transactions skew the distribution. Because of the log transfrom, any values between 0 and 1 will appear to be negative.



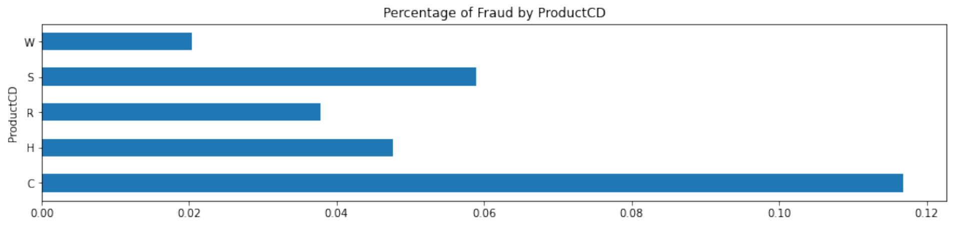


Also, we analyzed mean transaction amounts for fraudulent and non-fraudulent transactions:

- Mean transaction amt for fraud is 149.2448
- Mean transaction amt for non-fraud is 134.5117

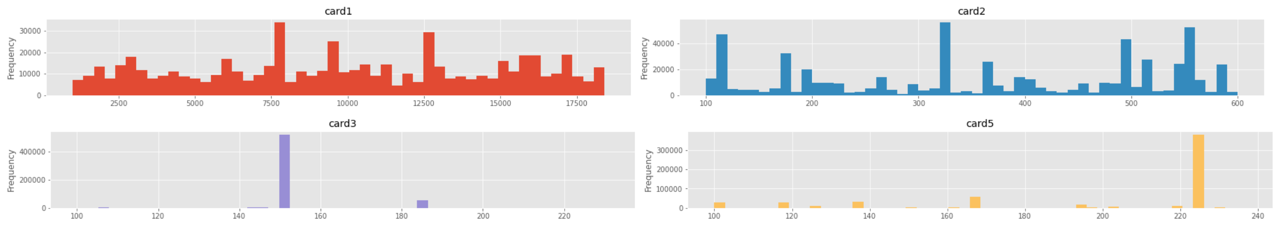
<u>ProductCD</u>

This is a categorical feature that represents product codes, and potentially aims to distinguish between different purchases. We observed that value W has the largest share in this feature while C has the least. On the other hand, the observations with value C contain 11% fraud transactions and those with W have 2%.

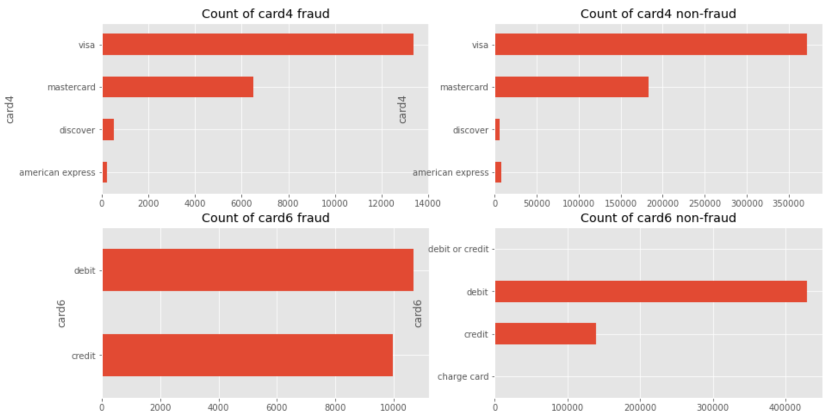


<u>card1 - card6</u>

These features contain card-related information and while they are categorical based on the description, some of them (card1 and card2) seem to be numerical based on their distribution.



Variables card4 and card6 labels observation based on card provider and card type, respectively. While the card provider does not seem to heavily influence the target variable, credit card transactions are linked with higher fraud proportions.

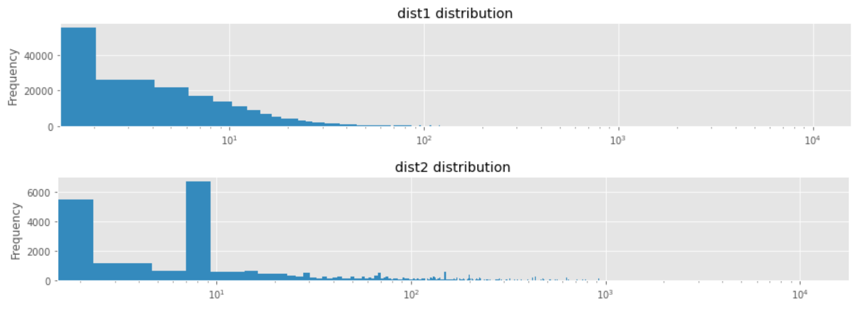


addr1_addr2

The data description states that these are categorical even though they look numeric. Perhaps they represent country (addr1) and region (addr2) codes. In addr2, the value 87 accounts for 88% of the observations and NaNs for a further 11%.

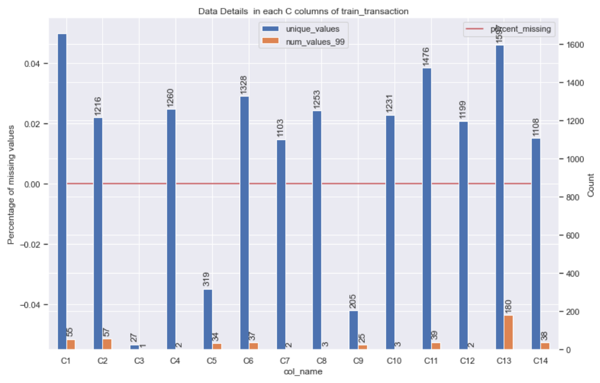
dist1_dist2

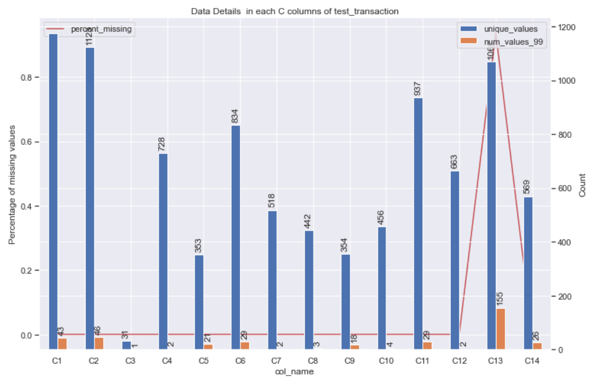
These variables could be the distance of the transaction vs. the card owner's home/work/IP/phone/etc. address. These log scale plots show that smaller values of these features are more frequent.



C1-C14

These are numerical columns associated with addresses, but we are not provided with the exact meaning behind the variables. We discovered that none of the C columns contains missing values in the training data, but this does not hold for the test data set, where C13 has more than 90% missing data and C14 has over 20%. These features are potentially to be dropped later.



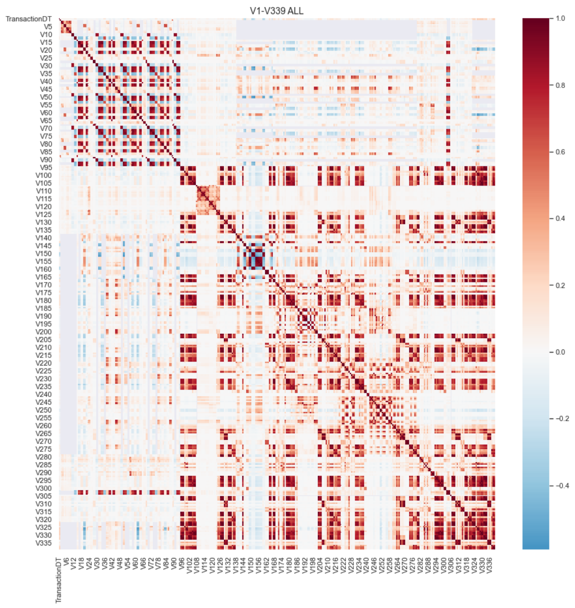


D1-D15

The D columns are additional temporal features, containing various timedeltas, such as time since the previous transactions. Most of the D columns have more than 90% null values.

V1 - V339

A very large portion of our data are Vesta engineered rich features, including ranking, counting, and other entity relations. The true meaning behind these columns is unknown, we analysed correlation between these variables. We notice that the first 100 Vs don't correlate much with the last 239 Vs. Also even though the first 100 have 6 different NaN groups there is much correlation between the groups. The latter 239 relate to each other.



0.3 Data Transformation

After observing the dataset we decided to make some transformations in order to apply some algorithms for the prediction part. There are two goals in the transformation process. The first one is to get the data ready for the algoritmhs we will use, which mainly means removing NaNs. The second one is to try to improve the quality of our data with other methods to get better results. Here are the transformations we make.

First of all we save the column "isFraud" in a variable Y and drop the original one off the train_transaction dataset. Because Y is already in a binary form, with no missing value, there is no need to transform it.

0.3.1 Rename columns

During the analysis phase, we noticed that the columns of the test datasets do not have the same names than in the train datasets. So first we change the "-" in the column's names of test dataset for "_".

0.3.2 Treat NaN values

Concatting datasets

In order to treat our data the best way possible, we concat our four datasets into two, one with all the transaction data, the other with the identity data. Of course, we save the shape of the test / train datasets so we can recreate them later. This process will allow us to be more accurate while treating NaNs, since we will be working on all the data we have for each variables.

Separate categorial values from numeric ones

Before solving the problems of NaN values we need split the variables in 2 categories : the numerical ones and the categorical ones. We will treat them differently.

Splitting of variables depending on the number of missing values

We split each set of data (numerical and categorical one) in three other sets (so we have 6 sets at the end). This splitting is based on the number of missing values, we make the limits like this :
- lower than 15% missing values (category one)
- between 15% and 60% missing data (category 2)
- more than 60% missing data (category 3)

We decided to drop the variables from category 3 off the datasets since they do not have enough values to be used.

Then for numerical data from category 1 we imputed the NaNs values by the mean (why Daemon) of the columns. Meanwhile for the category 2 we used the median.

Now that the numerical features are delt with, we need to focus on categorical ones. There are two steps for these features. The first one is to impute the NaNs, and the second one to encode them into numerical values So first we need to replace all the NaNs by the most frequent value.
Then we split the categorical features into 2 other datasets depending on their cardinal (the number of unique values they can take). For those which have a huge cardinal, we change the value by a number in the range of their cardinal.

For those which have a smaller cardinal, we create a new binary columns for each possible values for the features, it is the one-hot-encoding method. Then we need to drop one column for each initial feature so that there is less correlation between our remaining variables.

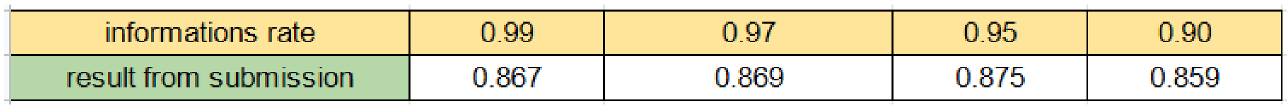
0.3.3 Concatenation to create a tran dataset and a test dataset

Now that we do not have any missing value in our datasets anymore, we recreate our train and test datasets with their original shape. Now we need to concat our training data as well as our testing data so we can feed them to our algorithms. By concatting transaction and identity features, we are creating new NaNs because there is not the same number of rows in the initial datasets. So we impute them with the most frequent value for initial categorical features, and with median for the numerical ones.

Now that we have our two datasets for training and testing with no NaNs, we are almost ready to use our algorithms. But for them to have the best results possible, we need to scale the data so that a feature with high values can not overwhelm the others.

0.3.4 PCA

To conclude our transformations we compute a principal component analysis. Our goal with this is to reduce the size of the dataset and limit the correlation between our features.
The PCA will create new columns which came from the combination of columns of the original dataframe. With it it will delete columns which are correlated.
Then with the variance we can see the part of information contained in each columns.
Our tests show that with the first 165 columns of our dataset we have 99% of the information, with 135, 97% , with 115, 95% and with 90, 90%. We can see that the main part of the information is located in the first variables.
Here are the sum up of our tests. We stopped at 90% because after we we are loosing too much information.



informations rate	0.99	0.97	0.95	0.90
result from submission	0.867	0.869	0.875	0.859

Figure 2: Result for the algorithm random forest with more of less data

We can see here that we have better results with 97% of our data than with 99%. We can explain it by overfitting of correlations between variables. So for all our submissions we will only take the 135 first variables of our data set, which correspond to 97% of the data.

Now, our datasets are ready to be used by algorithms.

0.4 Prediction method

The problem we treat is the detection of anomaly. In fact as we saw in the analysis of data there are only 3,5% of fraud in our dataset. So we looking for extrem case and not for equal group. That's why we used methods who are known as relevant for anomaly detection.

0.4.1 Logistic Regression

Parameters which changed

The Logistic regression model is a supervised learning model which is used to forecast the possibility of a target variable. The dependent variable would have two classes, or we can say that it is binary coded as either 1 or 0, where 1 stands for the Yes and 0 stands for No. It's a very famous and basic metohd in machine learning, that's why we decided to use it.

We had a result of 0.781.

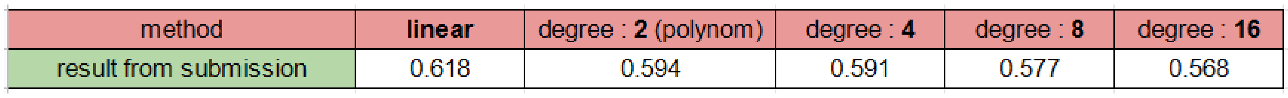
0.4.2 SVM

We decided to apply the Support Vector Machine on our data set. The SVM method is a supervised learning method which is very usefull to discrimination and regression problem. This method can efficiently perform a non-linear classification using what is called "kernel trick". In this method determine the maximum margin and the good kernel is very important. The role of the kernel is to find a border which separate quit well the classes. And the margin is here to have a marging of error which allow to not have overfitting.

In order to do that we used the module *SVC* from *sklearn.svm* library. We perform this method with different parameters.

Parameters which changed

We try differents parameters. For this method the kernel we chose is very important. We decided to try the linear kernel and the polynom kernel. For the polynome one we chosed different degree. Here are the sum up of our result :



method	**linear**	degree : **2** (polynom)	degree : **4**	degree : **8**	degree : **16**
result from submission	0.618	0.594	0.591	0.577	0.568

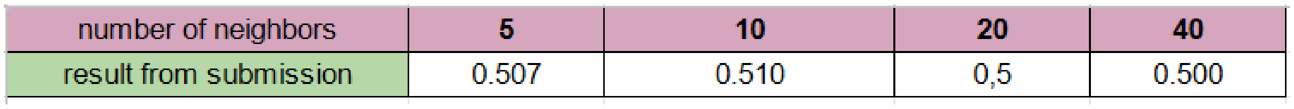
Figure 3: Result for the SVM with different kernel

0.4.3 KNN

The k-Nearest neighbors is a non parametric classification method used for regression and classification. That's why we decided to used it. The goal of the method is finding the class of an object basing on the k nearest neighbors. So the parameters which can be change is the number k of neighboors.

Parameters which changed

So we compute the KNN method with differents k. We used in each case the modul *KNeighborsClassifier* of the library *sklearn.neighbors*. Here are we results :



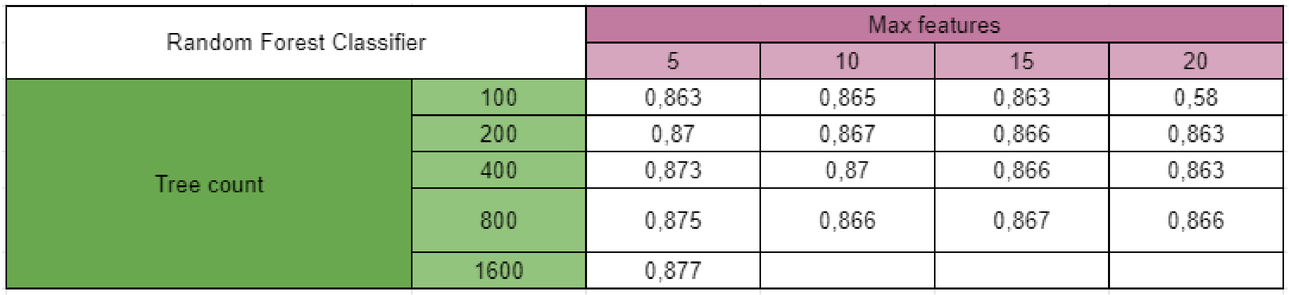
number of neighbors	**5**	**10**	**20**	**40**
result from submission	0.507	0.510	0,5	0.500

Figure 4: Result for the KNN method with different number of neighbors

0.4.4 Random Forest

The Random Forest algorithm is a frequently used ensemble learning classifier that uses multiple decision trees to obtain better prediction results. As the model's prediction performance is depending on multiple parameters, we tried several options before selecting the final settings.

To implement this model, we used RandomForestClassifier from the package scikit-learn. From the parameters of RandomForestClassifier, we focused on n_estimators (the number of decision trees in the forest, and max_features (the maximum number of features considered for splitting a node). We tried several combinations and obtained the following results for label encoded data.

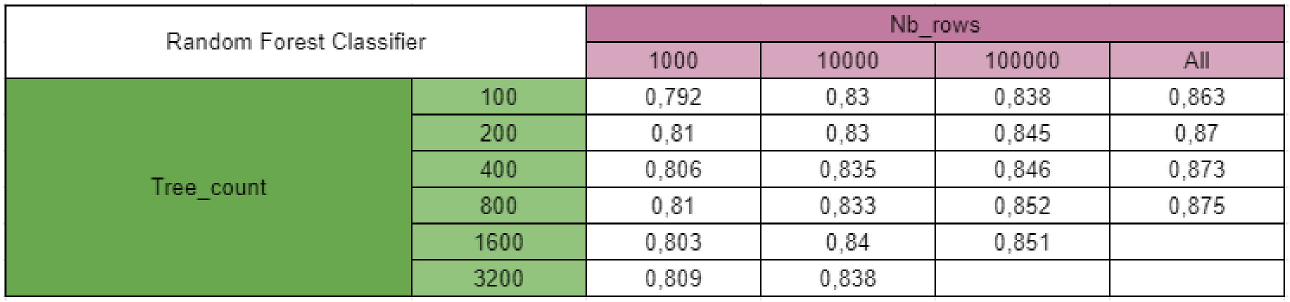


Random Forest Classifier		Max features			
		5	10	15	20
Tree count	100	0,863	0,865	0,863	0,58
	200	0,87	0,867	0,866	0,863
	400	0,873	0,87	0,866	0,863
	800	0,875	0,866	0,867	0,866
	1600	0,877			

Figure 5: Max features and tree count experimentation

This table shows that a great number of features considered for splitting a node is not giving us better results. However a smaller number make the algorithm a lot faster, so we decided to keep 5 max_features.

Furthermore, we now know for sure that increasing the number of decision trees in the forest gives us better results. But increasing this number also increases a lot the execution time, about 1h for 1600 trees and 5 max features, and without having a great effect on the result. In order to experiment greater numbers of trees, we will try to reduce the number of data we use in our training dataset.



Random Forest Classifier		Nb_rows			
		1000	10000	100000	All
Tree_count	100	0,792	0,83	0,838	0,863
	200	0,81	0,83	0,845	0,87
	400	0,806	0,835	0,846	0,873
	800	0,81	0,833	0,852	0,875
	1600	0,803	0,84	0,851	
	3200	0,809	0,838		

Figure 6: Tree count for given number of rows

This tables shows us that we can not really reduce the number of data to use in our fitting process if we want the best result, even for a great number of trees. Therefore we will set our tree count to 1000, but not more since the execution time would be too long. Our best score with these settings was 0.88.

However, we tried another way to treat data at the end. We did the same transformations, but by creating the testing and training datasets first. With this new method, the PCA-engineered dataset performed worse in general, while the label encoded dataset outperformed the binary encoded one (0.88 prediction score).

Finally, we concluded that the Random Forest model, for 1000 trees and (surprisingly) 15 max features, was able to reach 0.91 prediction accuracy on Kaggle, therefore we decided to use it as our final, best performing model.