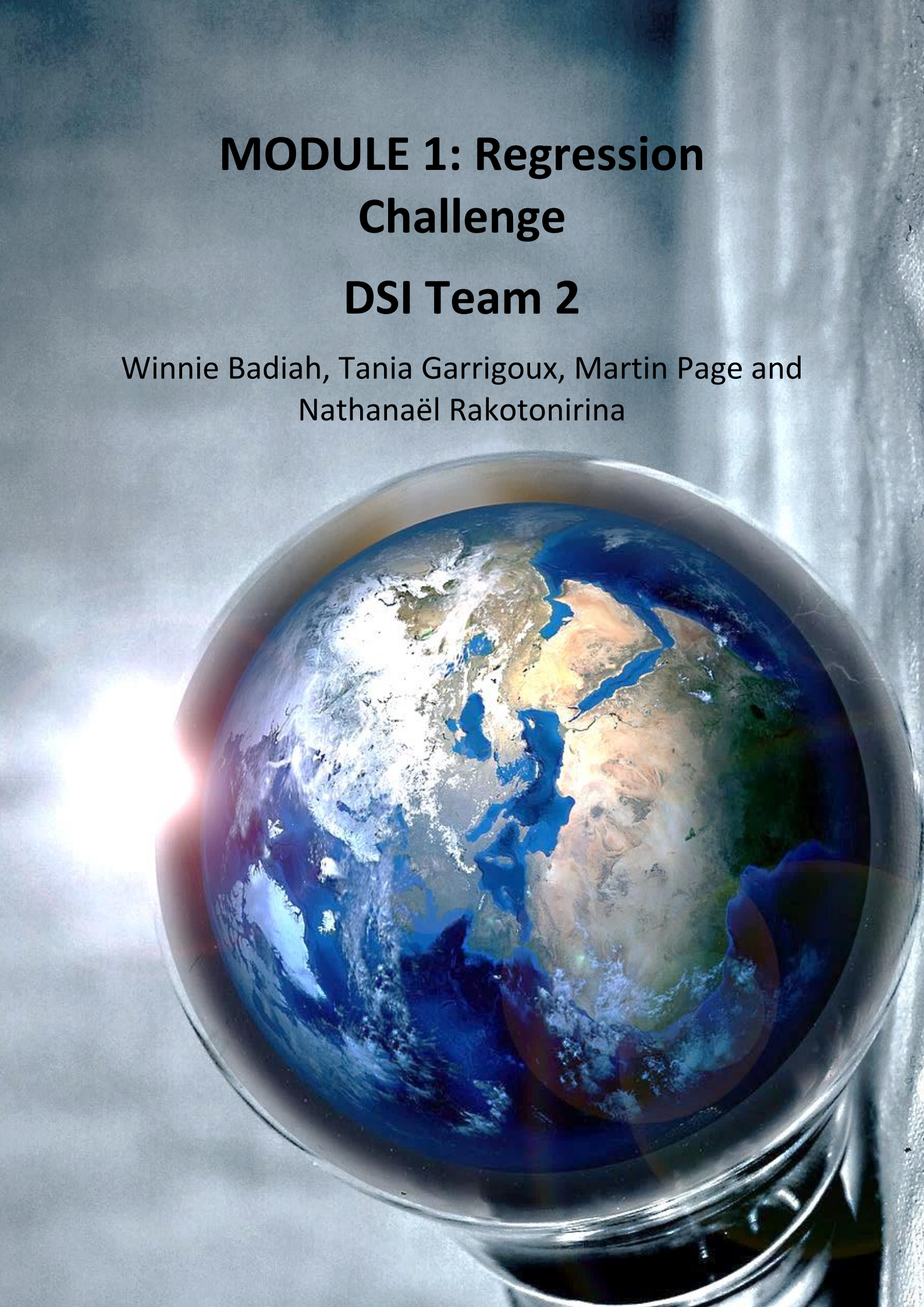
MODULE 1: Regression Challenge

DSI Team 2

Winnie Badiah, Tania Garrigoux, Martin Page and Nathanaël Rakotonirina



Zimnat Insurance Recommendation Challenge

Contents

Executive summary: the what, the why

A. TECHNICAL ASPECT

There is a major gap in insurance protection in Africa, with only 5-10% of Africans owing insurance cover. Insurance markets work best when the population to be insured is diverse and large. The ability of insurance companies to match relevant products to clients is critical for making insurance more effective and making insurance companies more successful.

Today, data are playing an ever more integral role in how businesses understand their consumers. In the context of the insurance industry, this understanding will help insurance companies refine, diversity, and market their product offerings.

Zimnat, a Zimbabwean life assurance and short-term insurance provider, set up a challenge on Zindi, a data science competition platform, to develop a machine learning model that uses customer data to predict the kinds of insurance products to recommend to customer (see https://zindi.africa/competitions/zimnat-insurance-recommendation-challenge). A train set of ~30 000 customers who have purchased two or more insurance products from Zimnat and a test set of ~10 000 customer with all but one of the products they own were provided. The task was to build a model on the train set to predict the most likely missing product in the test set, based on the customer's current profile.

We therefore articulated our **objectives** for this challenge as follows:

1. Train a machine learning model on the current profile of customers who own two or more insurance products
2. Apply the model on a test set to predict the most likely missing insurance product in the manipulated test dataset where one known product has been removed
3. Use cross entropy as the metric for evaluating the model

B. TEAMWORK ASPECT

In addition to the technical aspects of this challenge, our team defined **teamwork objectives** that we believed were necessary to achieve the project's technical goal.

1. Run efficient team meetings based in the Scrum methodology
2. Continuously reflect on our work process to find a systematic way of working
3. Develop protocols and implement systems to achieve an efficient workflow

The setup: getting things ready

Before we could begin exploring the data and building our models, it was necessary to perform data cleaning steps. Python 3.8.5 was used in this project. We performed the following cleaning steps:

a) Wrangled the data into an appropriate format using `pandas`

b) Explored the data to get a better understanding of its structure and relationships using Python and R 4.0.2

c) Created and selected features to use as predictors in our models

d) Created a preprocessor and a pipeline that performed numerical transformations and categorical encoding in a reproducible manner

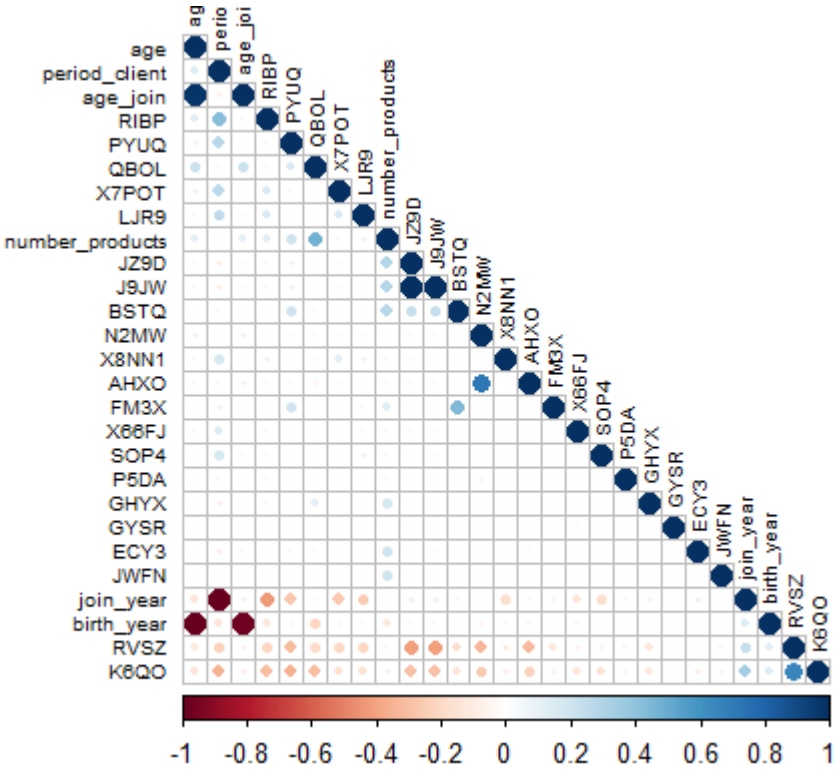
A. DATA FORMATTING

The following major steps were performed to format the data:

- `join_date` was converted to a Unix timestamp, a value in seconds from the epoch, enabling this value to be treated as a numeric variable
- target columns (`y`), the 21 insurance products, were separated from the predictors (`X`) for compatibility with the `sklearn` API
- The predictor variables (`X`) were assigned as either numerical columns or categorical columns so that these could be treated separately in the subsequent preprocessing.

B. EXPLORATORY DATA ANALYSIS

The train and test data were explored using summary statistics and visual representations to confirm our assumptions about the data. The features in the train set and the features in the test set had similar distributions and thus the **train dataset was deemed to be representative of the test dataset**. Further, **missing data points were identified** in `join_date` in both the train and test files, and we decided to use an imputer as part of the pre-processing pipeline to deal with missing data in a consistent and reproducible way (see below). Additionally, the **correlations between the numerical features** were computed and visualised. Some models assume feature independence and we thus needed to remove features that are strongly statistically correlated. This was the case for transformed features that are redundant. The correlations between the different products were considered interesting and a clustering algorithm was implemented to try and capture these relationships as an additional feature for our models. The full exploratory data analysis is available on this project's repository at the link in the appendix.



Correlation matrix of the numerical features in the training dataset.

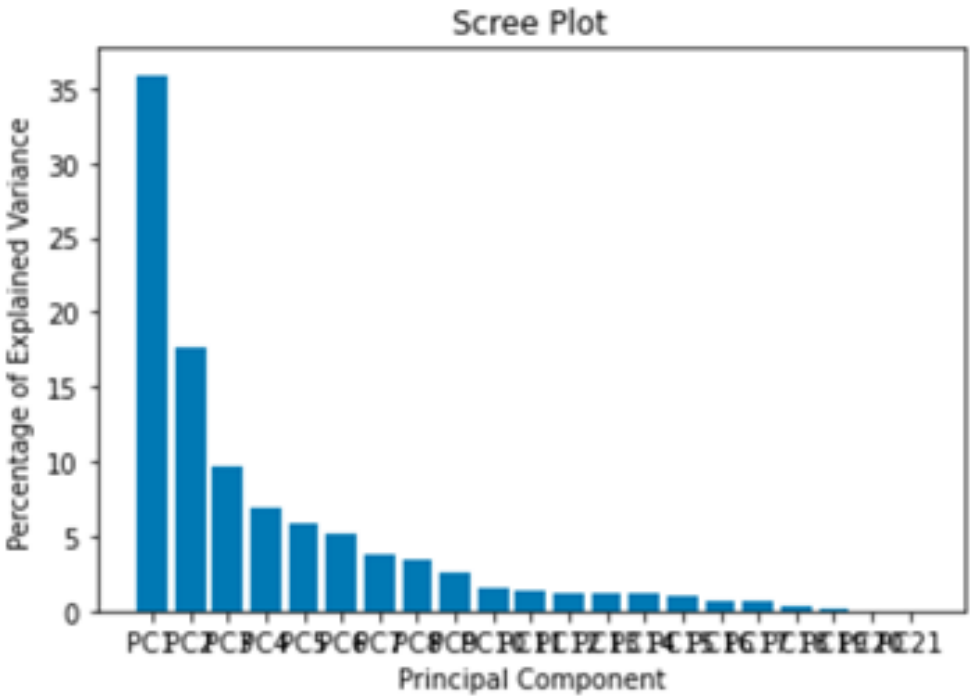
C. FEATURE ENGINEERING

We consider feature engineering as one of the most important parts of this project. Existing features have been modified and new features created to achieve a better representation of the problem. The following features were created:

1. `age` which is a transformation of `birth_year`
2. `period_client` which is a transformation of `join_date`, representing the number of years since the join date
3. `age_join` which computes the age at which a client first joined
4. `number_product` which counts the number of products a client chooses; for the test set we added one to the total count for the test data since one product was removed
5. `customer_cluster` which groups the insurance products into classes using unsupervised learning

In more detail, the clustering was performed using `GaussianMixture` in `sklearn` with the assumption that the customers could be in multiple segments. We did not determine the relationships present in the data as our goal was to use the cluster grouping a customer identifies with as an engineered feature in our classification problem.

The products' data had 21 dimensions, so we used principal component analysis (PCA) to reduce the complexity of the problem by estimating the number of dimensions that maximise variance of the data. We found that with 4 dimensions, 71% of the variance was explained in the train set and the order of the top four variables were: RVSZ, PYUQ, K6QO, QBOL.



We used the Mahalanobis distance metric in the general `GaussianMixture` to determine the optimal number of clusters and resolved to use 6 clusters with an average Silhouette coefficient of 0.70. We then assigned a clustering label to each Zimnet's customers (row in dataset) as a new column in the DaterFrame. We however did not explore the significance of the clusters.

D. PREPROCESSING AND PIPELINE

To keep our code clean and to ensure consistent processing of the data, we implemented a pipeline that performed steps on both the numerical and categorical variables. A pipeline will bundle the preprocessing and modelling steps using the `sklearn` API. With the pipeline established, different models could be defined and fit to a train dataset that has been processed in the same way.

The numerical transformer performed the following processes:

- `SimpleImputer` replaced all missing values with the median of that column
- `StandardScaler` scaled the data to have a mean of 0 and standard deviation of 1

Although there were only two missing values in the train set and one in the test set (in `join_date`), we wanted to set a thorough code which could be applied to new data if needed, so we added a numerical imputer. The scaling step was added since some models do not perform well if the data is not standardised. We then realised most missing values were in the `join_date` column. To fill these missing values, the median was taken not on all examples but only on those with the same birth_year.

The categorical transformer performed the following processes:

- `SimpleImputer` replaced missing values with the most frequent value of the column
- `OneHotEncoder` encoded the categorical features into a number of binary features (dummy variables)

We initially removed the variable `occupation_code` since its 233 unique values and thus high cardinality generally do not feed to `OneHotEncoder`. This however proved not to be a problem in later tests.

The preprocessing steps for the numerical and categorical columns were bundled into a preprocessing transformed using `ColumnTransformer`.

Modelling: getting down and dirty

<u>Problem summary</u>: multi-output (i.e. multi-label) binary classification problem: i.e. 21 products that can take the values 0 or 1. The provided dataset contained six categorical and one numerical predictors. The imposed metric for this project is cross entropy, also known as log loss. This loss function is defined as the negative log-likelihood of a logistic model:

$$\text{Log Loss} = (x,y)D\text{-}y(y') - (1\text{-}y)(1\text{-}y')$$

where y is a label in a labeled example, and y' the predicted value given the set of features x. Both y and y' are between 0 and 1.

A. PROBLEM REPRESENTATION

Finding the right representation of the problem was our most challenging task during this project. We started with a very basic approach before experimenting with less obvious representations.

<u>Multi-output regression</u>

The Zindi challenge is a regression problem. All team members were familiar with single-output regression, but this does not work neatly for this dataset. The products, of which there are 21, were used as targets (`y`) only. The model therefore had to output an array of size 21 with values between 0 and 1. To get the desired output, 21 models (one per class) were independently trained. The regression models were wrapped inside the `MultiOutputClassifier` class to get the desired behaviour.

Multi-output classification

Even though this is a regression problem, the targets are probabilities. Therefore, classification models are more suitable because they can directly output the probabilities of belonging to a certain class. We could not use a simple multi-class classification model since the classes are not mutually exclusive. Indeed, a client can choose more than one insurance product. The appropriate model was a multi-output, often called multi-label, classification model. As in the multi-output regression case, the products were used as targets (`y`) and a model is trained for each product. This was implemented using the `MultiOutputClassifier` wrapper. The multi-output classification model performed better than the regression one.

Chained multi-output classification

Chained multi-output classification is similar to the previous approach except that it exploits the relationship between the targets. This is still a multi-output classification model, which trains the model per product. The first model is trained as usual. However, when training subsequent models, the previous targets are used as additional features. Concerning the order of the targets, the most frequently chosen products are put first. The performance gain using the chained model was not significant. Also, during the test phase, some features are erroneous because one product was removed from the test set, which is manipulated by Zindi.

Single-output classification

The main problem with the multi-output classification approaches is that they do not consider the relationship between the different products. The single-output method trains a single model which can capture the interactions between the products. Data augmentation is performed by setting one of the chosen products to 0 and using its name as the target. This is done for each row and for each chosen product of the training data. This is our best problem representation. This is because it better matches the representation of the test set. In the test set, one product was removed, and the task is to find that product given the usual features and the other chosen products.

B. MODEL SELECTION

We implemented several models to test our different approaches to the problem representation. A description of each model follows.

SOLUTION ONE: LOGISTIC REGRESSION

The principle: Logistic Regression is a supervised learning classification algorithm used to predict the probability of a dichotomous target variable. This model is easy to implement, common enough that all the team members had an idea of how it worked, and it fitted the Zindi Challenge requirements of a probability output on a dichotomous target (provided the multiple targets were taken into consideration).

The challenges: The Zindi Challenge consisted of a multi target classification (ie we needed a prediction for 21 products). `LogisticRegression` handles only single targets, so we had to use a `MultiOutputClassifier` to fit one classifier per target. Unfortunately, the cross-validation functions in `sklearn` cannot handle the computation of log loss when using multi-output models. There is a dimension mismatch error preventing the computation. A solution for this issue was found after we completed our first submission.

The nitty gritty: We tuned the hyperparameters to find the model which best fits the data. Given the asymptotic nature of the expression of the log loss, there is a risk for the model to become completely overfit in high dimensions. Regularisation prevents this problem. We worked with the *`penalty`*, which defines the type of regularisation, and *`C`*, the inverse of regularisation strength. Regularisation mitigates the risk of the model becoming completely overfitted to the data in high dimensions because of the asymptotic nature of the loss function. Since the solver we used (lbfgs) only supports a

penalty **'L2'**, we did not test other options. At first, we tuned the regularisation strength manually using a basic loop without cross validation to assess the best tuning value for our model because at this stage we did not have a solution to overcome the dimension mismatch when using the built in **'neg_log_loss'** scoring in the sklearn API.

The evaluation: The best logistic regression model obtained a Zindi score of 0.076. This was a good score for a first, time-constrained submission, but we were convinced an ensemble method would outperform this model. We decided to look first into XGBoost.

SOLUTION TWO PART A: XGBOOST

The principle: XGBoost is a gradient boosting ensemble method, meaning that it combines the predictions of several models. This method will go through cycles, adding models which reduce the gradient of the loss function, performing what is called a "gradient descent". XGBoost can choose between two types of models to run at each iteration: tree-based models or linear models. We chose to work with tree-based models because they outperform linear models.

The challenges: In addition to wrapping the model in a `MultiOutputClassifier`, we designed a new scorer which directly works on probabilities as predictions without calling the `.predict_proba` method. This solved the problems encountered when trying to use **'neg_log_loss'** scoring in cross-validation functions in sklearn.

The nitty gritty: `XGBClassifier` has many hyperparameters that can be tuned:

- **Learning rate** (*eta*): makes the model more robust by shrinking the weights on each step.
- **Number of trees** to be added (*n_estimators*): an optimum value can be found for a given learning rate.
- **Tree-specific parameters**: maximum depth of a tree (*max_depth*), minimum sum of weights of all observations required in a child (*min_child_weight*), minimum loss reduction in the log function required to make a split (*gamma*), fraction of observations to be randomly sampled for each tree (*subsample*), and fraction of columns to be randomly sampled for each tree (*colsample_bytree*).
- **Regularisation parameters**: L2 regularisation term on weights (*lambda*), L1 regularization term on weights (*alpha*).

After using a random search algorithm (`RandomizedSearchCV`) to fine-tune the hyperparameters, the default values were found to be optimal.

The evaluation: Using the XGBoost model with tree-based models and default values for all parameters, we obtained a Zindi score of 0.066.

This ensemble method indeed improved our basic Logistic Regression model. However, it still predicted each target individually. We believed our model could be improved if we could find a way to take into account the possible correlations between the targets. Our first attempt was using a chained XGboost method.

SOLUTION TWO PART B: CHAINED XGBOOST

The principle: This is similar to the regular `XGBClassifier` model except that it relies on the chained multi-output classification problem representation. Our main motivation was to exploit the relationship between the targets. A model is still trained per product but the target of previous models is used as features for the subsequent models. The `XGBClassifier` is wrapped inside the `ClassifierChain` class to implement this behaviour.

The challenges: It was necessary to find a good ordering of the product. Random ordering did not result in significant performance improvement. The best ordering that we discovered was based on the frequency count of the products.

The nitty gritty: The same hyperparameter tuning as in plain XGBoost was used with this model.

The evaluation: The log loss obtained after submitting to Zindi was 0.064 which is a small improvement from the simple XGBoost model.

The lack of significant improvement with the chained XGBoost did not discourage us in our quest of building better models that could predict all targets at once, and preferably take into account the possible links between them. Next, we decided to double down on the model building task and developed three models in parallel: a Random Forest model, a Bayesian Model and a CatBoost model.

SOLUTION THREE: RANDOM FOREST

The principle: Random forest is an ensemble method that uses a decision tree as the base estimator. Each estimator is trained on a different bootstrap sample having the same size as the training set. Further randomisation in the training of individual trees is introduced by using only a sample (without replacement) of features at each node. For classification, the predictions of the individual trees are aggregated by majority voting, which is collected by the meta-model for final prediction.

The challenges: The `RandomForestClassifier` in `sklearn` is able to take multiple classes as an input so it was not necessary to use the `MultiOutputClassifier` wrapper as for some of our previous models. However, the random forest also suffered an error with the output dimensions when scoring on the **`'neg_log_loss'`** metric that is built into the `sklearn` API. It was necessary to code a new scorer that works directly on probabilities as predictions without calling the `.predict_proba` method.

The nitty gritty: We deemed the following hyperparameters to be the most important for model tuning: *`max_depth`, `max_features`, `min_samples_leaf`, `min_samples_split`, `n_estimators`*. A selection of tuning values was chosen after an internet search of commonly used values. This initial list was distilled to an exhaustive combination of 288 candidate models that were then fit using `GridSearchCV` and a cross-validation of 3, ultimately yielding 864 fits. The model was fit on all the original variables except `birth_year` which was transformed to the variable `age`, and in addition included the engineered variables, `number_product`, `customer_cluster`, and `age_join`.

The evaluation: The best random forest estimator produced from the cross-validated `GridSearchCV` yielded a log loss of 3.64 on the training set and a Zindi score of 0.0936.

As the Random Forest method did not improve our results, we turned our focus on constructing conditional probability functions for each target, given the value of the other targets as well as the predictors.

SOLUTION FOUR: BAYESIAN APPROACH

We researched previous projects that use the Bayesian approach and considered a Linear Regression model with normal priors to predict the customers products (posterior distribution, y).

We came across PyMC3 which is a probabilistic programming framework that allows for model specification directly in Python code. We had a challenge however in determining the optimum parameters for predicting the outcomes, y, which was assumed to be a linear function of 21 predictor variables, X_i where $Y\mu \sim N(\mu,\sigma 2) = \alpha + \sum \beta X_i$.

Since we had a model performing well (XGBoost) and a promising one yet to explore in full (Catboost), we decided to abandon the Bayesian approach due to the inadequate time to explore this approach further.

By this time, we were also considering the issue of the training data not representing the test data (since one of the '1' values in the test data products has been manipulated to a '0'). We decided to explore a method of transforming the training data in the same way as the test data had been

manipulated, to better match the test data used for predictions. We added this function as a possible preprocessing step.

Next, we chose to work with a single-output classification approach, allowing us to consider possible correlations between the targets. Our final method was yet another ensemble method, similar to XGBoost but with a better handling of categorical variables: the CatBoost classifier.

SOLUTION FIVE: CATBOOST

The principle: Catboost is another algorithm for gradient boosting on decision trees. Its main advantages are its support for categorical features and training using a GPU. Unlike the other models that we used during this challenge, Catboost does not require heavy preprocessing of the categorical variables. Furthermore, the model can be trained on GPU, which significantly (a couple of minutes instead of a couple of hours on CPU) increases the training speed.

The challenges: It requires the dataset to have column names to identify the categorical features. Our initial pipeline's output is a `numpy` array which gets rid of the column names. We then had to keep up and convert the data back to DataFrames every time it was transformed.

The nitty gritty: Although the default hyperparameters performed well, we still explored a wide range of hyperparameters. The main hyperparameters we tweaked are *`iterations`, `learning_rate`, `depth`, `l2_leaf_reg`*. The search was done using `GridSearchCV` before doing it manually. However, we could not find better hyperparameters than the default ones.

The evaluation: We used the best problem representation with Catboost which is the single-output classification. This was our best model as evaluated by the log loss score: 0.027 on Zindi.

Conclusions

The best model representation was the single-output classification. As done with the test data, one product of the training data was removed. It was then used as the target. By doing this for each row and for each chosen product, we have not only better matched the test data configuration but also performed data augmentation. The best performing and most practical model was `CatBoostClassifier`. We could rapidly experiment with it owing to the GPU support. Also, the categorical features were handled natively.

Last words from the team: the journey

A. TEAM OUTCOMES

- The duration of our Scrum stand-up meeting improved by 50%, down from 30 minutes to 15 minutes once we understood the meeting structure, which helped make efficient use of everyone's time.
- We developed a communication strategy where we communicated our tasks on the Trello board and sought clarifications via our WhatsApp group.
- At the beginning of the project, we worked separately on our individual section of code, but needed to collaborate and integrate the different sections better. As a first step, we decided to share our files under a common folder on Google Drive. We wrapped our individual code into functions that can be imported in a main.py file, each with its own versioning to make sure the code does not break.
- The above strategy still proved to be cumbersome and we decided to start using Git – in which half of our team members were already proficient. We have not yet extensively tested our workflow using GitHub, but we now all have at least a basic level understanding of the Git workflow.

B. PERSONAL JOURNEYS

Winnie

One of my main objectives in participating in the DSI 2020 Program was to understand how development teams operate from working on projects to deployment. I nominated myself to be the Team's Scrum Master to compel me to quickly learn the same. It's been an interesting journey: We have a super amazing team and I'm honoured to be in Team 2 as my first DSI Group. I am impressed by the diverse backgrounds & expertise - skills in Biology, Physics, Computer Science - in the team all holding or pursuing PhDs! I've learnt a lot from how to think through a challenge, to great communication & documentation, to running short stand up meetings. My key take-away in the past two weeks has been on the importance of effective project management to avoid frustrations within the team. I have come to appreciate the pressure development teams go through with regards to the (sprint) timelines given and how collaboration between the members is key. I am pleased to now have a basic understanding of using Git, Bitbucket and Sourcetree (and decided to not begin on Command Line yet) which I will improve upon in the coming weeks.

Tania

This first module has been quite a positive learning experience for me. As an academic and astrophysicist, I am used to working on big projects which require big amounts of data analysis. However, we rarely work in groups, and never on the same section of the code. Each person will develop his or her own functions for a specific individual purpose, which fits into a common goal, and all of it is predefined. It was at the same time challenging and rewarding to work as a team of four and learn how to be efficient when assigning tasks and sharing the code. I had never used GitHub as a repository to share code with teammates, but I am happy to say that by the end of the module I was able to use its basic features to meet our needs. I was also completely new to the scrum methodology, and it has been wonderful getting to learn how to use it to improve our efficiency as a team. The overall experience was further enhanced by the amazing people in the team and the great communication, which made it so much easier to navigate through the challenges and difficult periods.

Martin

It is actually quite astounding to look back at a period of only two-and-a-half weeks and have so much to reflect on. During the first few days I remember feeling quite lost at times and also frustrated that our process as a team was not more structured and decisive. Personally I have a stronger background in R so I also experienced a Pythonic overload at the beginning. What has come out strongly for me, and which was echoed in the Q&A session with previous DSI participants, is the value of communication and understanding one's teammates. As soon as we started asking each other questions we realised that we were stuck on many of the same points. We also developed an understanding of each other's time availability and skill sets, and we tried to leverage our diversity and different capacities to achieve our objectives. Even though some of the intensity of the teamwork is perhaps less than it would be if were physically present in the same space, I have been able to use the offline time to work through tutorials on machine learning methods that our team decided to use but with which I was not familiar, so I feel that I could use the remote format of the DSI to my advantage by focus on self-teaching when not working on a specific task for the project. I hope that the team work rhythm we established with the Scrum methodology will transfer to some extent to future teams, so that the team process can begin at a more structured level based on previous experience of good practices to follow.

Nathanaël

I learnt much more from these first three weeks than I expected. I come from a more academic background and have very little experience with real-world data science projects. I had the chance to

work with great people from diverse backgrounds. The differences between the team members are more of a benefit than a constraint. They allowed us to have a broader perspective on the problem we had to solve together. It was challenging to adapt the scrum guidelines to this project. We had to figure it out by ourselves and find our own approach. The communication with the other team members was very good and it made things so much easier. I appreciated the exposure to more practical challenges like team management, merging different collaborations with version control. I could clearly see that our team's dynamic has been progressively improving. It took longer to establish best practices like using Git but everyone became more comfortable with it in the end.

Appendix

The repository where the data and code for this project are stored is: https://drive.google.com/drive/folders/1vnBJBZaDNlkse0DS0xfPmJBQEiB0HQ09?usp=sharing