# Case study findings

## TASK 1

‘task.app.data.csv’ dataset (dataset 0) was left joined with ‘task.cb1.csv’ (dataset 1) on ‘key2’. The resulting dataset was left joined with ‘task.cb2.csv’ (dataset 2) on ‘key1’. I have used left joins, because only the dataset 0 has the response variable. Thus, I have added some extra data to the dataset 0 from the other 2 datasets.

Here are the sizes of the datasets. They are very different – dataset 0 is much smaller than dataset 1 and 2.

1) dataset 0

column count: 5

row count: 798

element count: 3990

2) dataset 1

column count: 169

row count: 14571

element count: 2462499

3) dataset 2

column count: 37

row count: 10137

element count: 375069

After the joins the final dataset has the following size:

dataset\_full - before cleaning NAs

column count: 209

row count: 798

element count: 166782

I remove some columns and rows that have too many missing values. For columns there should be at least 20% non-missing values, and for rows – 5% (later I will apply more aggressive filtering). And I get the following dataset:

dataset\_full\_clean - after some columns and rows with many missing values are removed

column count: 60

row count: 772

element count: 46320

After this clean-up, there are still 17 473 Nulls in the dataset (around 1/3 of total number of elements/cells).

### Conclusion about data integrity:

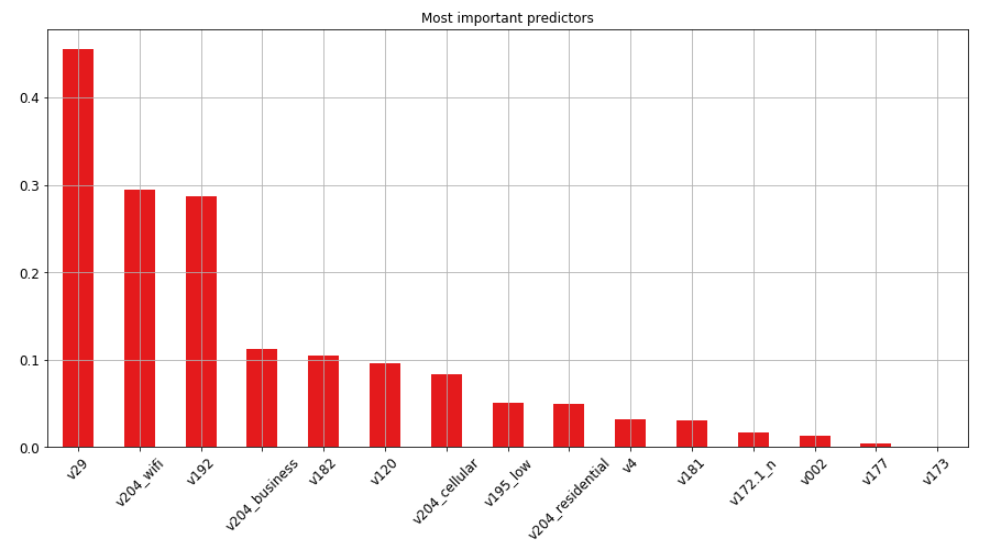
Only small portions of dataset 1 and 2 are used. Even for those parts that are used, there is a significant amount of missing values (NaNs or Nulls). This can really affect the performance of the model. With such data structure it is very important to have a good strategy of dealing with missing data.

## TASK 2

To determine the strongest predictors of target variable I apply 2 approaches:

1. To look at the correlations between each predictor and the target variable. It is a less robust approach, but it is still helpful to do it as an additional check.
2. Response variable is a binary variable (1 or 0). So, I have chosen Logistic Regression to estimate the importance of various factors. It is not the best ML-algorithm to get a high prediction accuracy, but it is very suitable for the task to determine most important factors. It provides coefficients for each predictor measuring their impacts on the target.
   1. To deal with collinearity and overfitting I have used regularization (L1-norm) that adds penalty on too high coefficients.
   2. I have performed some dimensionality reduction. I have removed redundant factors that have essentially the same correlation with the target as some other factors.
   3. To improve the performance of the model, I have optimized its parameters (such as regularization strength) with cross-validation.
   4. I have normalized all predictors before using Regression, thus the coefficients are comparable.

The graph below shows the most important factors:



The most important predictors are v29, v204, v192, v182, v120, and v195. These findings are consistent with the correlations between predictors and the target (see Table 1 in Appendix).

Because of similarity/redundancy I have excluded some factors earlier. Most importantly, I have excluded v191 (which is identical to v192). But, I could say that v191 is also an important regressor.

## TASK 3

I have used Random Forest Classifier model to do the predictions. It is a very powerful model both theoretically and empirically. It works well against overfitting. At the same time, it has only few parameters to optimize and they are not too sensitive to changes. So, it is relatively easy to calibrate them. Also, since it is based on a decision tree, there is no much need to create additional variables (like polynomials, interactions and different powers of existing factors), thus much less time is needed on feature engineering and there is a smaller risk that you can miss something important.

I have optimized all the parameters for the model using Cross-Validation and Grid-Search. The optimized model has the following parameters: max\_depth=26 (max tree depth), min\_samples\_split=10 (minimum number of observations in a split node), n\_estimators=50 (number of trees in the forest). The other parameters have their default values, like max\_features=’auto’=’sqrt’ (which says that the maximum number of features to consider at split should be a square root of the total number of features).

### 1. Performance of the model

5-fold cross-validation on the train-set gives the following results:

accuracies = [ 0.81451613 0.81451613 0.81451613 0.81300813 0.8442623 ]

mean accuracy = 0.820163762452

I train the model on the whole train-set and it gives the accuracy of 0.8995 when predicting target from the train-set. When I train on train-set and predict on test-set, then the accuracy is 0.8516.

Thus, the model still could be optimized, as it is overfitting a bit. But overall, the results are consistent from 0.81 to 0.85.

### 2. Comparison with other models

1. Firstly, it is good to see if the model performs better than simply predicting the majority class for all the predictions (in this case it implies simply predicting that each observation has a target of 0).

1-train[target].mean() = 0.8071

1-test[target].mean() = 0.8516

So, Random Forest model performs slightly better than majority class prediction.

1. Let’s compare with the scores of Logistic Regression from Task 2.

Logistic Regression gives the following result in 5-fold cross-validation on the train-set:

accuracies = [ 0.80645161 0.81451613 0.80645161 0.81300813 0.83606557]

mean accuracy = 0.815298611738

If I train the model on the whole train-set, then it gives the accuracy of 0.8233 when predicting target from the train-set. When I train on train-set and predict on test-set, then the accuracy is - 0.8516.

Thus, Random Forest model performs slightly better than Logistic Regression as well.

### 3. Conclusions

Random Forest model gives a higher accuracy than a majority class prediction model and than the logistic regression. However, I do not see a significant improvement, thus a higher focus should be put on feature selection and dealing with missing values. Also, it would be interesting to try some other models like XGboost or some SVM. It is a bit disappointing the model does not perform much better than majority class prediction. However, if we would focus not just on accuracy, but on other aspects like precision and recall, then perhaps it would be more useful. I will discuss various options to improve prediction accuracy in the next section.

## TASK 4

### 1. Better handling of missing values (Null)

The biggest issue with the used dataset is that it contains a lot of missing values. My approach was to (1) remove rows and columns that contain too many missing values. My criteria was – at least 5% of items in rows should be non-zero and at least 60% items in columns should be non-zero. I was much more careful with rows, because they contain target variable and removing that is not preferred. (2) In the case of string columns that contain classes (like ‘big’ and ‘small’) I have created dummy variables for each class, and dropped original columns. If there were any missing classes/categories, then a dummy variable was created for them as well. (3) For numerical columns, I have imputed missing values with a median for a given column (median is less affected by outliers than the average).

Instead of imputing with medians, a better approach would be to predict each missing value, based on the value of other features for the same observation. This also can introduce some bias, but it could be more precise and it does not lead to a decrease in variance as imputing with medians.

The package and models I have used were not able to handle missing values explicitly. However, there are models that can. For example, XGBoost model (special boosting algorithm) can handle Nulls treating them as a separate class/category for each feature. So, in this case it would be very valuable and interesting to check the prediction accuracy with such approach. In this case no rows and columns should be dropped and more information would be available for predictions.

### 2. Dealing with imbalanced dataset

Out of 798 observations, response variable is 0 in 645 observations, and it is 1 in 153 cases. It is not a very big disbalance, but it is possible that prediction accuracy would be better if I would deal with this imbalance. (a) The simplest approach is to randomly remove 492 rows where response variable is 0. This would result in a balanced dataset where we have 153 cases of response variable being 0 and 153 cases being 1. (b) A bit better approach would be to put more weight on observations where response is 1. Each such observation would weigh 4.2 (645/153). (c) Employ some of the many other approaches of dealing with imbalanced dataset.

I have used some parameters to deal with imbalanced set in my models (In Random Forest and Logistic Regression - class\_weight parameter). But it gave lower accuracy than the default option when giving each observation weight one. Still, it would be better to investigate it further and see if providing customized weights for each target class would help.

### 3. Extracting more information from date features

Columns v173, v175 and v177 contain some date information. In my model I have subtracted epoch date (1 jan 1970) from each date and converted it to seconds, which is in float format. Essentially, each cell with a date after the transformation will show how many seconds has passed after 1 jan 1970 till this cell's initial date. However, it does not capture such things as seasonality (by days, months, years, hours, etc.). In general, it would be good to understand what these dates are about and then to extract some more valuable features like the seasonality.

### 4. Dimensionality reduction

I have removed columns with too many missing values. I have also identified similar features by comparing their absolute correlation with the target variable and have removed the redundant ones. However, different approaches could be used to remove some features.

(1) It would be better to take into account correlations between predictors and not just with the target. (2) In Random Forest model I could have used only the key factors determined by Logistic Regression. (3) Another simple approach is to exclude features that have too little variation. (4) One can try looping through all the features and excluding each at a time and seeing how it affects performance. Features can be excluded also in different groups. Such brute force approach is computationally intensive, but still could be helpful.

Overall, there is no single correct solution, thus it would be helpful to try different approaches and to use cross-validation to find the best strategy.

### 5. Trying different models

I have tried Random Forest Classifier and Logistic Regression models. It would be helpful to test other models as well. Boosting models can be very powerful, particularly XGBoost, which is able to handle missing values. SVM models can do very accurate predictions as well. Finally, Neural Network models can give the best estimates, but it could be much harder to prepare them and different packages should be utilized to use them.

### 6. More targeted feature engineering

I have tried to build a generic model that would be able to automatically handle different data structures and do the necessary transformations. However, in production a more targeted approach can provide better predictions. For instance, column ‘v179’ has values like ‘500 Moderate’, ‘496 Moderate’, ‘147 Low’, ‘291 Low’, and in total it has 83 unique values. It might be a good idea to remove numbers and keep only text, or to separate numbers from text (even though it looks like ‘v179’ is just like ‘v184’ with some randomly added numbers). Such targeted approach can be applied on each column. Of course, it would take a lot of time and such code would much less reusable, but it would be able to increase prediction accuracy.

## Appendix

### Table 1

Correlation between variables and response. We are interested in magnitude, so I have used an absolute value and sorted them from the largest to the smallest.

|  |  |
| --- | --- |
| Variable | Absolute correlation |
| response | 1.000 |
| v192 | 0.194 |
| v191 | 0.194 |
| v29 | 0.173 |
| v204\_wifi | 0.149 |
| v204\_business | 0.105 |
| v204\_cellular | 0.097 |
| v204\_residential | 0.077 |
| v002 | 0.072 |
| v182 | 0.067 |
| v120 | 0.052 |
| v177 | 0.048 |
| v172.1\_n | 0.045 |
| v174 | 0.045 |
| v173 | 0.042 |
| v195\_low | 0.034 |
| v197 | 0.034 |
| v196 | 0.034 |
| v172.1\_y | 0.033 |
| v201\_moderate risk | 0.030 |
| v200 | 0.030 |
| v181 | 0.026 |
| v204\_mobile | 0.025 |
| v178\_certified | 0.024 |
| v180 | 0.023 |
| v193\_yes | 0.022 |
| v178\_validdomain | 0.022 |
| v193\_not sure | 0.022 |
| v183 | 0.021 |
| v184\_lower fraud risk | 0.020 |
| ..… | ….. |
| v184\_data entry review | 0.012 |
| v186\_fraud score 601 to 799 | 0.012 |
| v193\_no | 0.012 |
| v178\_emailinexistent | 0.012 |
| v203\_nan | 0.011 |
| v203\_no | 0.011 |
| v202\_nan | 0.011 |
| v202\_good | 0.011 |
| v201\_nan | 0.011 |
| v186\_nan | 0.011 |
| v172.1\_nan | 0.011 |
| v184\_nan | 0.011 |
| v199\_moderate | 0.011 |
| v195\_nan | 0.011 |
| v194\_nan | 0.011 |
| v194\_yes | 0.011 |
| v178\_nan | 0.011 |
| v193\_nan | 0.011 |
| v199\_nan | 0.011 |
| v172.1\_p | 0.010 |
| v172.1\_u | 0.010 |
| v4 | 0.008 |
| v186\_fraud score 1 to 100 | 0.007 |
| v201\_moderate by proxy reputation and country code | 0.007 |
| v204\_nan | 0.005 |
| v204\_wired | 0.003 |
| v184\_moderate fraud risk | 0.002 |
| v186\_fraud score 301 to 600 | 0.002 |
| v123 | 0.002 |