Customer calls prioritisation

End-Point Project Report

Author: Dobromir Marinov
Degree: BSc Digital and Technology Solutions
Registration Number: 1600802
University Supervisor: Dr Daniel Karapetyan
Second assessor: Dr Nicos Angelopoulos
Company Supervisor: Laurence Smith
Date: August 2019

Abstract

MSX International operates a call centre in Italy on behalf of Ford. Each week the centre receives a large list of customers that have to be contacted, to try and get them to book an appointment with an authorised car dealerships in the country. The correct prioritisation of customers is extremely important to the business, due to the fact that the call centre contacts only a part of the customers from the list and makes money only on successful bookings. The project explores the viability of replacing the current legacy prioritisation system, with a more modern machine learning approach, in order to maximise the number of successful bookings.

Table of contents

Chapter 1

Introduction

There are many software systems, still in use today, which are built using methods and technologies that are outdated. Those systems are known as *legacy systems* and represent a major challenge in terms of maintenance and the addition of new features [1]. Furthermore, the behaviour of some legacy systems is hard-coded; that poses a great opportunity for optimisation of the systems [2], using smarter methods, such as machine learning.

MSX International operates a call centre in Italy that makes weekly telephone calls to customers on behalf of car dealerships authorised by Ford. The aim of the calls is to get customers to book service appointments for their vehicles. A new list of customers is received approximately once every week and the number of clients to contact is greater than the total number of calls that the call centre can make per week. Because of that, the customers on the list are scored, and only the customers with high priority score are contacted. The prioritisation scoring is done by a static algorithm that uses hard-coded weights and thresholds for scoring each customer. The values for the weights and thresholds have been selected by human experts in the domain, using statistical analysis on the historical data. Once the customers are scored, a priority queue is created; it is used to assign cases to agents working in the call centre. In addition to the fact that the call centre is unable to contact all customers from the weekly lists, the number of agents in the centre is not fixed due to holidays, sickness leaves, etc. and thus the total number of customers, that can be contacted on any given week, fluctuates.

Because the prioritisation algorithm in the current legacy system is hard-coded and based on heuristics, it will be interesting to see if machine learning can be applied to the problem to improve and optimise the performance. In many cases, machine learning models have been shown to achieve better performance and generalisation that any of the hand made solutions that human experts can create [3, 4]. One other benefit of machine learning models is that they can adapt to new data by retraining, making this approach much more

resilient to change compared to hard-coded systems.

The hypothesis for the project is that there might be a correlation between the likelihood of booking a service appointment and customer or vehicle specific features. For example, owners of more expensive vehicles might be more likely to book an appointment, instead of rescheduling or postponing their service events. Finding such correlation would allow for the creation of a predictive model that can maximise the number of successful bookings per week and will consequently increase the revenue of the entire call centre.

1.1 Aims and objectives

The main aim of the project is to build a machine learning system by using the labelled historical data from previous calls. The new system will then be compared to the legacy system that is currently being used in terms of performance, maintainability and scalability. As a secondary aim, feature engineering based on previously unused data will be attempted to see how newly created features affect the performance and generalisation of the machine learning model.

The starting objective is to understand the data that will be used for creating the machine learning model and to analyse it using a variety of statistical methods. This will allow for the second objective, the data cleaning, to be performed. Once the data is cleaned, the next objective is to engineer new features for the system. With the features created, the next step would be to try different machine learning algorithms to create and train a model. The final objectives of the project will be to validate the model and to optimise its hyperparameters before comparing its performance to that of the legacy system. One important step for comparing the systems is to define a performance metric that is capable of taking into consideration the fluctuation of the number of customers that are contacted each week.

Chapter 2

Background

Before discussing the design and implementation of the newly developed machine learning system, the different phases of a machine learning project should be explained. The discussion in this chapter will start with the data; it will cover the topics of data analysis, data preparation and feature engineering. It will then continue with an overview of model training and an in-depth look into the validation and optimisation of models; more precisely cross validation and hyperparameter optimisation.

2.1 Data analysis and preparation

Data analysis is a multi-step process that is performed in order to gain a better understanding of the data that is being used [5]. The process starts with a preliminary data exploration that allows for the main data errors to be spotted so they can be fixed in the data cleaning step. It also provides an overview of the data, which is critical for the understanding of the problem and for the selection of a suitable modelling algorithm [6].

A major step in preparing the data for modelling is data cleaning. Data cleaning is a process that aims to remove data inconsistencies and errors in order to ensure that a given dataset follows a formal data specification [7]. Some examples of data cleaning include: the conversions of date and time to a specific format, measurement conversion, currency conversion, etc. If a data specification does not exist, it is typically created after the preliminary data exploration and is used to filter or amend data points that do not conform to it. The decision whether data points should be deleted or changed is typically dependant on the specific use case and on the data itself [8]. For example, in many cases it makes sense to convert fields that contain text to either upper-case or lower-case so that they are not treated differently. In some cases, it might also make sense to replace an invalid value

with a value that explicitly shows that it is invalid. For example, if a machine learning model is being built in order to distinguish fraudulent from benign transactions, a field that contains an invalid value might be a key indicator for the model; removing such field or changing its value to match the specification, would most likely result in a decreased performance. Therefore, data cleaning is a dynamic process that is highly dependent on the problem that is being solved.

After the data is cleaned, the main data analysis can be conducted. The aim of this step is to understand the data and to establish the relations between the different fields and data points. Visual data analysis can be used in order to better understand the datasets [9]. A common approach for studying the strength of the relationships between variables is the use of correlation analysis [10]. This type of analysis can uncover previously unknown correlations that can be taken into consideration at the feature engineering phase.

2.2 Feature engineering

Feature engineering is the process of creating features from data. The aim is to create features that will allow for the training of machine learning models and will improve their performance [11]. Feature engineering requires domain knowledge and is an essential step in almost all applied machine learning projects [12]. An example of feature engineering can be the creation of a new feature that expresses the elapsed time between two dates, instead of using the raw dates as features.

A good indicator of the quality of newly engineered features is the their importance. Feature importance is a measurement that quantifies how much a specific feature affects the performance of a machine learning model [13]. The performance of a machine learning system is also dependent on the algorithm that will be used for training; some algorithms can handle correlated features naturally, while for others it might result in significantly under-performing models [14]. Feature engineering is therefore a non-trivial process, that plays a big role in the final performance of the machine learning models.

2.3 Modelling

In order to get predictions on new data points, a machine learning model needs to be created first. The process of creating a model involves the selection of an algorithm which will be used for training. The training is done on historical data in order to improve the model's performance and the data can be either labelled, for supervised learning [15], or unlabelled, for unsupervised learning [16]. It is also worth mentioning that there is a third type of learning, reinforcement learning [17], in which an agent learns from its interactions

with an environment and the feedback that it receives. However, this type of learning is out of the scope of this project and will not be discussed further.

Before the training can commence, the original dataset is split into testing and training datasets. The machine learning algorithm has access to both the features and the labels of the training dataset and uses them to try and learn a function that most optimally represents the relation between them. The training is an iterative process and aims to minimise the difference between the predictions and the ground truth. Gradient descent and variations of it, such as stochastic gradient descent, are the optimisation algorithms that are most widely used [18]. After the training has completed, the performance of the machine learning model is evaluated against the test dataset, which contains data points that have never been seen by the model.

The separation of training and testing datasets and the final evaluation against the testing dataset are important, due to the fact that machine learning models can overfit. Overfitting happens when the machine learning model memorises all the samples in the training dataset and matches them too closely, effectively capturing the noise of the data [19]. The key giveaway of overfitting is that the model achieves high performance on the training set, but a low performance on the testing set. On the contrary, underfitting models are characterised by low performance on both the training and testing datasets which results in models that do not capture the underlying trend of the data points [20].

It is important to note that there is no single metric for measuring the performance of machine learning systems. Some metric that are commonly used are mean square error [21] and area under the receiver operating characteristic curve [22]. However, machine learning problems can oftentimes require a custom metric to be created, in order to obtain a good performing model, tailored to the specific problem.

2.4 Cross-validation

As explained previously, the data used for machine learning models needs to be split into a training set and a testing set and is usually done at random. However, the way in which the data is split can affect the performance of the final model [23]. For example, in a multi-class classification problem, a training set that has a balanced representation from all classes is more likely to result in a better performing model, compared to a training set that has data points for only one of the classes. In order to eliminate the bias that is introduced by using a fixed splits and to measure the real performance of the model, the cross-validation technique is commonly used [24].

The idea behind cross-validation is to train a model on multiple different data splits and to average the final performances of the models. Each split creates a different training and

testing datasets and the averaging of the performance over all splits, aims to account for the data splitting bias explained earlier. Because of that, the cross-validated performance of a model is a much more realistic measure of the actual performance that the model will achieve when given an independent dataset. It is worth noting that there are different types of cross-validation techniques, each of which defines a different way in which the splits are created [25].

2.5 Hyperparameter optimisation

Hyperparameters are a type of parameters that are selected before the training process begins and they dictate the way in which the training process is executed. Machine learning models can have anywhere from zero to hundreds of hyperparameters and different combinations of them affect the final performance of the models [26]. Therefore, it is important to find a hyperparameter combination that results in a high performing model.

The solution to that problem is hyperparameter optimisation; it is a process that aims to find a set of hyperparameters for a machine learning model with which it achieves the highest performance and is represented by the equation:

$$\lambda^* = \underset{\lambda \in \Lambda}{arg\,max}\, f(\lambda)\,, \tag{1}$$

In equation 1, λ^* represents the optimal set of hyperparameters; i.e. the set of hyperparameters λ, sampled from the full hyperparameter search space Λ, that maximises the function $f(\lambda)$. The value of the function $f(\lambda)$ represents the final performance of a machine learning model, trained with a hyperparameter set λ [27].

There are different methods for hyperparameter optimisation, but in this project only grid search and random search will be used, due to their relative simplicity. Both methods evaluate only a subset of the full hyperparameter search space, because the hyperparameter search spaces are typically vast and it is unfeasible to do an exhaustive search [28]. The difference between grid search and random search is in the way in which they choose which hyperparameter set to evaluate next. For grid search, the hyperparameters are given a set of values that they can take; afterwards, all combinations between them are tested [29]. On the other hand, random search samples the values for the hyperparameters from the full search space [30].

Chapter 3

Product implementation

The development process starts with the raw historical dataset that was provided by the company and ends with the creation of the final machine learning model, which is fully optimised and cross-validated. Figure 1 shows an overview of the full project.

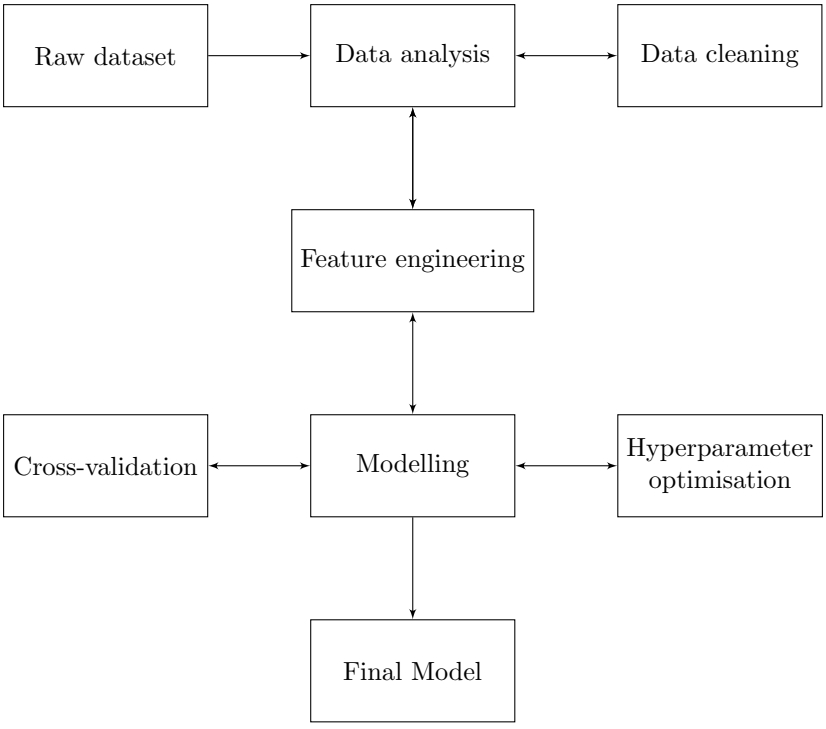

Figure 1: Project life-cycle diagram.

In the beginning, a preliminary analysis of the raw dataset is conducted, followed by a data cleaning procedure. The data analysis and feature engineering are then repeated until a sufficient number of features are created for training. Using the features, a machine learning model is trained, validated using cross-validation, and optimised using hyperparameter optimisation. It is important to note that the cycle of modelling and feature engineering is typically done multiple times, by gradually adding new features and retraining the system, in order to evaluate the impact of the new features on the overall performance.

3.1 Dataset overview

The dataset that has been provided for the project contains information about customer calls made on the behalf of car dealerships, in order to book an appointment with a specific customer. It is a labelled dataset and contains the outcome of the calls. The raw dataset contains some additional fields that are recorded by the system, such as the country of the dealerships, which are not useful in this project as it is only for Italy. All such features have been excluded from the further discussion, due to their irrelevant nature. An overview of the fields in the raw dataset can be found in Table 1.

Field	**Description**
ID_CUSTOMER_CONTACTS	Unique identifier of the customer.
NUMBER_OF CALLS	How many times was the customer called.
EVENT_DATE	Scheduled visit date.
DEALER_CODE	Unique identifier of the dealer.
VEHICLE_MODEL	Model of the vehicle.
VEHICLE_REGISTRATION_DATE	When was the vehicle registered.
MARKETING_EVENT	The type of the visit (e.g. MOT).
BATCH_DATE	Date of the record.
DEALER_TOWN	The town of the dealer.
DEALER_POSTCODE	The postcode of the dealer.
CUSTOMER_TOWN	The town of the customer.
CUSTOMER_POSTCODE	The postcode of the customer.
OUTCOME	Outcome of the call.

Table 1: List of all fields from the original dataset and their descriptions that have been provided for the purposes of the project. Some fields have been omitted due to their unsuitability for the project.

The dataset contains data points for 2018 and 2019 and the total number of records is 153,419. In addition to the labelled dataset, which contains only customers that have been

contacted, a dataset has been provided that shows the total number of available customers to contact per week.

3.2 Preliminary data analysis and cleaning

The initial data analysis was conducted in order to discover all of the inconsistencies in the original dataset and to fix them. The first major data cleaning work was done on the dates and on their format. The original dataset contained dates in different formats, with some of them also containing time. Additionally, the data extract provided was supposed to contain historical data for 2018 and 2019, but some of the records were from previous years. In order to clean the dataset, data rows were filtered to remove all data points from earlier years. In addition, data columns were formatted following the ISO 8601 [31] standard as 'YYYY-MM-DD'. In addition to invalid dates, some of the records were missing the vehicle registration date. For such records, the missing vehicle date has been filled with an average registration date calculated from the full dataset.

The outcomes of the calls have also been normalised. They have been mapped to either 'successful' or 'unsuccessful', from the original values that they had. The original values had multiple categories, due to the fact that they included a reason for the customer rejection. The original categories have been narrowed down only to two, due to the fact that there already exists a system that deals with customer rejections due to invalid mobile numbers or similar reasons. In the context of this project, the particular reason for rejection are not of interest and the data can safely be used for binary classification.

The main software tools that were used for data cleaning and exploration in this project were *Pandas* [32] and *Matplotlib* [33]; they are third party libraries for the programming language Python. *Matplotlib* is an open source, visualisation library that has been used in order to create plots and graphs for visual analysis in the project. It has been used to produce histograms, bar charts and scatter plots, in order to compare and contrast different values from the dataset. It is extremely customisable and allows for consistent styles to be defined and used for the creation of wide variety of plots. *Pandas* is also an open source library. It is used for handling large datasets using optimised data structures. The library is capable of working directly with CSV files, both reading and writing, and can be easily used for splitting datasets into training and testing subsets. Additionally, it can apply functions iteratively to multiple elements in both rows and columns of the original dataset.

3.3 Feature engineering

The first step in creating features for the machine learning algorithms, was to use the data from all of the date columns. They were transformed to columns that contain the days since the original dates. It was also experimented with encoding the elapsed time in months and years.

The region and macro region of Italy, in which the customers were living, were also added as features. A data file was created to map the different postcodes in Italy to the different regions and macro regions in the country, in order to create those features.

A major challenge was to normalise the vehicle models. The original data has been taken from free-text fields and contained different representations for vehicles of the same model. It also contained abbreviations and typographical errors. In order to make them usable for machine learning, they were matched to all known vehicle models of the manufacturer, using fuzzy matching techniques [34]. A threshold of 75% was set and all vehicle models that were at least 75% similar to one of the known vehicle models, were mapped to it. If the vehicle model was not similar to any of the known vehicle models, it was mapped to the 'other' category.

The previously mentioned features, the regions and the vehicle model, plus additional features, such as the marketing event, needed to be transformed into numerical values. This was important, because most of the machine learning algorithms are unable to handle text data and require all such fields to be numerically encoded. Initial experiments were done with one-hot encoding [35], but it was later replaced with encoding based on James-Stein estimator [36], in order to obtain better performance. For a given categorical feature, James-Stein encoding calculates the mean of all target values for features in the same category and the mean of the target values of the whole dataset. It then sums the two mean values, in order to produce the numerical representation of the categorical feature. For example, if we want to encode the marketing event for MOT with a James-Stein encoder, we would have to calculate the mean of the target values (the outcomes of the calls) for all data points which have MOT as their marketing event. We would then need to calculate the mean value of the target values for all data points and add the two numbers together.

The final step of the feature engineering process was to use the postcodes of both the customers and the dealers and to calculate the driving distances in meters between them. In order to do that, the postcodes were first used to find the coordinates of the dealers and customers using an open source geocoding tool; the coordinates were then passed to a routing engine, which calculated the driving distance, using data from an open source map.

The software tools that are used for feature engineering in the project include a Python

library called *FuzzyWuzzy* [37] and two open source tools called *OSRM* [38] and *Nominatim* [39]. *FuzzyWuzzy* is used for measuring the difference between text strings, using Levenshtein distance [40]. It calculates a similarity index between 0 and 100 that represents how similar the text strings are. A threshold can then be selected, for example 70%, and all strings that have a higher similarity than the threshold can be grouped together. *Nominatim* is an open source tool for geocoding. It is capable of providing the exact latitude and longitude of a location, based only on its address or postcode. *OSRM* is an open source engine for routing; it can calculate driving distances and find the fastest routes between map coordinates. It also contains additional services for solving more complex routing problems, such as the travelling salesman problem [41], however, they have not been used as part of this project because are out of its scope. Both *Nominatim* and *OSRM* are based on the open source map *OpenStreetMap* [42] and have been used in the project as services provided from docker containers running locally. In order to setup the services locally, an *OpenStreetMap* data extract, for the country of Italy, was obtained from *Geofabrik* [43]. It was then used to initialise a database that is used by the services when an HTTP request is made to them.

3.4 Modelling

The first machine learning algorithm that was considered was logistic regression [44]. However, it was very simplistic and prone to underfitting when used for more complex problems. It was also unable to handle imbalanced datasets.

The second step was to try ensemble methods, such as random forest [45] and gradient boosting [46]. Both of the algorithms build multiple decision trees, however, gradient boosting builds them sequentially on the residuals from the previous trees, whereas random forest builds different trees on different subsets of the original dataset. Because random forest uses the original dataset for all of its decision trees, it also requires the dataset to be balanced before training, in order for the final model to be unbiased. In contrast, gradient boosting can handle slightly imbalanced datasets naturally and can create unbiased models without the explicit need for balancing the dataset.

The key Python libraries that have been used for training machine learning models were *Scikit-learn* [47] and *XGBoost* [48]. *Scikit-learn* is an open source library that provides a large variety of implementations of machine learning algorithms. It has been used extensively in the training and evaluation of different models, due to its ease of use and comprehensive documentation. The library *XGBoost* is a custom implementation of a gradient boosting algorithm. It is used in this project due to the fact that its core is written in C++, making it faster and more efficient.

3.5 Validation and optimisation

In order to find the real performance of the trained machine learning system, cross-validation has been used to remove the data splitting bias. The cross-validation was performed by splitting the data in 20 subsets, using a stratified K-Fold algorithm [49]. This algorithm ensures that the ratio between the two classes in each of the subsets is the same as the ratio in the original dataset. Each of the subsets contained 7,571 records, which is approximately the average size of a weekly batch of customers that is received by the real system. The final cross-validation score is then calculated as the average of the scores of the 20 models, evaluated on one of the 20 different subsets and each of them trained on the remaining 19 subsets. The cross-validation in this project has been implemented using a sub-module from the *Scikit-learn* library that provides the logic behind the creation of the data splits.

The hyperparameter optimisation is done using either grid search or random search. It is implemented using the parameter grid class from the *Scikit-learn* library. A method has been implemented in order to allow the hyperparameter optimisation to be done using cross-validation with the custom model performance metric. A grid with reasonable values for the hyperparamters have been created, based on previous experience with the dataset and the *XGBoost* machine learning library.

Both the cross-validation and the hyperparameter optimisation are suitable for parallelisation, because models trained with different hyperparameters or trained on different datasets are independent. Because of that, both systems have been implemented to run multiple training procedures in parallel, in order to speed up the process. The number of models trained simultaneously is equal to the number of available cores on the machine on which they are trained and is selected dynamically. The Python library *Joblib* [50], has been used in order to implement the parallel execution.

3.6 Functional testing and integration

The project contains 84 unit tests that cover 99% of the code that has been written, which can be found under the *test* directory in the *GitLab* repository. They are executed by a continuous integration system on every commit to the master branch of the repository and ensure that the behaviour of the code is correct. Another major purpose of the unit tests is to ensure that new changes do not introduce regression in the system.

The continues integration system uses a *YAML* file to specify the steps to be executed after a commit. It always starts in a new environment and pulls the latest version of the project from the repository. It is then responsible for downloading and installing all dependencies

of the project, before executing the suite of unit tests. It then reports the result of the tests and if there is a failure, it also sends an email notification.

The main library used for writing the unit tests is *PyUnit* [51], with the addition of the library *mock*, which comes bundled with *PyUnit*. It is used extensively for simulating some of the more expensive calls to other services, such as the geocoding and routing services. The continuous integration has been implemented on the *GitLab* platform with their custom pipelines. It is possible to setup different procedures for continuous integration on different branches, however this is out of the scope of the current project.

Chapter 4

System evaluation

The evaluation of the performance of the new system will be split into two parts. The first part will include a comparison between the importance of the different features that have been engineered. It will also discuss some of the underlying reasons for the under-performance of some of the features. The second part, which will be the main part of the discussion, will explore the overall performance of the final system and will compare it to the performance of the legacy system.

In order to evaluate the performance of the systems, a custom metric needed to be defined. The aim of the systems is to maximise the number of successful calls that are made each week. However, the lists of customers to contact, which are received by the call centre each week, can be different in size. Furthermore, the number of customers that the call centre can contact each week is also fluctuating. Because of that, the metric needed to take into consideration not only the number of successful predicted calls, that resulted in bookings, but also the probability of contacting different percentages of customers from the full list. Figure 2 shows the probability of contacting different percentages of the total customers each week.

The analysis shows that in order to accurately compare the performance of the systems, the fluctuation of contacted customers needs to be taken into account. For a given percentage of all customers, the number of true positives (customers that were predicted to make a booking and resulted in a booking) can be counted, and the probability of contacting that percentage of customers is also known. The proposed custom metric can then be defined as the weighted sum of true positives, over different percentages of the data. In practice, that is implemented as the sum of products, between the number of true positives that the system predicted in n percent of the customers and the probability of contacting n percent of the customers.

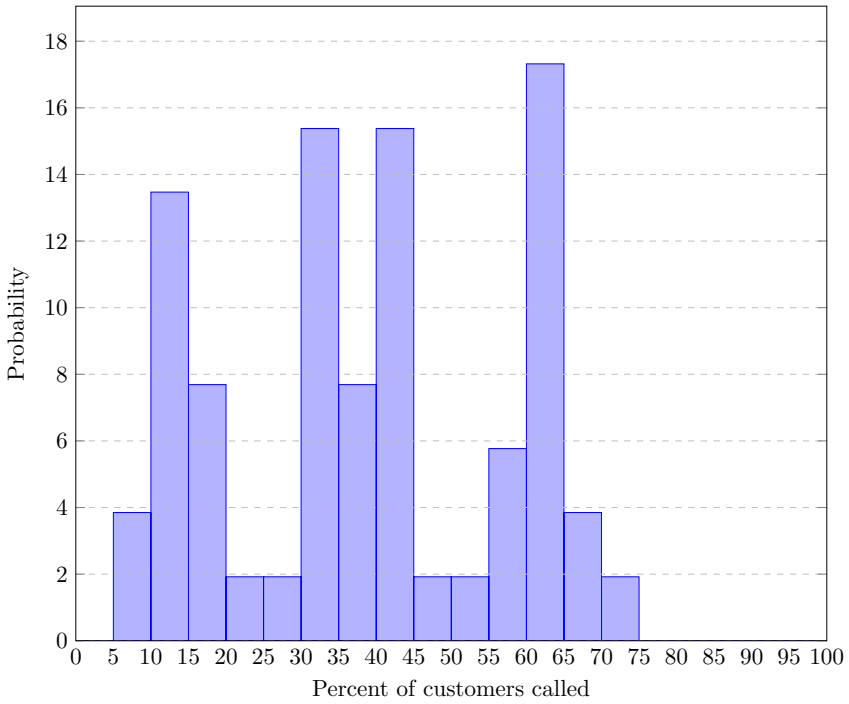

Figure 2: Summary of the probabilities of contacting different percentages of the total customers. The summary has been created based on historical data from the systems. The high probability of contacting only 10 to 20 percent of the customers corresponds with the summer and Christmas holidays; the high probability of contacting between 60 and 65 percent of the customers corresponds with the periods in which the call centre operates at full capacity.

4.1 Evaluation

The first part of the evaluation process was to take a look into the features that were engineered. Not all features are equally important to the model, because some of them contain more information about whether or not a call will result in a booking. Figure 3 shows the importance of the different features that were being used. The distance is the least important feature and that is due to the fact that it was not available for a large portion of the data, due to invalid or missing postal codes. It is interesting to see that the days since the last visit and the marketing event, are key predictors of the success of a given call. In contrast, the legacy system uses the days since last visit as a key predictor, but does not give a high importance to the marketing event.

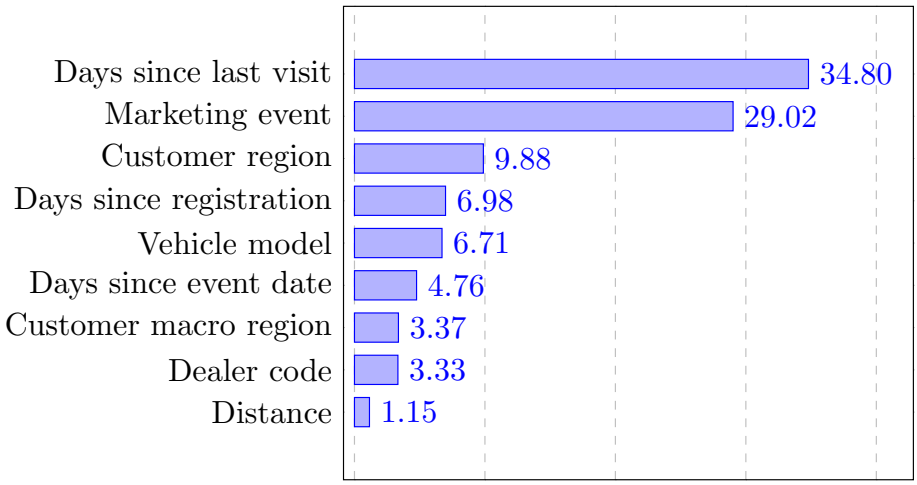

Figure 3: Features used in the training of the machine learning models, ordered in decreasing order of their importance. The importance has been calculated by the XGBoost library while training the models.

For the main evaluation of the new system, 100 runs using different data splits have been performed. The performance of the system has been evaluated using the custom metric. The distribution of the performance for all 100 models can be seen in figure 4.

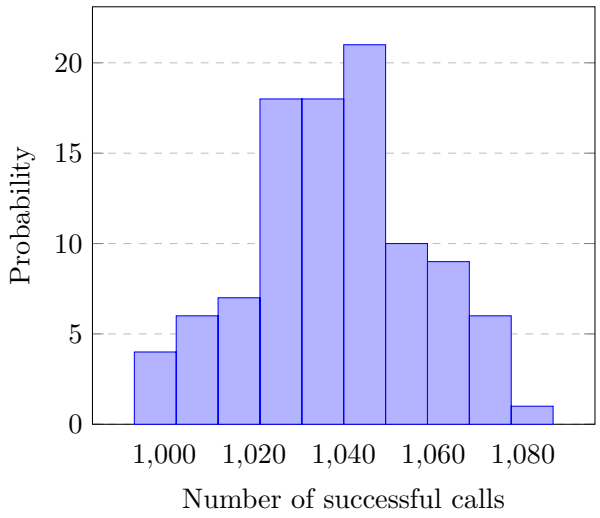

Figure 4: Distribution of the performance of the new system created by evaluating 100 models, trained on different parts of the full dataset. On average, the model identifies 1,038 customers that will make a booking.

Despite the changes in the data, the machine learning model can almost always correctly identify 1,000 of the customers that will be willing to make a booking. For comparison, the old system can identify only 964, of the total 1,664 client records in a data split, that will result in a booking.

In the final experiment, the performance of the legacy system and the new system were evaluated on the same data splits. Their performance was compared over different percentages of customers contacted. Figure 5 plots the performance of both systems.

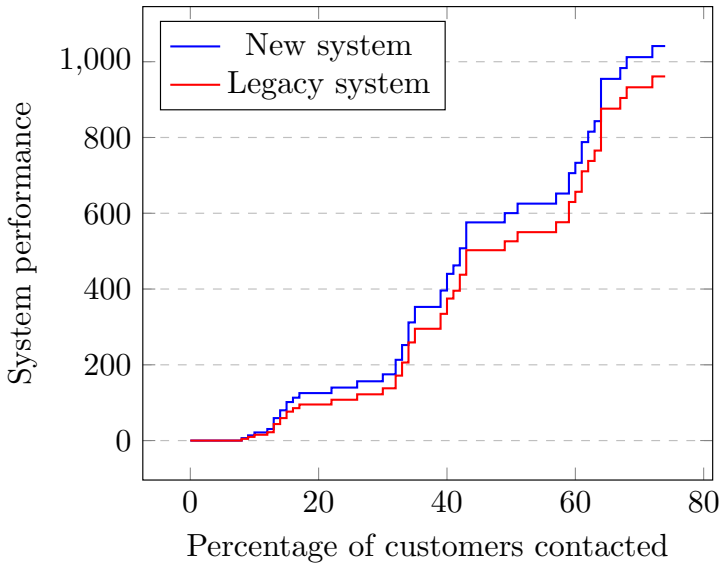

Figure 5: Comparison between the legacy system and the new system over different percentages of customers contacted. The diagram shows that the new system is consistently better than the legacy system, making it more efficient for the business.

The new machine learning system is clearly outperforming the legacy system over the full comparison range. On average, it achieves an increase of 49.22 more bookings. It is important to note that the most of the performance increase is due to the use of machine learning and more specifically, gradient boosting. The newly engineered features did not contribute significantly to the performance boost, with two of the top three features, based on their feature importance, already being present in the legacy system. However, the increase in performance shows that using machine learning is better suited for the problem and that employing it will result in benefits for the company.

Chapter 5

Project planning

The original scope of the project was to develop a new machine learning system for prioritising customer calls, to engineer new features for it using previously unused data and to compare its performance to the legacy system. It was a preliminary research, with the aim to provide a proof of concept system, based on machine learning techniques, in order to evaluate their suitability for solving the business problem.

It was estimated that the project can be completed in two and a half months of full-time work. In addition, all of the data needed for the project was already available to the company, making it independent from third-parties. Because the project was part of the apprenticeship degree program and had to be completed by a single developer, there were no additional cost in terms of allocating more developers to the project.

5.1 Commercial value

Starting with the performance increase achieved by the newly developed machine learning system, an average of 49 more bookings will be made per week. The company is paid approximately 25€ per successful call, resulting in approximately 1,225€ increase in revenue per week. Furthermore, the increase in scalability and maintainability introduced by the new system, will reduce the development and operating costs, resulting in further savings for the company.

5.2 Project management tools

The project management tools used for this project were *Jira* and *GitLab*. *Jira* provides a Kanban board that allows for the creation of tickets that describe the different project tasks and for the organisation of those tasks into different columns, based on their status. In the context of this project, three different types of tasks have been used for labelling the tickets: epic story, task and sub-task. An epic story represents a larger piece of work, with a scope of approximately 3 weeks. The epic stories were then split into tasks, with each task representing approximately a week worth of work. The tasks were further split into sub-tasks that can typically be completed in one to two days. The project contained 3 epic stories that covered the project initiation, the core system development and improvement and the project finalisation.

GitLab provides private *Git* repositories used for version control. A single branch has been used for the development of the project and only code has been stored in the repository. All data files have been excluded from version control, due to their large size. Code documentation, containing instructions for running the project and information about the third-party dependencies, have been provided in the form of a *README* file in the repository.

Additional information about the *Jira* board and the *GitLab* repository, including links to them, can be found in appendix A.

5.3 Risk management

One of the most substantial risks for projects with hard deadlines, is project slippage. The main strategy for mitigating that type of risk was to have regular weekly meetings with the academic supervisor for the project. In each meeting, the work done up to that point was discussed and appropriate plan was set for the future. That included the prioritisation of critical tasks and the resolution of unexpected development problems.

A substantial risk was introduced, due to a change in project priority by MSX International. In the second half of the project, the company decided that they can no longer provide the initially agreed time investment and the scope of the project had to be reduced to accommodate the change. The change that the company made, effectively reduced the development time in half for the second part of the project. This was mitigated by negotiating a reduction of the project scope. The agreement that was accepted by both the company and the university, was to leave out some of the more in-depth feature engineering, as part of the future work for the project. The changed allowed for a timely completion of the other parts of the system.

Chapter 6

Conclusion

In conclusion, the project has demonstrated the benefits of using a machine learning system over the hand-crafted, legacy system. The legacy system has worse performance and is also difficult to change due to its inability to be retrained and its hard-coded nature. In comparison, the new machine learning system has shown that it can improve the performance of the call centre and as a result, their revenue; it can also be retrained on new data in less than a day. Additional benefits are the increased maintainability and scalability that the new system can offer, making it more reliable and future-proof. For example, the data cleaning and training procedures for the models, support the addition of new features. Furthermore, because the custom performance metric has been implemented as a separate component, it is possible to replace it easily, based on the future needs of the company. The extensive testing will also ensure that no regression is introduced when making new changes and will support the future development of the project.

6.1 Future work

As part of the future work, it will be interesting to see if additional features can be added to the machine learning model in order to improve its performance. It will also be useful to apply more cleaning procedures and to obtain more data from the data warehouse, in order to try and improve the calculations of the distances between the customers and the dealers. As part of the deployment process, it will be important to define clear procedures for deploying the model to the production system and for the retraining of the model with new data.

Chapter 7

Bibliography

[1] K. Bennett, "Legacy systems: coping with success," *IEEE Software*, vol. 12, pp. 19–23, Jan. 1995.

[2] A. van Deursen, P. Klint, and C. Verhoef, "Research Issues in the Renovation of Legacy Systems," in *Fundamental Approaches to Software Engineering* (G. Goos, J. Hartmanis, J. van Leeuwen, and J.-P. Finance, eds.), vol. 1577, pp. 1–21, Berlin, Heidelberg: Springer Berlin Heidelberg, 1999.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv:1312.5602 [cs]*, Dec. 2013. arXiv: 1312.5602.

[4] T. W. Nattkemper, T. Twellmann, H. Ritter, and W. Schubert, "Human vs. machine: evaluation of fluorescence micrographs," *Computers in Biology and Medicine*, vol. 33, pp. 31–43, Jan. 2003.

[5] J. Ritchie, L. Spencer, and L. Spencer, "Qualitative data analysis for applied policy research," Sept. 2002.

[6] C. H. Yu, "Exploratory Data Analysis," *Exploratory Data Analysis*, p. 31, 2017.

[7] E. Rahm and H. Hai Do, "Data Cleaning: Problems and Current Approaches," *IEEE Data Eng. Bull.*, vol. 23, pp. 3–13, Jan. 2000.

[8] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller, "Continuous data cleaning," in *2014 IEEE 30th International Conference on Data Engineering*, pp. 244–255, Mar. 2014.

[9] D. A. Keim, "Information visualization and visual data mining," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, pp. 1–8, Jan. 2002.

[10] C. Vielhauer and R. Steinmetz, "Handwriting: Feature Correlation Analysis for Biometric Hashes," *EURASIP Journal on Advances in Signal Processing*, vol. 2004, p. 389304, Apr. 2004.

[11] J. Heaton, "An empirical analysis of feature engineering for predictive modeling," in *SoutheastCon 2016*, pp. 1–6, Mar. 2016.

[12] V. N. Garla and C. Brandt, "Ontology-guided feature engineering for clinical text classification," *Journal of Biomedical Informatics*, vol. 45, pp. 992–998, Oct. 2012.

[13] M. Tsuchiya and H. Fujiyoshi, "Evaluating Feature Importance for Object Classification in Visual Surveillance," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 2, pp. 978–981, Aug. 2006.

[14] L. Yu and H. Liu, "Efficiently Handling Feature Redundancy in High-dimensional Data," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, (New York, NY, USA), pp. 685–690, ACM, 2003. event-place: Washington, D.C.

[15] O. M. Mozos, C. Stachniss, and W. Burgard, "Supervised Learning of Places from Range Data using AdaBoost," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 1730–1735, Apr. 2005.

[16] H. Barlow, "Unsupervised Learning," *Neural Computation*, vol. 1, pp. 295–311, Sept. 1989.

[17] M. van Otterlo and M. Wiering, "Reinforcement Learning and Markov Decision Processes," in *Reinforcement Learning: State-of-the-Art* (M. Wiering and M. van Otterlo, eds.), Adaptation, Learning, and Optimization, pp. 3–42, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[18] L. Bottou, "Large-Scale Machine Learning with Stochastic Gradient Descent," in *Proceedings of COMPSTAT'2010* (Y. Lechevallier and G. Saporta, eds.), pp. 177–186, Physica-Verlag HD, 2010.

[19] T. Dietterich, "Overfitting and Undercomputing in Machine Learning," *Computing Surveys*, vol. 27, pp. 326–327, 1995.

[20] H. Jabbar and R. Z. Khan, "Methods to Avoid Over-Fitting and Under-Fitting in Supervised Machine Learning (Comparative Study)," pp. 163–172, Jan. 2014.

[21] J. V. Hansen, "Combining predictors: comparison of five meta machine learning methods," *Information Sciences*, vol. 119, pp. 91–105, Oct. 1999.

[22] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, pp. 1145–1159, July 1997.

[23] Z. Reitermanová, "Data Splitting," 2010.

[24] J. Shao, "Linear Model Selection by Cross-validation," *Journal of the American Statistical Association*, vol. 88, pp. 486–494, June 1993.

[25] P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-Validation," in *Encyclopedia of Database Systems* (L. LIU and M. T. ÖZSU, eds.), pp. 532–538, Boston, MA: Springer US, 2009.

[26] P. Probst, B. Bischl, and A.-L. Boulesteix, *Tunability: Importance of Hyperparameters of Machine Learning Algorithms*. Feb. 2018.

[27] D. Marinov and D. Karapetyan, "Hyperparameter Optimisation with Early Termination of Poor Performers," *CEEC 2019*, Sept. 2019. arXiv: 1907.08651.

[28] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for Hyper-Parameter Optimization," in *Advances in Neural Information Processing Systems 24* (J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, eds.), pp. 2546–2554, Curran Associates, Inc., 2011.

[29] E. Ndiaye, T. Le, O. Fercoq, J. Salmon, and I. Takeuchi, "Safe Grid Search with Optimal Complexity," *arXiv:1810.05471 [cs, math, stat]*, Oct. 2018. arXiv: 1810.05471.

[30] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[31] I. standardization organization, *ISO 8601: 2004 (E): Data Elements and Interchange Formats, Information Interchange, Representation of Dates and Times*. ISO, 2004.

[32] "pandas: powerful Python data analysis toolkit — pandas 0.24.2 documentation." Available at http://pandas.pydata.org/pandas-docs/stable/ (accessed on 26/05/2019).

[33] "Matplotlib - plotting 3.1.0 documentation." Available at https://matplotlib.org/ (accessed on 2019-08-22).

[34] M. Cayrol, H. Farreny, and H. Prade, "Fuzzy Pattern Matching," *Kybernetes*, Feb. 1982.

[35] S. Garavaglia, "A Smart Guide to Dummy Variables: Four Applications and a Macro," p. 10, 1998.

[36] R. Tibshirani, "Stein's Unbiased Risk Estimate," p. 12, 2015.

[37] "Fuzzywuzzy - fuzzy string matching library built in python." Available at https://github.com/seatgeek/fuzzywuzzy (accessed on 2019-08-22).

[38] "Osrm api documentation." Available at http://project-osrm.org/docs/v5.22.0/api (accessed on 2019-08-22).

[39] "Nominatim - a tool to search osm data by name and address." Available at https://nominatim.org/release-docs/develop/ (accessed on 2019-08-22).

[40] L. Yujian and L. Bo, "A Normalized Levenshtein Distance Metric," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, pp. 1091–1095, June 2007.

[41] M. Held and R. M. Karp, "The Traveling-Salesman Problem and Minimum Spanning Trees," *Operations Research*, vol. 18, pp. 1138–1162, Dec. 1970.

[42] "Openstreetmap - open source map." Available at https://nominatim.org/release-docs/develop/ (accessed on 2019-08-22).

[43] "Geofabrik." Available at https://www.geofabrik.de/ (accessed on 2019-08-22).

[44] R. E. Wright, "Logistic regression," in *Reading and understanding multivariate statistics*, pp. 217–244, Washington, DC, US: American Psychological Association, 1995.

[45] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, pp. 5–32, Oct. 2001.

[46] R. E. Schapire, "The Boosting Approach to Machine Learning: An Overview," in *Nonlinear Estimation and Classification* (P. Bickel, P. Diggle, S. Fienberg, K. Krickeberg, I. Olkin, N. Wermuth, S. Zeger, D. D. Denison, M. H. Hansen, C. C. Holmes, B. Mallick, and B. Yu, eds.), vol. 171, pp. 149–171, New York, NY: Springer New York, 2003.

[47] "Documentation scikit-learn: machine learning in Python — scikit-learn 0.21.2 documentation." Available at https://scikit-learn.org/stable/documentation.html (accessed on 26/05/2019).

[48] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," 2016.

[49] N. A. Diamantidis, D. Karlis, and E. A. Giakoumakis, "Unsupervised stratification of cross-validation for accuracy estimation," *Artificial Intelligence*, vol. 116, pp. 1–16, Jan. 2000.

[50] "Joblib: running python functions as pipeline jobs — joblib 0.13.2 documentation." Available at https://joblib.readthedocs.io/en/latest/ (accessed on 2019-08-22).

[51] "Pyunit: Unit testing framework built in python - v3.3 documnetation." Available at https://docs.python.org/3/library/unittest.html (accessed on 2019-08-22).

Appendix A

Portfolio

A.1 Jira

The *Jira* board that has been used for the project management and planning, contains all of the tasks that were done in the project and can create reports summarising the full project life-cycle. The board can be found here: Jira board.

A.2 GitLab

The *GitLab* repository contains all of the code for the project and can show the full history of commits, throughout the project life-cycle. It also provides the platform for continuous integration, that executes all functional tests, on every commit to the main branch. The repository can be found here: GitLab repository.