TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

"FlixCharter: Analyzing the value of bus charter requests"

Authors	Bilous, Oksana
	Rathore, Aurangzeb A.
	Regalado Fernández, Luis Andrés
	Sloof, Marit
Mentor(s)	Dr. Berit Johannes
	Luca Petrone
Co-Mentor	Philippe Sünnen
Project Lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

Feb 2020

Abstract

Flixbus is a German transport industry based company that offers intercity bus service in Europe. FlixCharter is a startup within Flixbus that offers bus rental for group trips across Europe. The company uses a bundler system to connect trips in a way to minimize empty kilometers travelled. As such, factors pertaining to strong business decisions can have a substantial impact on the company's day to day margin. Elements such as cancellation can limit the production of accurate forecasts, a critical tool in terms of revenue management performance. To help circumvent such problems, we used analysis techniques on past datasets, provided by the company, for meaningful information and trends that would aid in future decisions. Additionally, we anticipate that with large enough datasets it is possible to build models that can predict decisive factors with a large accuracy.

Contents

1 Introduction

With the amount of data produced and available nowadays, many challenges and benefits lie within analyzing this data for companies. Data can provide valuable insights in customer buying patterns and ways of lowering costs by which it can help to make certain business decisions (Wang, Gunasekaran, Ngai, and Papadopoulos, 2016). These opportunities are also relevant FlixCharter, a bus rental company (from now on also referred to as the Company). This project is dedicated to evaluate incoming bus charter requests based on their available data, and hence optimize their operations management. In order to arrive at the desired outcomes, we first provide a complete understanding of the Company.

The customer experience A customer can create desired trips on the Company's website by filling in the departure time and location, arrival time and location, and the number of passengers, see Figure 1. Based on this information, an instant price is calculated and a 7-day price guaranteed offer is generated for the customer. If the customer reserves, he can change or cancel the trip for free up to fourteen days before departure. If the customer cancels within the 13 days prior to the scheduled departure, only a certain amount of the price is reimbursed. In general, the booking process for customers can be summarized as in Figure 2.

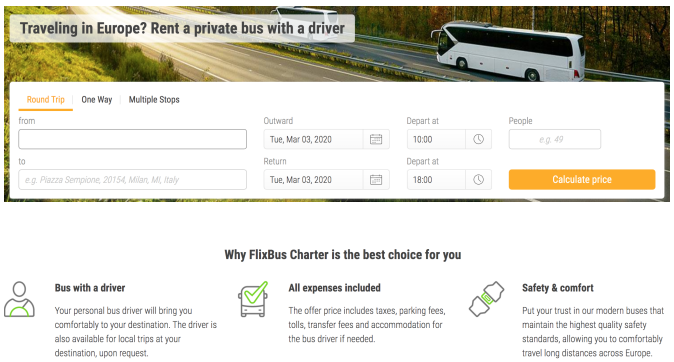

Figure 1: The website of FlixCharter where a customer provides all first relevant information.

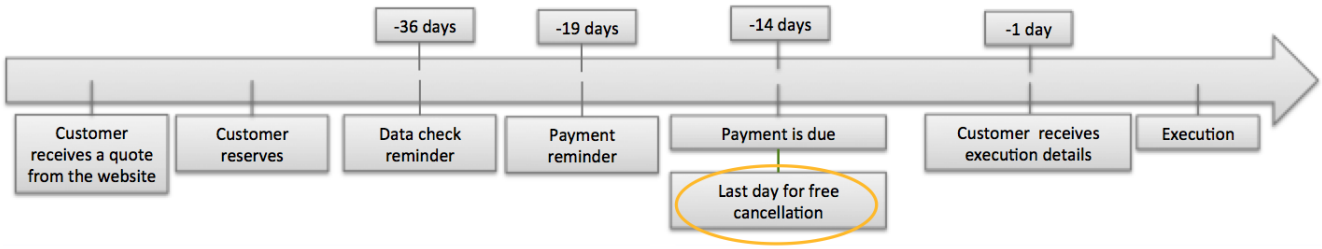

Figure 2: General booking process

Business perspective As soon as the customer reserves his request, which means that the contract is now legal binding and the customer receives a reservation confirmation,

FlixCharter will search for a matching bus. In order to do so, the customer request is then divided up in blocks, or also called trips. That means, if a customer wants to go from A to B, and then back from B to A, the request consists of two blocks. Then, per trip a bus will be assigned. FlixCharter does not own their own buses, but has agreements with other bus companies. That is, FlixCharter rents buses from bus partners far in advance, called rental orders. This construction provides a stable demand for the bus partners, and a certain bus capacity for FlixCharter. A trip assigned in a rental order can then be combined with other trips, which means that the same bus with the same driver drives a sequence of trips, also called a trip bundle. For example, a trip from customer 1 that goes from A to B and another trip from customer 2 that goes from B to A right after customer 1 arrives at point B could be bundled together. This would reduce idle time and empty kilometers of the bus, resulting in more profitable trips. The already existing so called TripBundler combines trips with rental orders.

As there is not enough capacity with only rental orders, a trip on the other hand can be sourced 1:1. This means that the customer request is handed over to the sourcing team who will try to find bus partner who takes care of the trip. The sourcing team would like to have the trip handed over as soon as possible in order to have enough time to search for a partner. While the optimization team would like to keep the trip as long as possible in the optimization pool in order to create trip bundles. The issue here comes when a trip may be not as reliable as they could expect, meaning that it could get canceled/declined. When a trip that belongs already to a bundle does no longer take place, the rental order has to be either assigned a new trip that matches with the lost one in order to keep the bundle as it is, otherwise it would have to drive empty kilometers. These kind of situations would be prevented when knowing in advance if a trip is likely to get canceled, if it is a common trip and if it has high chance of having big changes.

The main objective of this project is to evaluate how 'bundable' an incoming trip is, i.e. how much priority we should give the incoming trip for putting it in the TripBundler. Hence, we introduce TripEvaluator that outputs the following requested attributes:

- **Cancellation/Declination Probability.** The likelihood of a trip to get cancelled by the customer or declined by FlixCharter, which would affect the completeness of the bundle in case the trip is selected for bundling.
- **Change Probability.** The likelihood of a trip to get changed, which could affect a bundle if the trip happens to be chosen for bundling.
- **Plurality.** The plurality at FlixCharter refers to the attribute of a trip that describes how common or unique a trip is.
- **Sourcing Complexity.** How hard it is for a trip to get sourced in a 1:1 way.
- **Last/Best Time to Source.** The moments when the highest profit could be achieved (best time), and when we can still source without reducing the profit significantly (last time).

These attributes can serve as a guidance in order to decide which incoming trips to include for 1:1 sourcing and which ones to put in the TripBundler. The total phases now for an incoming trip are visualized in Figure 3.

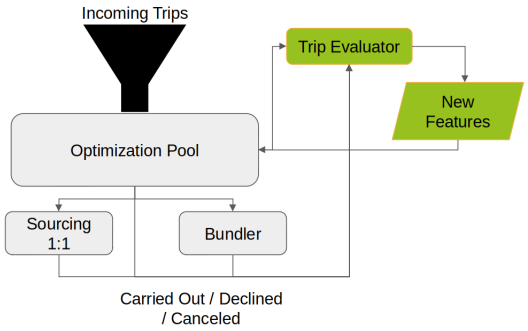

Figure 3: Bundling and Trip Evaluator integration

The remainder of this paper is organized as follows. First, Chapter 2 presents how we acquired the dataset, accompanied by some descriptives. Chapter 3 then elaborates on the methodology and results for creating TripEvaluator. Chapter 4 presents the conclusion, and describes next possible steps.

2 Data

This chapter discusses the dataset and provides preliminary insights into the variables.

2.1 Data Acquisition

FlixCharter extracted the information from a PostgreSQL-like database that they were using at the beginning of the project. The records were provided to us as XLSX (Microsoft Excel native format) files.

The structure of the data itself has been modified on FlixCharter's side during the project as the database has been migrated to a SnowFlake system. In the future, the obtained data has to be adapted in order for the TripEvaluator to keep operating.

The following datafiles were provided:

- *cr*: containing information regarding the customer requests, such as the origin, the creation date, payment type, etc.
- *trip_with_revenue*: data regarding the individual trips from a customer request. For example a round trip is comprised of 2 trips: one heading to the destination, and one coming back to the starting point. Data regarding coordinates and dates of the trips can be found here.
- *customer*: containing the encrypted names of the customers for each customer request.
- *cr_status*: a log for the customer requests, where all the status changes are stored along with the date when the change was performed.
- *breaks*: information regarding the breaks performed during the trips for each customer request.

- *cr_partner_requested*, po and cr_po_sufficiency contain information regarding the requests and the execution for sourcing 1:1.
- *trips_in_RO_file*: contains a list of all the trips that have been carried out in a rental order.
- *depot*: catalogue of depots with details of their location and the partner owner.
- *change_file*: information regarding changes made on orders by customers.

After all data aggregation and data preprocessing, the total data features on customer request and trip level can be found in the Appendix. A variable name that starts with TUM indicates that we created it.

2.2 Data Cleaning

During the data cleaning procedures several inconsistencies were observed, and handled, in order to be able to perform the data exploration and modeling.

By inquire with the Company it has been defined that the data collected before 2017 had a higher likelihood to contain errors, therefore all the customer requests with a creation date earlier than 2017 were removed from the set. For records concerning change file, records that did not have a reservation date corresponding to them were removed.

Furthermore, the optimization team pointed out that only the customer requests that happen to be reserved at some point are considered for bundling, therefore for our analysis we do not consider customers orders that have never been reserved during their timeline. The customer request (CR) timelines were obtained from the cr_status table.

By request we also removed all the trips with the flag cr_custom_solution_target_margin as according to FlixCharter these are orders that could highly skew the data. The trips belonging to this kind of CRs (which not necessarily have to be on the same request to belong to the same order) are usually major events. An example was given in which a company in Italy held an event for its employees in Rome, and they booked trips through FlixCharter going from their branches in other cities to the place where the event took place. This means that the behaviour for this trips wouldn't match to other similar ones.

Once we filtered the major CR table, the other tables were adjusted to keep only those "cr_id"s still included in the main table.

Afterwards, columns regarding dates and money were converted to a format usable within Python (i.e. dates in string format converted to DateTime objects. Strings with the coin symbol for money amounts) where converted to floats without coin identifier.

Among the information that was left out because of it being erroneous, we have a depot placed in Antarctica (see Figure 4), trips with no duration (arrival and destination point being the same) and trips with sourcing benchmark being negative.

We also checked the timeline of the customer requests in order to identify at what time were they reserved, booked, canceled or declined. In order to do that, we verified the details related to each customer request in *cr_status* table, and we inspected whether they were in correct chronological order according to our process understanding. If booked and reserved statuses were occurring more than once, we extracted the date of the first time when they were changed. For canceled and declined dates the last appearance was taken.

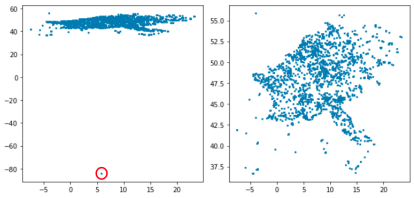

Figure 4: Visualization of depots placement with and without the Antarctica Depot

The dates reserved, canceled and declined can be already found on the CR file, but there was a time difference between them of 1 hour, so we took the date we considered from the status file and added 1 hour to all of them. Also, for each CR we calculated the days from each of these statuses to departure. Finally, we removed CRs which have a status change that happens after departure.

Presence of missing values and outliers distort the data distribution and might lead to less reliability of the results. For this reason, records that have a price margin and break duration higher than the upper 0.005 quantile or lower than the 0.005 lower quantile were removed. Regarding missing values, for features that were missing like the number of breaks, and break duration a 0 was inputed. Next to this, customer requests that have no specified *customer_type*, *paymenttype* or *trip_purpose*, are inputed as $UNSPECIFIED$. Likewise, for customer requests that were canceled or declined and no reason for this was specified, are inputed with $UNSPECIFIED$.

2.3 Data Exploration

As part of our exploration procedures, we inquired how the number of trips behaved over time. The objective was to be able to generate a classification where we could subset the data for some of our analysis.

The temporal classification was done by seasons, day of the week and time of the day.

By discussion with the Company, it was decided to use each day of the week (Monday, Tuesday, Wednesday,...) on its own, since they behave differently and the data wouldn't be distributed considerably as there are only 7 possible outcomes.

For seasonality, we inquired how the number of trips changes over the number of week. That is, we summed up all the trips with a departure day in certain number of the week in a particular year. This distribution can be observed in Figure 5

Afterwards, we aggregated the information by week number over all time, this way we could identify the different seasons over which the trips at FlixCharter happen. By inquiring with the Company, we found out there were two high seasons. This was confirmed by inspecting in the data segregated by year, and in the one aggregated only by week number (see Figure 6). As we can see in the Figure 6 there are two peaks reached at different times, and therefore we defined the moment when these increase and decrease as seasons, and the moments in between them as another 3 seasons. Giving us a total of 5 seasons going from week 1 - 13, 14 - 28, 29 - 34, 35 - 44 and 45 til the end of the year (which varies in number as the total of week depends on the year).

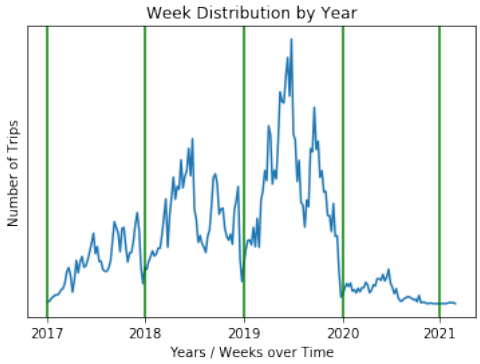

Figure 5: Trips distribution aggregated by number of week over the years [Number of trips removed for privacy purposes]

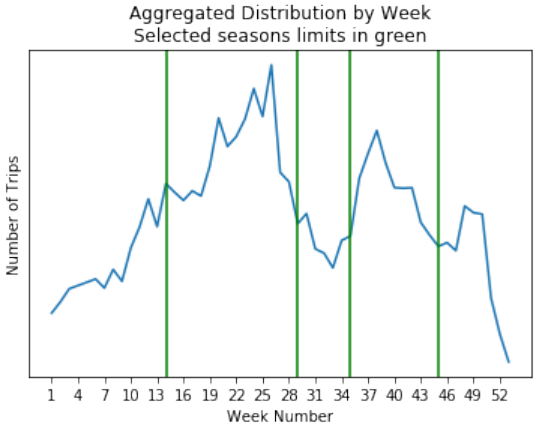

Figure 6: Trips distribution aggregated by number of week [Number of trips removed for privacy purposes]

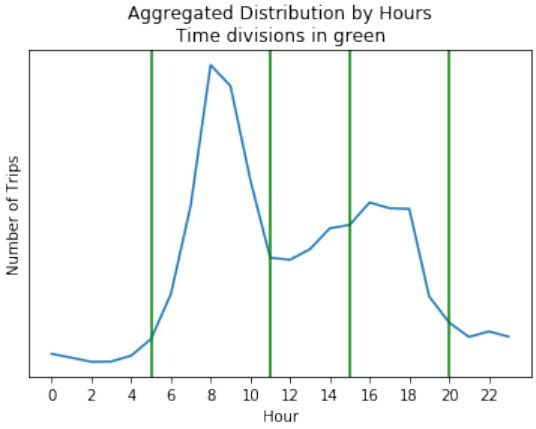

Figure 7: Trips distribution aggregated by hour of the day[Number of trips removed for privacy purposes]

Furthermore, we inspected the behavior of the amount of trips by hour, in order to define time frames depending on when the trips seem to have an increase/decrease of quantity, and group them up for our analysis. As observed in figure 7 there seems to be a low number of trips from 20:00 until 5:00, then from this time until 11:00 there's a really clear peak in the number of trips carried out, subsequently from 11:00 til 15:00 another different behaviour is observed (a slow rise) and finally from 15:00 to 20:00 we have another smaller peak.

These two defined ranges (seasonality and time of the day) will be used for further procedures we performed in our models, but they can be easily modified on the developed library in case better patterns are observed by the Company itself, or in case the behavior of this data changes over time.

3 Methodology and Implementation

This chapter will first provide the methodology for the general implementation of the library. Hereafter we go into further detail on methodologies and algorithms utilized for the inner functionality of the TripEvaluator.

3.1 General View of the Implemented Solution

The final expected product by FlixCharter was shaped once we have had a complete understanding of how the tool was meant to be used. We decided to build the requested TripEvaluator as a library. Once an object of this library is created, the Optimization department from FlixCharter expects the library to be able to perform the following activities with an instance from the class TripEvaluator:

- Extracting all the related data from their database.
- Use the current extracted data, to prepare models that should be able to compute the output variables.
- The tool should be capable of receiving a trip that FlixCharter wants to assess, i.e. to investigate whether it should be sourced 1:1 or bundled. Thus, the TripEvaluator should return the desired computed features related to the this trip.
- Once a CR has been concluded (Canceled, Declined or Carried Out) the library should be able to retrain the submodules with the new given data, this will allow the tool to be in a constant improvement through the time.

With the proper understanding of the wishes of the company, we designed a library that will be structured as defined in Figure 8. In said UML class diagram, we can observe the main class which is the Trip Evaluator, having all submodules (Plurality Calculator, Cancellation/Declination Probability Calculator, etc) being a part of it, so the modules (composites) should always be created as composition of the TripEvaluator, and not on it's own (the composition relation between them enforces so). For further details about how the UML class diagram is defined, we refer to Miles and Hamilton (2006).

There exists one DataPreparator class, for which the goal is to take the raw extracted data, and set it ready for the other submodules, as this should be done not for just one class, it's wise to keep it as an independent class from the calculators. The calculators can have several distinct methods, but all of them should have a constructor (which initializes the models or prepares the required data for computing the feature of other trips), a retrieveFeature (a method that for a given trip/CR calculates the required created feature) and a retrainModel/pushNewData function (method that retrains the models, or updates the given data, in order to provide predictions using the most recent available data).

The logic behind the defined structure, is to be able to execute the following algorithm properly:

1. The user creates an instance from the class TripEvaluator by importing it into a project

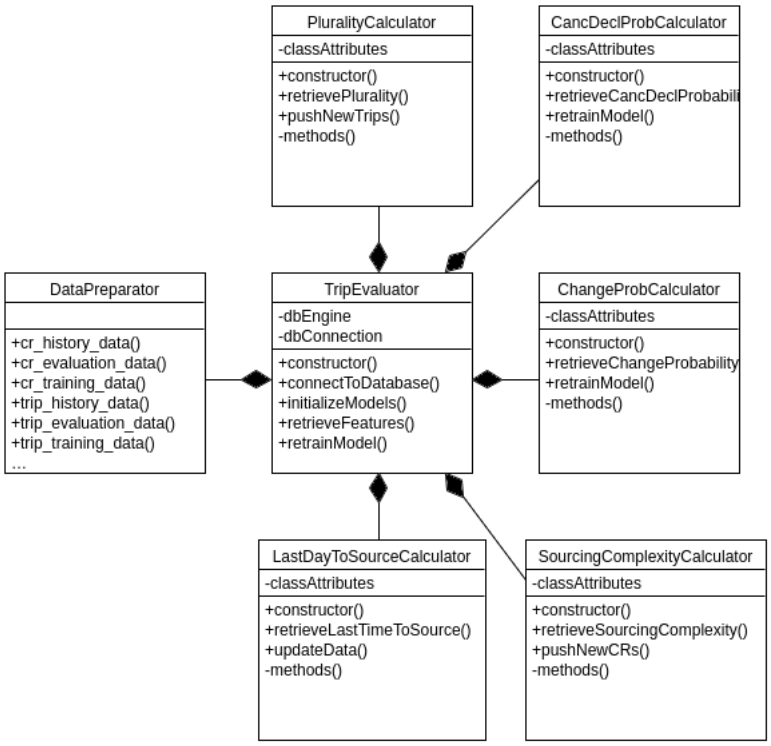

Figure 8: UML Class Diagram for Defined Library

2. Then, the user connects the object to the database through the method connectToDatabase() of the object

3. When the initializeModels() methods get called, first the data is extracted and then the XX_history_data() method is called for each of the types of datasets that the submodules required. Then an instance of each Calculator will be created given the historical data, and the models get trained for the first time (for those that require so) or the data is pushed to a pool containing the information, which will be used for future calculations

4. When all the modules are created, the user can call the retrieveFeatures() function given the trips that wish to be tested, and all the related information created by the TripEvaluator will be returned in form of a DataFrame.

5. Every time the user wishes so, the models can be retrained and the pools can be updated with the most recent information about the canceled/declined/carried out trips, making the accuracy of them higher

The different classes allow for some flexibility on the parameters they use, in this way the Company can adjust them as needed. Parameters as the seasonality definition can be sent into the class for them to be used instead of the ones that were defined as default. This is to prevent the need of hard-coding these parameters when in the future they are wished to be changed.

In consultation with FlixCharter, it has been agreed that the current state of the library will be certainly modified and mostly used as a basis for them to grow over. The amount

of data at the moment is fairly low, so as the time passes by, the models will require adjustments. A section of this report will be highly focused in all the areas of improvement that can be done for the TripEvaluator, and additionally the functionality of the code will be discussed on detail with the Company in order for them to be able to make modification when they require so.

3.2 Cancellation and Declination Probability

Being able to estimate the probability of a customer request to get canceled or declined is essential for decisions related to sourcing. Let us consider the following scenario: A route was formed as shown on figure 9. A bus for this particular rental order is located in F and needs to be returned there after the trip execution. Shortly before departure, a customer cancels the trip from A to B. Due to the low plurality of this trip and lack of luck, no similar trip appears in the optimization pool. That results in driving 'empty kilometers' to the departure point and, consequently, decreasing the margin.

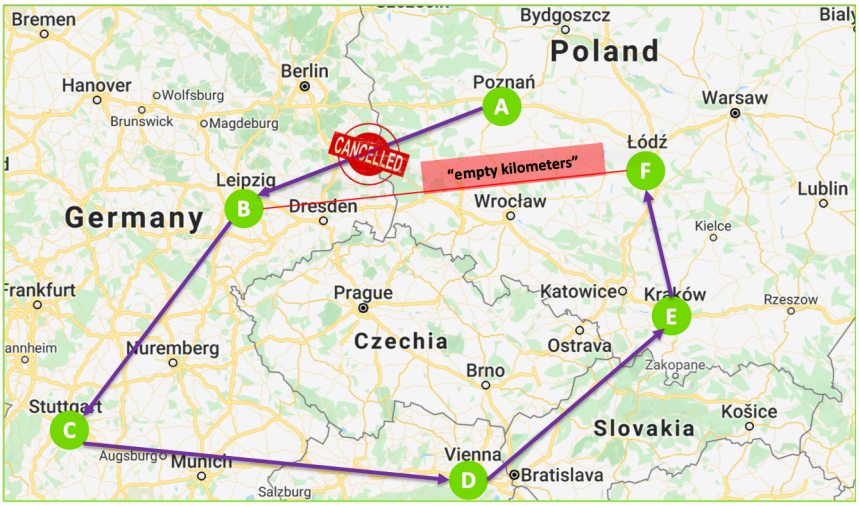

Figure 9: Impact of a canceled trip in a rental order on the revenue.

If a trip is likely to be canceled and takes place quite rarely at the same time, sourcing 1:1 should be considered. Here we can observe another drawback of lacking knowledge about the cancellation probability. In case of 1:1 sourcing, a reliable partner might be willing to stop providing buses to FlixCharter because of high rate of canceled deals. Therefore, not only the optimization team can exploit the cancellation and declination probabilities for their business, but also the sourcing team. Taking into account this additional feature, they can immediately create partner requests for reliable trips in order to possess enough time buffer for selecting profitable offers. On the other hand, it is reasonable to postpone the sourcing procedure to a later time if the cancellation probability is high.

Sometimes, FlixCharter employees can evaluate a customer request by looking at its attributes. For example, they might have had a customer that had a higher rate of cancellations in comparison to the executed trips, so they can deduct the high cancellation likelihood. Another useful feature to explain the reliability of a customer request could be the 'voucher redemption flag'. If a customer redeemed a discount voucher to book a trip, he tends to be more confident about his intention to travel.

However, it is very time-consuming to consider all attributes (about ninety) living in different tables for customer request analysis. In addition, the latent structure and the distribution of the data are not explicitly visible. The machine learning model is required to predict the final status of a customer request based on a set of relevant features. The possible instances of the final status are 'CARRIED OUT' (successfully executed trips), 'CANCELED' (cancellation on the customer's side), 'DECLINED' (cancellation on the Company's side) and 'EXPIRED' (no reservation after receiving an offer). Figure 10 displays all possible scenarios for status changes. As mentioned in Chapter 2, we only consider customer requests that were once reserved, so we only consider status timelines up from this point.

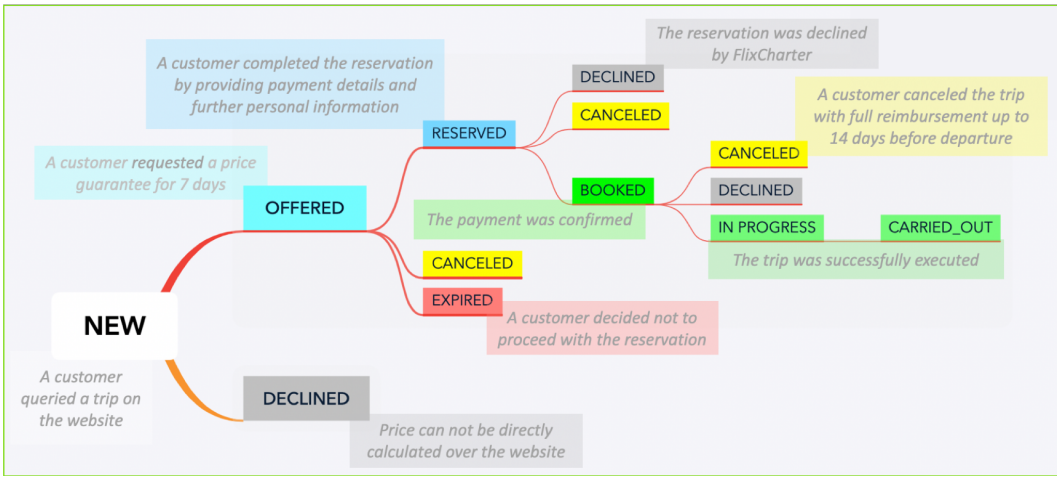

Figure 10: Timeline of possible status instances for a customer request.

The information about status changes with corresponding timestamps were provided by the Company. In an attempt to understand the customer behavior, we analyzed the status timelines of customer requests. By looking at Figure 11 we can observe when customers tend to make a reservation, booking or cancellation. For example, the peak of the green line around fourteen day before departure is explained by the obligation of a customer to pay before this deadline, otherwise the request will be declined by FlixCharter due to a missing payment. We also observe reservations and bookings made within 14 days before deparure, called instant bookings. However, those records do not play an important role for the optimization pool because the Bundler only considers trips with sufficient time buffer.

Going back to the main objective of this chapter, we encounter the classification problem where the model inputs are vectorized customer requests with its attributes, and outputs are class labels of the final status.

The first observation we can make by looking at the status bar chart on figure 12 is the imbalance of the data. An extreme case is represented by the class *DECLINED* where we possess only few records to make proper predictions. After communicating with the optimization team of FlixCharter, we decided upon the binary classification with two classes: *CAR_OUT* for the executed customer requests (label 0) and *CANC/DECL* for the ones that were canceled or declined (label 1).

Even with this adjustment the issue of imbalanced classes remains with proportion 3:1.

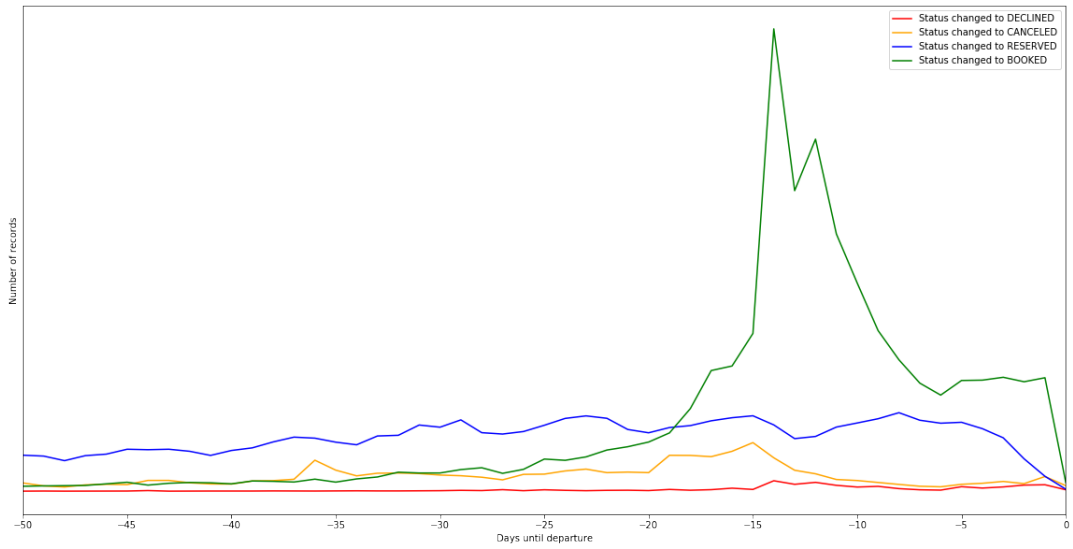

Figure 11: Occurrence of status changes of customer requests at day t before departure of the first trip.

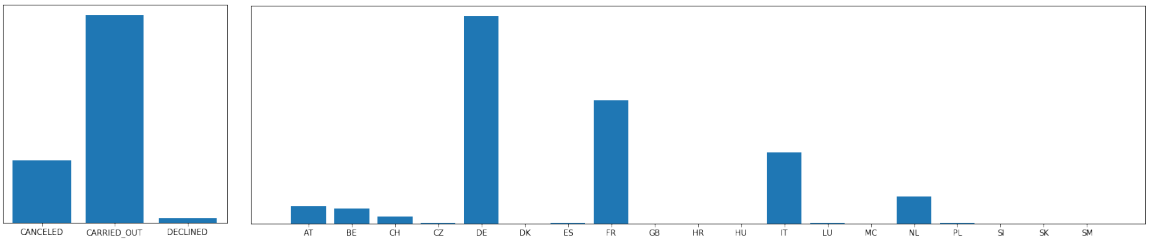

Figure 12: Distribution of customer requests (a) with different statuses and (b) across different countries.

This is a common obstacle in many classification settings including medical diagnosis and fraud detection. The results of machine learning classifiers trained on imbalanced data show poor predictive performance, especially for the minority class. In our setting, classifying customer requests with true label *CANC/DECL* as *CAR_OUT* populates the group of False Negatives (FN) which corresponds to error of type II.

The techniques designed for data preparation with imbalanced classes fall into two categories: oversampling and undersampling. Taking into account the amount of data we have at our disposal, undersampling was not considered. Oversampling techniques supplement the training data with records of the minority class until the unskewed data distribution is achieved. We applied two methods on the customer request dataset:

- **Random oversampling** naively creates random copies of the minority class.
- **Synthetic Minority Oversampling Technique (SMOTE)** has gained more popularity in machine learning and was first introduced by Chawla, Bowyer, Hall, and Kegelmeyer (2002). Minority points are added by introducing a synthetic observation with every minority observation along the line segments that connect to the k nearest surrounding neighbors. Depending on the amount of oversampled successes needed, k is chosen. The following formula shows how a synthetic observation is

created:

$$x_{new} = x_i + (\hat{x}_i - x_i) * \delta, \tag{1}$$

where δ is a random number between 0 en 1, x_i is the original data point and $\hat{x}_i$ is the nearest data point.

Since SMOTE oversampling resulted in higher performance measures for the classification models we applied, it was implemented using the python library ***imblearn***.

Before we go deeper into details about classification models, we recall from Chapter 2 that the python class ***CancDeclProbCalculator*** possesses the attribute *model* which is supposed to be a trained classifier allowing us to predict cancellation and declination probability of the constantly incoming customer requests. In the first step of the class constructor, the necessary features of a customer request are selected for the model. Even though the most part of data preprocessing was already conducted by the ***DataPreparator*** class, the minor adjustments are still needed.

Firstly, we defined the set of relevant numerical columns such as the number of passengers and special requirements, total duration of breaks, price ratio of the price the customer paid for his order compared to the market price and many others. Also the departure date gave rise to numerical attributes because it was encoded by counting the number of days passed since 01.01.2017 - the earliest date possible in the training data. Another remarkable feature of a customer request that we derived under the scope of customer analysis were the number of cancellations and executed trips made by a customer prior to the creation date of his request. Since numerical features exist on different domains, for each feature i the normalization should be performed. We decided to use the standard MinMaxScaler that can be later used as an attribute of ***CancDeclProbCalculator*** to normalize the new customer requests: $\hat{x}_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$.

Secondly, the categorical features need to be determined. Apart from attributes like payment type or trip purpose provided to us directly, we generated some additional flags. For example, one of them marks requests where the customer's language is different from the native one in the departure country. The idea for this feature was given to us by the FlixCharter employees who noticed that travel agencies from other countries tend to reserve buses in advance for the case they might have tourist groups in future. Since most fields like customer type or trip purpose, but not language, are optional while booking, this feature can not be captured by the model based on the existing ones.

While analyzing the customer behavior, we did not see the clear tendencies and correlations in the data until we started to divide it in segments for different countries, seasons and days of week. The figure 12 shows the distribution of orders across countries. Germany, Italy and France have the highest bulks and are clustered separately while all other countries are aggregated to a new cluster. With the recommendation of the Company's optimization team, the seasonality buckets were defined as well by grouping some year calendar weeks when a trip took place. The bucketing columns were appended to the categorical features. After collecting all categorical features, the one hot encoding was performed.

The last step before applying the classification model is to split the data into train and test sets following the best practice rule of 20 percent for the test set.

In the following we will present three classification models that we applied on the training data: logistic regression, neural network and random forest classifier. For each approach we will briefly introduce the methodology, implementation and optimization of hyperparameters for achieving the better performance. At the end of this chapter, we present the model comparison by evaluating classifiers.

Logistic Regression Classifier Logistic regression is a linear method for classification, which means that the decision boundaries are linear. It models the probabilities of the two classes, 1 (the customer request is likely be canceled or declined in the future) or 0 (likely to be not canceled or declined in the future), via linear functions in X, while ensuring that the probabilities sum up to 1 and remain in [0,1]:

$$P[y_i = 1] = F(x_i^{'}\beta) \tag{2}$$

$$\rightarrow f(t) = \frac{e^t}{(1+e^t)^2}, \text{the corresponding distribution function of F.} \tag{3}$$

The unknown parameters are estimated using maximum likelihood assuming the observations are independent:

$$\begin{aligned} L(\beta|\mathbf{y}) &= \prod x_i^{'}\beta^{y_i}(1-x_i^{'}\beta)^{1-y_i} \\ \rightarrow lnL(\beta|\mathbf{y}) &= \sum_{y_i=1} ln(f(x_i^{'}\beta)) + \sum_{y_i=0} ln(1-f(x_i^{'}\beta)) \end{aligned} \tag{4}$$

The value of $\boldsymbol{\beta}$ is estimated for which Equation 4 is maximized, i.e. setting the derivate to zero and solve for $\boldsymbol{\beta}$. The resulting $k+1$ (k variables and 1 constant) equations are non linear in β, and solved using the Newton Raphson algorithm to give the estimate $\hat{\boldsymbol{\beta}}$ (Hastie et al., 2004). The significance of individual explanatory variables can be tested by the usual t-test. This is an advantage of using logistics regression compared to other alternative classifying models as it gives a more useful description by providing marginal effects. However it should be noted that parameters of the logistic regression model are chosen to maximize the maximum likelihood, and not directly to maximize a measure of fit between the observed outcomes y_i and the predicted outcomes $\hat{y}_i$ by which it might perform worse in terms of classification.

The features used for estimating logistic regression are determined by forward feature selection. Forward Feature Selection is a simple algorithm that defines the best fitting parameters for a model and it can be simply implemented to several kinds of approaches. The algorithm starts by testing all the features individually and looking for the best accuracy. Then the parameter with best result is chosen, and all the remaining parameters are tested one by one in combination with the fixed feature. Then, the best pair is chosen, and the iteration continues adding parameters until the accuracy sees no improvement, or the are no further parameters to be added.

In the end, features that attribute most to the accuracy and had significant effect on the outcome to be canceled are: previous number of cancelled requests, total distance of the request, number of days before departure the customer reserved, number of buses, if the customer used a voucher, the customer request type, customer type, payment type, customer request origin, number of special requirements, and the type of luggage request.

Neural Network Classifier The main drawback of logistic regression is the fact that it can only learn linear decision boundaries. However, our data is complex and may require non-linear transformations. In the search for a more powerful classifier we attempted to train a Feed-Forward Neural Network (Bishop, 2013, pp.227-245) with 164 different combinations of hyperparameter settings. Varying the batch size, the number of epochs, layers and number of nodes in those layers, a compromise between high values for test accuracy (ACC), area under curve (AUC) and F1 score was desired (refer to the end of this chapter for details about the performance metrics). We observe that the highest F1 score is being achieved with a one-layer MLP. The closest approximation to the optimal set of hyperparameters is reached by fixing 24 epochs, batch size of 100 and one hidden layer with 70 nodes.

The architecture of our single-layer Neural Network Classifier is sketched on Figure 13. Each customer request in the input layer has 62 features after data preprocessing. We consider rectifier for the first activation function and sigmoid activation for the output layer. The outputs are cancelation/declination probabilities (values between 0 and 1). We use the binary cross-entropy between $f(x_i, W)$ and the true labels y_i as the cost function for backpropagation using gradient descent strategy. The model implementation was performed with the python library *Keras*, a high-level neural networks API.

$$E(W) = -\sum_{i=1}^{N} y_i \log f(x_i, W) + (1 - y_i) \log(1 - f(x_i, W))) \tag{5}$$

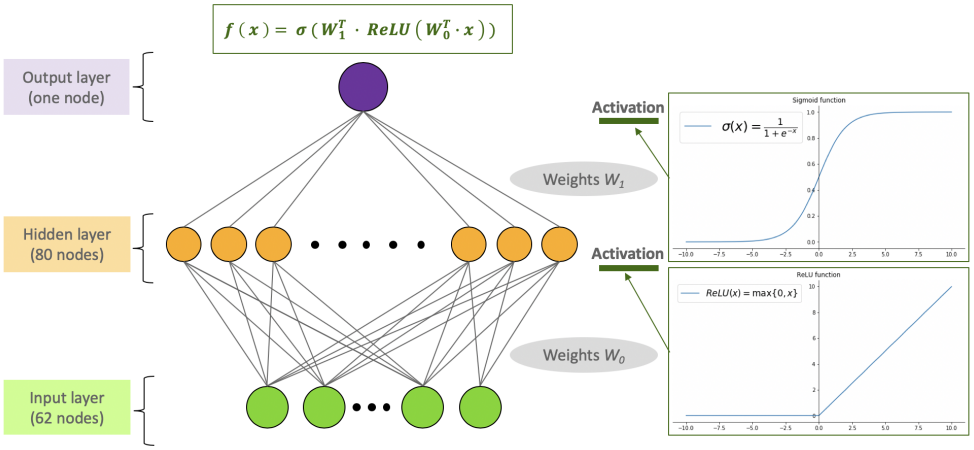

Figure 13: The architecture of our single-layer Neural Network Classifier with optimal hyperparameters.

Random Forest Classifier Random Forest, introduced by Breiman (2001), is a decision tree method which is a somewhat non-parametric method in nature and can be used for both classification or regression. A single decision tree summarizes the set of splitting rules for segmenting the predictor space into a number of regions (also called leaves). Recursive binary splitting is used to grow a classification tree, which means that starting from the top of the three it successfully splits the predictor space, where each split is indicated via two new branches further down the tree. It is called greedy, because

at each step of the of the tree-building process the best split is made, instead of looking ahead and picking a split that will lead to a better tree in some future step. The Gini index, a measure of total variance across the 2 classes, is used for finding the best splits:

$$G = \sum_{k=0}^{1} \hat{p}_{mk}(1 - \hat{p}_{mk}), \tag{6}$$

where $\hat{p}_{mk}$ is the proportion of training observations in the m^{th} node that belongs to the k^{th} class. If all $\hat{p}_{mk}$ are close to 0 or 1, then G will be small, and this will lead to more pure regions.

Random Forest applies the technique of bootstrap aggregation, also called bagging, to decision trees. The idea of bootstrap is to construct B bootstrap samples from the original data, where B is set to an arbitrary high number. For a given test observation, we record the class predicted by each of the B trees, and take the majority vote. Bootstrapping reduces the variance of the model without increasing the bias, which leads to a better model performance. Furthermore, when building each tree, at each split it can consider a random subsample of the predictors by which the trees will be uncorrelated (therefore the name Random Forest), and is by default set by $\sqrt{\text{nr of total features}}$. In general bagging and Random Forest are good methods for improving prediction accuracy, however the results can be difficult to interpret. An overall summary of the importance of each predictor can be presented by considering the total amount that the Gini index is decreased due to splits over a given predictor, averaged over the B trees. A large value indicates an important predictor.

A Random Forest model was implemented and optimized by performing a random grid search over the hyper parameters with 5-fold cross validation using GridSearchCV of SKlearn. Optimal results were found with $n\_estimators = 116$ (number of bootstrap samples), $min\_samples\_split = 2$, $min\_samples\_leaf = 1$, $max\_features =$ auto (meaning at each split consider $\sqrt{\text{nr of total features}}$ features), $max\_depth = 220$, $bootstrap =$ False. Interestingly, there is no bootstrap performed, so the whole dataset is used when constructing the random forests, instead of bootstrap samples.

Model evaluation All models are evaluated on the following classification metrics:

- **Area under the Receiver Operating Characteristics (ROC) curve (AUC)** Firstly, we compare on the AUC, which is a widely used accuracy metric for classification is. The AUC measures the area under the entire ROC curve and reflects how well the model is capable of distinguishing classes. An AUC score of 1 means that the model is able to distinguish classes perfectly.

- **Accuracy** Secondly, we consider accuracy, which is defined as the total correct predictions divided by the total number of predictions made. An accuracy of 1 means the classifier predicted the classes for all customer requests correctly.

- **F1 score** However, we should not be misleaded by high accuracy, since it could be that the classifier often misclasssifies the minority class. Therefore, we lastly

compare model results also on the F1 score, which is the harmonic mean between precision and recall. Precision is the number of correct positive results (canceled/declined) divided by the number of positive results predicted by the classifier. Recall is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as canceled/declined). Hence, the F1 score tells us how well the classifier predicts the canceled/declined cases, which is what we are most interested in. Here also holds, the highest and best value F1 can take is 1.

- **Running time** Since the amount of the Company's data is far from the definition of "big data", the algorithm running times were not stumbling blocks for achieving efficient results and, therefore, were not included into the model evaluation.

Results As observed in Table 1, out of all models Random Forest performed best, while the Neural Network outperformed Random Forest slightly with 0.01 on recall. Logistic Regression results in worst performance suggesting that linearly discriminating between the classes is not a good approach. Figure 14 also visualy points out how Random Forest outperforms Logisic regression and the Neural Network on AUC. Based on the overall results of Random Forest is the most promising model that FlixCharter would actually use for predicting cancelation/declination probabilities. However, we let the python class ***CancDeclProbCalculator*** to allow for the end-user to choose the classifier he wants to train by setting a particular value for the parameter *model_approach*: *'LogRegr'*, *'NeuralNet'* or *'RandFor'*, in order if someone is interested in the other model results as well.

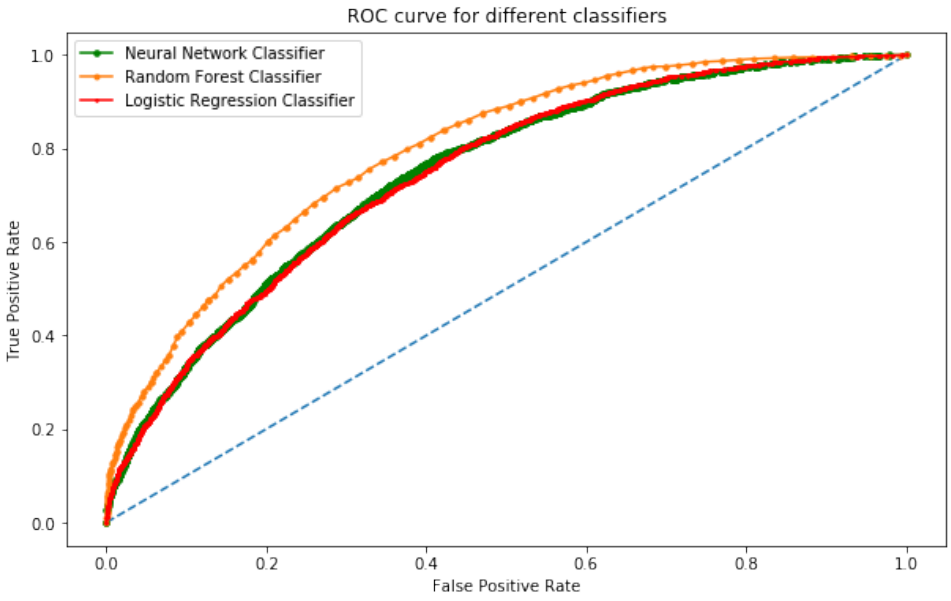

Figure 14: ROC curves for the implemented models.

Table 1: Model results

Model	Accuracy	AUC	Precision	Recall	F1 score
Random Forest	**0.79**	**0.8**	**0.57**	0.44	**0.5**
Neural Network	0.74	0.75	0.46	**0.45**	0.45
Logistic Regression	0.66	0.62	0.37	0.56	0.45

Model interpretability with Local Interpretable Model-Agnostic Explanations (LIME) Many FlixCharter employees from optimization, sourcing and sales departments are involved in operations for combining particular trips to tours (rental orders) and assigning other trips separately to partner offers. The cancellation and declination probability is a core feature for making those decisions effective and profitable for the Company. Whether one personally uses our machine learning classifiers as tools, or is directly forwarding the predictions to the Bundler, a vital concern remains: if the users do not trust a model, they will not use it. While in logistic regression the feature coefficients can directly reveal the relationship between input and output, Random Forest and Neural Network are seen as "black-box" models. In the latter it is possible to investigate activation units and to link internal activations back to the input. However, this requires a thorough understanding of the network and does not scale to other models. Here we propose to apply the LIME technique introduced by Ribeiro, Singh, and Guestrin, 2016 for asserting trust in individual predictions of a model as a way to provide a global acceptance.

LIME stands for **Local Interpretable Model-Agnostic Explanations**, meaning that it can be applied to any "black-box" machine learning model. It explains the predictions in a manner understandable to humans by learning an interpretable model locally around the prediction. Normally, the class of linear models G is assumed to be explanatory enough. LIME aims to minimize the following function:

$$\xi(x) = argmin_{g \in G} L(f, g, \pi_x) \tag{7}$$

In our binary classification setting, $f : \mathbb{R}^d \to [0, 1]$ is the probability estimator. Further, let $\pi_x(z)$ be a proximity measure between z and x. While $x \in \mathbb{R}^d$ is the original representation of an instance being explained, $x' \in \{0, 1\}^{d'}$ denotes a binary vector for its interpretable representation. LIME samples vectors around x' by drawing nonzero elements of x' uniformly at random. Given a perturbed sample $z' \in \{0, 1\}^{d'}$ (which contains a fraction of the nonzero elements of x'), the sample in the original representation $z \in \mathbb{R}^d$ can be recovered to obtain $f(z)$. $L(f, g, \pi_x)$ denotes a measure of how unfaithful g is approximating f in the locality defined by π_x, e.g.

$$L(f, g, \pi_x) = \sum_{z, z' \in \mathcal{Z}} \pi_x(z) \ (f(z) \ - \ g(z'))^2 \tag{8}$$

As you can see on fig. 15, the output of LIME is a list of explanations, reflecting the contribution of each feature to the prediction of a data sample. It allows to determine which feature changes will have most impact on the prediction. Although in this particular example the status was predicted correctly by both Neural Network and Random Forest Classifier, the latter is more confident about its decision. We observe that most features such as origin of a customer request (*website*), its type (*instant offer*), latitude of the

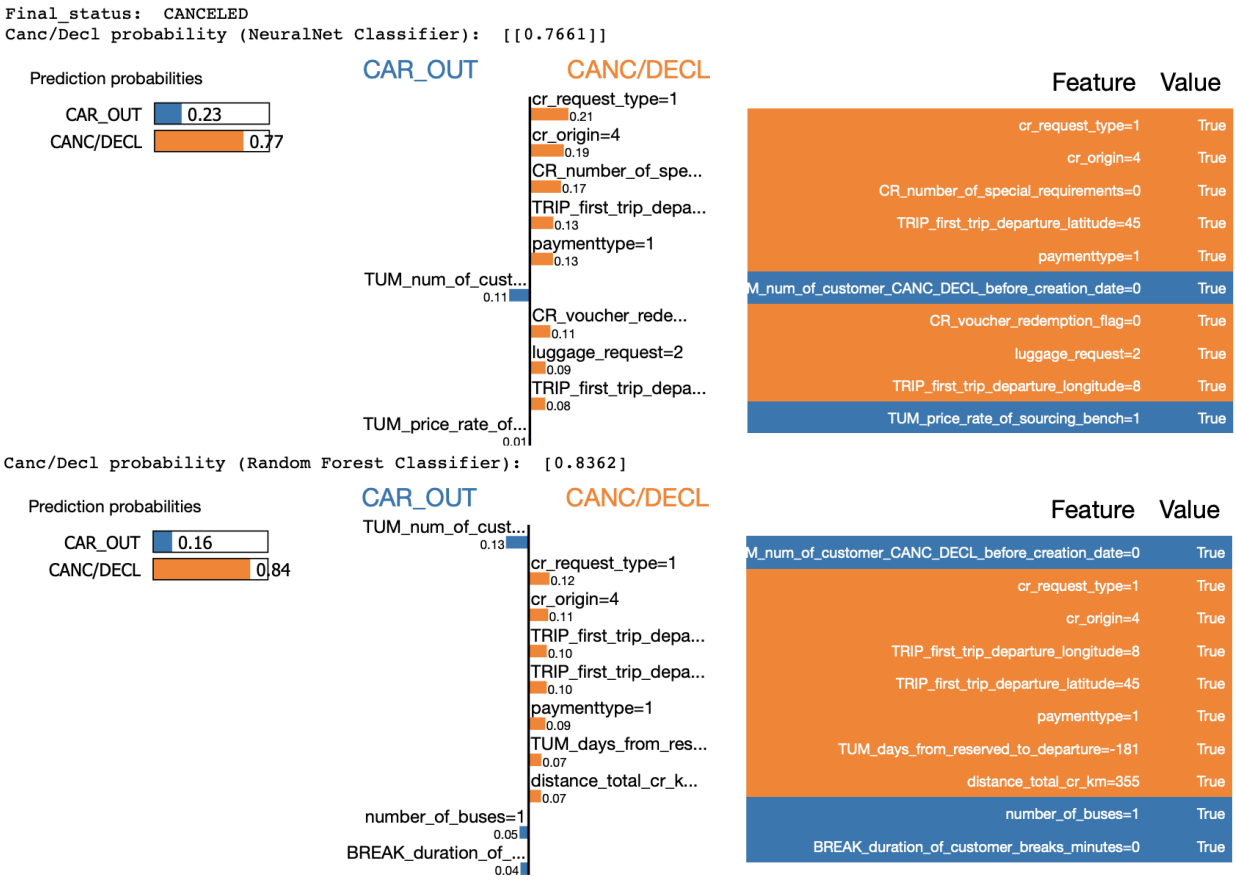

Figure 15: Example of the LIME output for one customer request with the final status *CANCELED*. The prediction was made with (a) Neural Network Classifier and (b) Random Forest Classifier.

departure location and many others 'vote' for label *CANC/DECL* while the minority of features such as the number of previous cancellations, fair market price or absence of breaks contribute slightly to the *CAR_OUT* probability. Note that in case of uncertainty the LIME report would indicate feature attraction to both directions revealing the atypical behavior of the data.

The user can call the method *limeExplain(CR, num_features)* of a ***CancDeclProbCalculator*** object to get the LIME report similar to the ones above with customized number of top features for a customer request CR. In the background the python library *'lime'* was deployed for the method implementation.

The main outcome of this chapter is the ability of ***CancDeclProbCalculator*** to immediately predict the cancellation/declination probability for an incoming customer request with 79% accuracy, as well as to increase the user's trust into the model by providing visual explanation for the predicted label.

3.3 Plurality

The Company plans to address the following scenarios while knowing this information:

- When a trip gets bundled, it is useful to know whether it is likely to get a trip that

could replace it.

- If a trip is really common, it can also be sent to sourcing as any could replace it.
- If a trip is really unique, it'll unlikely get bundled, and will be considered for a 1:1 sourcing instead.

The measure itself has not been defined before, and it is not possible to precise the exact plurality of a trip. By inquiring with the Company, we defined the following aspects that could play a role on finding a way to define the plurality of a trip:

1. The trip's distance to other trips.
2. How many trips take place at a similar time as the inspected trip.
3. The duration of the trip with respect to similar ones.
4. How many similar trips also belong to the same passengers' bucket.

The duration couldn't be properly integrated as consensus was not reach about how to define similar trips in duration and how to relate it with the other attributes from Plurality. The number of passengers was no longer required as a factor by discussing with FlixCharter as they wouldn't consider if a trip requires less or more buses than the other while replacing it, sourcing it, or bundling it. In order to calculate the similarity of a trip with respect to another regarding the distance, they have to be close to each other in both, departure points and arrival points. We came up with the following formula to assign the similarity of two trips:

$$Similarity = \frac{Distance(x_{departure}, y_{departure}) + Distance(x_{arrival}, y_{arrival})}{2dMax} \tag{9}$$

Where Distance is calculated by using the air distance which is the following:

$$a = (\frac{sin(x_{latitude} - y_{latitude})}{2})^2 + cos(x_{latitude}) \times cos(y_{latitude}) \times (\frac{sin(x_{longitude} - y_{longitude})}{2})^2 \tag{10}$$

$$distance = 6371.0000785 \times 2 \times atan(\sqrt{a}, \sqrt{1-a}) \tag{11}$$

Additionally, a factor of 1.2 was multiplied to the outcome. This was indicated by Flix-Charter, to be used in practice to make the result more accurate as while driving the exact path given by the air distance is not possible to be followed.

dMax would be number defined by the Company which indicates the maximum distance they would consider from departure and arrival points to substitute a trip with another. As of now a dMax of 200km was defined for the project, but the coding itself allows for specifying a different value.

The first challenge to be able to compute such information, is that the distance would have to be defined for each possible pair of trips. The complexity of this algorithm $O(2n^2)$

which would be also limited by the RAM memory as Pandas data frames are handled that way. We know in advance that we need to compute the distance only for a significantly lower of number of combinations as trips wouldn't be near all the rest of them.

A first approach using loops to compare all the trips and storing the results into a sparse matrix turned to fail, as while we increased the number of trips, the loading times exponentially grew until the result was either taking to long to be useful, or directly stopping at a certain moment when the RAM was overloaded.

Then, we decided to implement a bucketing approach, since we know that the latitudes and longitudes of the points can already indicate how far are the points from each other, (even though, it wouldn't be accurate at all), then we could find the trips in the same buckets and only calculate the distance from those. Reducing highly the number of computations.

We first inspect how much the latitude/longitude has to change in order to achieve the dMax distance.

We observed that a change of 1 point in latitude represented an approximate distance of 100km and 1.5 points was the equivalent for longitudes.

The distance from (48.264960, 11.667933) and (49.264960, 11.667933) is 111.19 km which is 1 point increment in latitude. The distance from (48.264960, 11.667933) to (48.264960, 13.167933) is 111.02 which is 1.5 points increment in latitude.

Note: The number of points changes depending on the position on the globe, so we placed our tests in an area in Europe to try to be as accurate as possible.

These values are used as factors in the following way: $Factor \times dMax/100$

This way, we know the extreme case in which dMax could be achieved, looking as shown in Figure 16

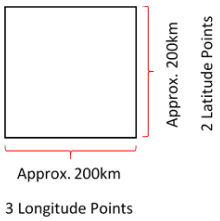

Figure 16: Space coverage with current bucketing

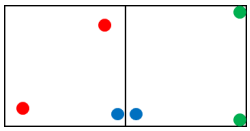

Figure 17: Trips visualization in buckets

Meaning, that if we assign a group (buckets) to each latitude and longitude from both, arrival and departure, we could find all the trips that within one group are at most approximately 20,000 km (the diagonal) in both points, but assuring that a trip that goes from above to below, or from left to right without inclination, gets considered.

The issue faced while doing this, is that if a point is placed at the border of a group, and another point really nearby is placed at the border of the order bucket, it wouldn't be taken into account, such as visualized in Figure 17.

The red points are expected to be further than 200km, but still in the same bucket to be calculated, the green points are at most 200km and in the same bucket, so they are considered, the blue points however, seem to be close to each other, but as they belong to different buckets they wouldn't be considered.

Therefore, we need to compare the points that are also neighbours with respect to their buckets. This procedure can be easily performed by using an "explode". This basically retrieves the same row (which would be a trip) several times, only with a changing column which in this case, will receive the actual bucket, a bucket below and a bucket above. In this way, when we join this column to itself, we would find all the trips that are in the same bucket as the target trip, or in a neighbour bucket. Afterwards we proceed to calculate the distance from both trips, and if the distance exceeds the dMax, the record gets dropped. All the joining procedure is done in an SQL database to reduce the workload on RAM. In Figure 18the a simple time comparison of up to 300 trips is done, we observe that calculating all trips distances is faster than generating the bucket when the number is reduced, but as the amount increases it's much more slower. Considering not only few trips' distances will be computed, the join bucketing approach performs highly better.

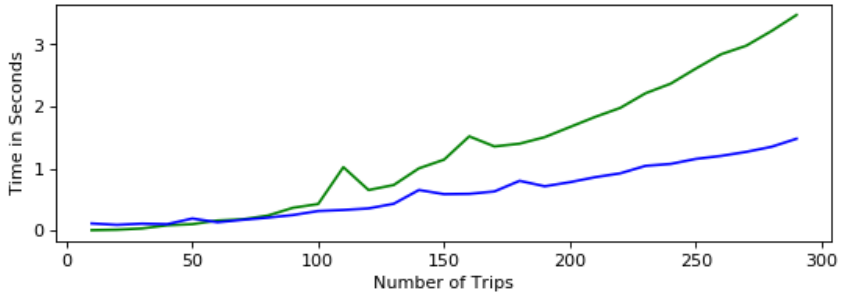

Figure 18: Performance of bucket approach (blue) against all-to-all approach (green) for distance calculation

Once all the combinations of trips, where the departure points and the arrival points are no further than 200 km from each other, we filter all those trips that are on the same temporal frame.

By inquiring with the Company, we defined the following aspects that should match while considering two trips being on similar times:

- The trips should happen on the same day of the week (e.g. Monday, Tuesday...)
- The season (week number range) should match. We inquired based on the number of trips distribution for defining the week ranges we would use for each trip.Refer to section 2.3
- The departure time also has to be in the same range, for this we performed the same procedure for identifying the number of trips happening at each hour and assigning ranges of hours to assign the buckets. Refer to section 2.3

Once we have all the trips combinations where the temporal and distance similarities match, we proceed to calculate the plurality.

As a first approach, we calculate the mean of all the trips within the same distance/time buckets, and then give a penalization to those trips with a reduced number of similar ones, but as there was no straightforward way of how to assign the penalization, we came up with a different implementation.

As we can consider the trips as nodes connected to each other when they are considered similar, and the similarity itself being the weight of that edge, we could implement a Page Rank algorithm in order to assign the plurality of each trip.

Page Rank Page rank is an algorithm developed by Google which was used to set the ranking that would be used for displaying the websites on their searcher. In short, each link from a website to another, represented an incoming edge to the pointed website. The more incoming edges a page had, the higher the ranking. An important remark is that the weight of an outgoing edge is decided depending on the number of incoming links; that is, a link coming from a small personal blog wouldn't worth as much as one from Wikipedia. A broader description of the approach can be found in Meyer and Langville (2012).

This approach is completely relatable to the objective of plurality calculation since the trips would behave as pages, and the more similar trips they have, the more common (the higher the rank) they have. The resulting rank would be a number between 1 and 0, which could be interpreted as the closer to 1 the more common, and the closer to 0 the more unique. There were some observed trips to have a really high number of similar trips, which gave them a really high page rank resulting on most of the trips having a lower plurality, this was solved by removing the outliers and adjusting the ranks without them, afterwards a plurality of 1 would be given to all the observed outliers.

Results As no accuracy could be tested for this feature, in order to inspect whether the feature was reasonable or not, we retrieved new trips information after the date of the last extraction (which is dated to November) and checked for all the new trips that were reserved at least once. We selected the first 100 records out of a total of 9326, and checked whether the selected trips had similar ones according to the parameters we established for distance and time. We compared whether the plurality seemed to go up according to the number of available trips in the future. The results can be seen in Figure 19

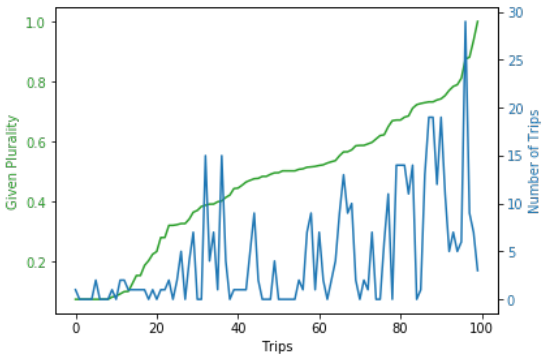

Figure 19: Tested plurality over new 300 records

We observed that for the range lower than 30 percent there was none to few trips, so we could say new similar trips were unlikely, while above 75 percent, there were always at least 3, with the maximum of a trip having 29 similar trips being at 0.87 plurality, which could be interpreted as new trips are likely to happen. This information is not so clear

when the range is in-between. But the current results can work as a guidance for such trips with pluralities in a lower and higher range.

3.4 Change Probability

At any point of time following a customer request, a customer might make some changes in their order. These changes can range from something irrelevant to the tripâ^{TM}s place in the bundler, like changes regarding "Luggage Request" to something significant like a change in "Destination Stop". As such, similar to cancellation and declination, it is vital to be able to predict a significant change in order to foresee whether or not the trip should be considered for the bundler. As an example we consider a trip A that has been bundled. A certain change made it impossible for the same trip to be carried out with the bundler and has to be removed. Depending on when the trip was removed, the bundler might not be able to replace it with another trip, resulting in empty kilometers.

An underlying aspect about changes is that while a customer can make near about any change in their order, not all changes are treated equal. In fact, many of the changes do not play any effect on the place of the trip in the bundler. Additionally, unlike cancellation, a change in a trip makes it possible to still be sourced 1:1 without repercussions from the partner as, by its nature, the trip still takes place but only needs to be reconsidered for the bundler. Hence, if a trip has a high likelihood of significant changes made, 1:1 sourcing should be highly considered.

Ground Truth Generation While the type of changes made may vary, for a change to be considered significant, specific in-house criteria needs to followed. The criteria is specified to reflect exactly what kind of changes need to be taken into account that would threaten a trip's position in the bundler. In order to perform analysis, these ground truths needed to be generated.

- Request made for Bike slots - owned buses cannot accommodate this request
- Request made for ski boxes - owned buses cannot accommodate this request
- Specific request for on-site bus
- Arrival Time change - If the new arrival time varies from the old arrival time in the range of 30 minutes (give or take)
- Departure Time change - If the new arrival time varies from the old arrival time in the range of 30 minutes (give or take)
- Departure Stop - The new location from where to depart has to have its similarity recalculated
- Arrival Stop - The new location for where to arrive is has to have its similarity recalculated

- Number of passengers - Change in the number of passengers depends on the size of the buses available for the bundle. A change that does not require a change of bus size and/or only requires another bus of the same size does not need to flagged as a significant change as the bus continues in its bundled route

Analysis against the generated ground truth proved to be tricky. No inherent pattern could be observed between the calculated significant changes feature and other features provided by the company. The customer behaviour with regard to significant changes seem to stem from factors beyond what the data can show us. Much of the distribution against significant changes follows the distribution of the total changes which suggests a simple reflection of the quantity of changes made overall as shown in an example in Figure 20 where number of significant changes follows the distribution of total changes which in turn follows the distribution of overall trips made during the periods.

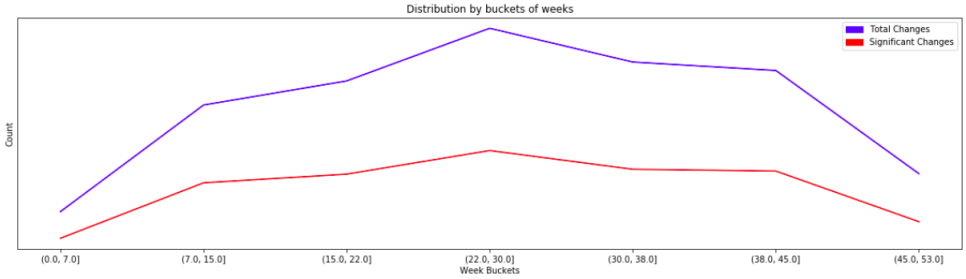

Figure 20: Distribution of Significant changes and total changes over week buckets

Another complication is that, unlike cancelation, a trip can still have a change made in it after a change has already been made. In fact, there is technically no limit to the number of changes one can make on the same trip and that brings a bias to what it means to have a change probability. Since the change probability inherently depends on the whim of the customer there does not exist a real threshold to when a change can be made. Figure 21 denotes the erratic behaviour of the number of changes made with respect to the number of the days left to the departure.

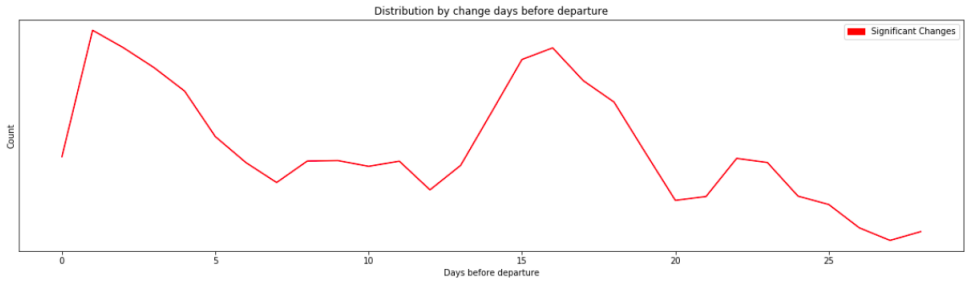

Figure 21: Distribution of changes across last 30 days before departure

This highlights the idea that grounds for changing remains abstract and at the complete whim of the customer. The nature of the customer itself, annotated by âœcustomer typeâ has been observed to play a small role with its relatively high correlation with the significant changes made. Specifically, institutional trips (University, Schools, Company) tend to have slightly higher chances of making changes. This could be credited to the

earlier booking nature of such institutes before the specifics have been properly ironed out. The relationship is small. But amidst highly uncorrelated and undescriptive data, it is worth mentioning.

The relatively highest correlation that could be observed between whether a trip has previous changes already made. There is a likelihood that another change might be made. This behaviour can be understood by the fact that a customer is shown to understand the capability of making a change and is therefore, if need be, likely to make a change again.

Modeling and result
Features selected and extracted for fitting on a model in order to see if latent features could be learned from. As before, Logistic Regression and Fully-Connected Neural Network Classifier were used to check whether or not any patterns can be picked up on that were not clear on the surface.

Table 2: Model results

Model	Accuracy	F1 score
Neural Network	**0.68**	0.48
Logistic Regression	0.58	**0.57**

3.5 Sourcing Complexity

This section discusses two approaches for determining the sourcing complexity for an incoming trip. If the customer request is given up for sourcing, the sourcing team will offer the trips to their bus partners, whereafter the bus partners can respond and provide a price offer. Sourcing complexity of an incoming customer request is relevant to know in order to decide if the request should be included in the Bundler or not. Trips that are hard to source are preferred to be kept in the optimization pool, as finding a bus partner would be hard, or less profitable than using a rental order.

Heuristic The first approach is a more heuristic way of determining if an incoming trip will be hard to source or not. Based on past customer requests that were sourced 1:1 and for which we have all possible attributes, we calculated the following variables:

- Partner response rate (number of sufficient partner offers divided by the number partner requests that were sent).
- Profit margin (profit divided by the benchmark price).
- Number of days before departure a partner was found.
- Distance from departure to closest partner depot.

For an incoming request we can then find the mean for above attributes from other requests that had the same number of passengers, departed from the same country and departed at the same calender week, weekday and hour. There remain some pitfalls when assessing above mentioned features:

i) The higher the response rate, does not necessarily mean it was easier to source. For example, the sourcing team may have received many partner offers, however all of those offers could not have been the target price they wanted to go for and thus therefore they had to wait long until a competitive price offer was offered.

ii) Another drawback is when analyzing the distance of the closest partner depot is that we do not have any data on how active all the partners are. It could be that a partner depot is very close, however that partner might almost never drives for FlixCharter.

iii) The smaller the number of days before departure a partner was found, does not have to indicate that it was hard to source as it could also have been that the sourcing team had set the trip aside for a while, since they knew for sure that they could find a partner offer at any time.

For the aforementioned drawbacks on assessing the features that should indicate if a trip was hard to source, we also attempted an approach to come up with a model-based solution for obtaining the sourcing complexity of an incoming customer request.

Clustering As a second approach we employed a semi-supervised machine learning model. By requesting the Company, we obtained 45 customer requests that were hard to source, and 25 that which complexity was low (Not all the elements could be used since some of them have been removed from the final set during data cleaning). With that we attempted to implement the clustering algorithm from K Nearest Neighbours (KNN). It is a classification algorithm that given a set of labeled elements and a number K, chooses the class for a new element. This is done by calculating the euclidean distance between the new element all the already labeled elements, and finally the mean class of the K closest elements is given. (Bishop, 2013)

The current features used for the clustering are customer type, payment type, pax, number of buses, number of trips, total distance of CR in km, duration of all trips, price all net, non standard flag, sourcing benchmarks, number of special requirements in CR, sum of trip sourcing benchmarks, first trip departure latitude, first trip departure longitude, last trip departure latitude, last trip departure longitude, distance between departure and arrival, calendar week from departure, weekday from departure and hour of departure.

13 elements from each (hard and easy to source) have been randomly chosen for initializing the clusters, and then the rest of the customers requests that were sourced 1:1 (labeled and not) were added for performing a semi-supervised learning. The results for the remaining labeled items have been tested for several amount of possible number of neighbours, none of them showing reasonably good results (Table 3) and as no reasonable amount of labeled elements has been obtained, no further procedures have been performed.

FlixCharter's optimization team is currently waiting for further elements labeled from the sourcing departments, this could be further tested in order to observe whether there's an improvement in the approach. At the moment, the tool only displays the the created metrics previously discussed in this chapter.

Table 3: Results for Complexity on KNN

Number of Neighbours	5	6	7	8	9	10	11	12	13	14	15
Accuracy Reached	56%	40%	64%	28%	52%	32%	60%	4%	36%	16%	40%

3.6 Last/Best Time to Source

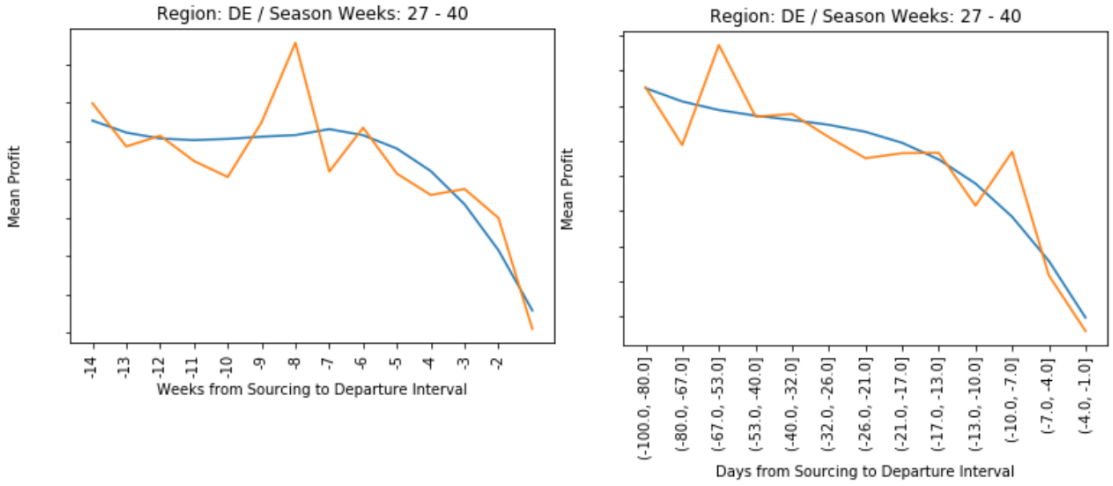

Figure 22: Comparison between Equi-width and Equi-depth bucketing [Profits removed for privacy purposes]

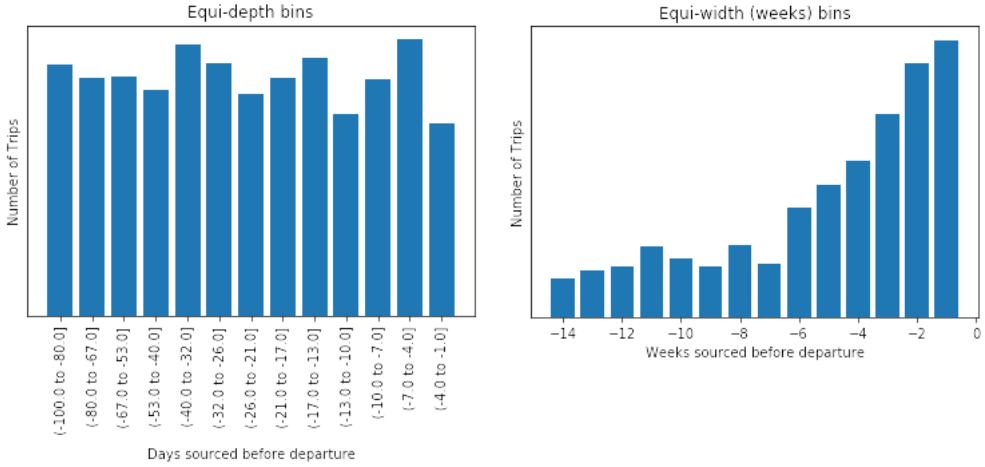

Figure 23: Comparison between Equi-width and Equi-depth bucketing [Number of trips removed for privacy purposes]

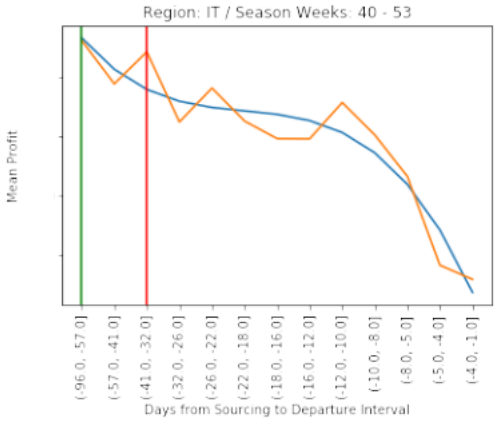

Figure 24: Last (red) and best (green) moments to source using the current approach [Profits removed for privacy purposes]

As no particular measure could be found, FlixCharter provided us with the current approach they were considering for finding out the last and the best time to source.

As it was done already, the data was split in four groups, which are Germany, France, Italy and other Countries. Then, for those trips carried out with a Partner Offer, they inspected how many days before departure the trip was sourced, and afterwards grouped them by this number. Finally, on each group (trips on same region and number of days of sourcing before departure) they calculate the mean margin obtained on each day, and finally inquired what was the last time to source before the margin got reduced significantly.

This first approach defined by the company was further developed by us in order to reduce the current time frame they have for sourcing, since currently it was above 6 weeks. We did so by firstly segregating the data further more, but this time by seasonality, since the time they have to source with a good margin highly depends on whether the trip is carried out during high or low season. By plotting the results we observed the behaviour showed no clear pattern, and sometimes a really high/low margin could be attained by a reduced price. Inquiring with the Company we settled that the best measure for this approach would be to take the mean profit, that way small profits regarded with high margin wouldn't have the same weight as a trip with a really high profit but small margin.

Once we could observe the behaviour through the days, the data was too variable in order to be able to decide which day was the best/last time to source. It has been observed that the number of trips in each day was really higher as the departure date was closer, therefore it was needed to get the mean over more than just one day. The options for doing so was on one hand grouping the results every 7 days (equi-width), or attempt to have equi-depth buckets. By request of FlixCharter it has been decided to use the second one for our approach. The comparison of both approaches can be seen in Figure 22, and how the number of trips distributes through the bins can be seen in Figure 23

Finally, we inquired with the Optimization team that the best time to source should be given by the highest bucket. The last time to source was computed by fitting a polynomial of degree 3 on the data and then setting the highest point as the highest income (usually to be the first point) and then calculating when it gets reduced by a certain percentage (in the sample case 25% was chosen) and setting the closest bucket as the last time to

source. If this date happened to be earlier than the largest bucket before smoothing the data, the last day to source would be chosen to be the same as the best one. An example of how the results would be seen by the user can be observed in Figure 24

The tool itself allows the user to set the regions, the seasons and the percentage to be used, this way it can be adjusted by the company in case the trends change when more data is included. When a trip is given to the module, the suitable bucket for it will be given and the last and best time to source will be returned.

4 Conclusions

By using a combination of data cleaning methods, machine learning models and heuristic approaches, we developed a tool that enables FlixCharter the possibility of getting a better insight of their trips.

The library is able to extract the information that the company currently possess and then return the desired features for evaluating the trips.

For cancellation and declination probability, we have reached good accuracy by implementing Random Forests, which is already above 75%, which mean they can be taken already into account for assessments while deciding what action to be taken on a trip regarding this matter.

Regarding plurality, the current approach's precision cannot currently be measured. However, by testing new data, the possibility of observing whether a trip is highly common, or unique can be done when low or high plurality ratio is attained. Meanwhile for the middle range of plurality the insight is not as good. This could be improved with more data, but requires more follow up in order to get to know if the approach gets better or worse with the amount of data.

Concerning the sourcing complexity, at the moment, the company is only taking some metrics into account for deciding themselves. These are given in a more readable way, but no real evaluation of how hard or easy to source is being done. A clustering implementation could return good results with a higher amount of labels given, but as it is now the results achieved are rather uncertain. FlixCharter has already informed us, more data about this has already been requested in order for them to give follow up to this implementation.

In regards of the last and best time to source, it is required to be further defined how can one decide these moments, since no precise condition has been given, the current approach is flexible for them to try several combinations of regions, season and profit loss percentages for them to find out appropriate parameters.

As of now, the final product reached during this project is not meant to be used as it is forever, but as a starting point upon which the Optimization team can build upon. The possibilities offered by the data over which the Company operates, are really promising, although the current amount of information didn't allow us to achieve the best possible results, the current indicators show that the way to go is well shaped. A change in FlixCharter's perception about keeping tiny data has also been seen over the last year, which encourages the capability of improving the efficiency of the currently established methods on this paper.

References

Bishop, Christopher M. (2013). *Pattern recognition and machine learning*. Springer.

Breiman, Leo (2001). "Random Forests". In: *Machine Learning* 45.1, pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

Chawla, Nitesh et al. (2002). "SMOTE: Synthetic Minority Over-sampling Technique". In: *J. Artif. Intell. Res. (JAIR)* 16, pp. 321–357. DOI: 10.1613/jair.953.

Hastie, Trevor et al. (2004). "The Elements of Statistical Learning: Data Mining, Inference, and Prediction". In: *Math. Intell.* 27, pp. 83–85. DOI: 10.1007/BF02985802.

Meyer, C. D. and Amy N. Langville (2012). *Googles PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press.

Miles, Russ and Kim Hamilton (2006). *Learning UML 2.0*. OReilly.

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2016). ""Why Should I Trust You?" Explaining the Predictions of Any Classifier". In: *KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144. DOI: 10.1145/2939672.2939778.

Wang, Gang et al. (2016). "Big data analytics in logistics and supply chain management: Certain investigations for research and applications". In: *International Journal of Production Economics* 176, pp. 98 –110. ISSN: 0925-5273. DOI: https://doi.org/10.1016/j.ijpe.2016.03.014. URL: http://www.sciencedirect.com/science/article/pii/S0925527316300056.

Appendix

Table 4: Variable descriptions on customer request level

Variable	Variable description
cr_id	Customer request id number, unique identifier for every request.
Customer type	
customer_name	Encoded name of the customer.
customer_type	Type customer, e.g. School, Club, Company, etc.
customer_language	Language of the customer who made the request.
trip_purpose	Trip purpose, e.g. Excursion, Day trip, Event, Wedding etc.
pax	Number of passengers.
TUM_prev_number_canceled_after_res	Number of previous canceled customer requests.
TUM_prev_number_carried_out	Number of previous carried out customer requests.
TUM_prev_number_expired	Number of previous expired customer requests.
Customer request characteristics	
cr_origin	From where the customer request was made, eg. chat, email, outbound sales, phone or website.
cr_request_type	instant booking, instant offer, no offer, or price range.
cr_creation_date	Date when request was made
cr_request_status	Last status known about the customer request, e.g. booked, canceled, declined etc.
cr_reservation_date	Date when the customer request was reserved, if this is the case.
cr_cancelation_date	Date of cancellation, if this is the case.
cr_cancelation_reason	Reason of cancellation, if it the request was canceled.
cr_declination_date	Date when the customer request was declined, if this is the case.
cr_declination_reason	Reason of declination, if the request was declined.
cr_booking_date	Date of booking.
cr_voucher_redemption_date	Date of voucher redemption, if this is the case.
paymenttype	How the payment was made, e.g. with credit card, on account or with SEPA.
Trips characteristics	
number_of_trips	Number of trips within the customer request
distance_total_cr_km	Total distance of all trips in the customer request in km.
departure_bucket_days	Rounded amount of days between arrival and departure of the first trip.
departure_date_first_trip	Date of departure for the first trip in the customer request.
TRIP_first_trip_departure_time	Date of departure for the first trip in the customer request (from trip file).
TRIP_first_trip_departure_longitude	Departure longitude of the first trip in the customer request.
TRIP_first_trip_departure_latitude	Departure latitude of the first trip in the customer request.
departure_country_first_trip	Country of departure for the first trip in the customer request.

departure_region_first_trip	Region of departure for the first trip in the customer request.
TRIP_last_trip_arrival_time	Arrival time of the last trip in the customer request (from trip file).
arrival_date_last_trip	Date of arrival for the last trip in the customer request.
TRIP_last_trip_arrival_longitude	Arrival longitude of the last trip in the customer request.
TRIP_last_trip_arrival_latitude	Arrival latitude of the last trip in the customer request.
arrival_country_last_trip	Country of departure for the last trip in the customer request.
departure_city_last_trip	City of departure for the last trip in the customer request.
departure_region_last_trip	Region of departure for the last trip in the customer request.
TUM_dist_between_dep_and_arr	Distance between the first trip departure and last trip arrival of the customer request.
duration_all_trips_hours	Total hours of all trips in the customer request.
number_of_buses	Number of buses needed for the customer request
wheelchairrequirement	Dummy, 1 if a wheelchair request was made for the customer request.
bikesslotrequirement	Dummy,1 if?
busonsightrequirement	Dummy, 1 if ?
skiboxrequirement_id	Dummy, 1 if a skibox request was made for the customer request.
luggageinbusrequirement_id	Dummy, 1 if luggage in the bus is required.
non_standard	Dummy, 1 if it is a non stand customer request.
SEGMENTS_number_of_segments	Total number of segments in the trips of the customer request.

Partner variables

PO_main_partner_id	Name of the main partner.
PO_main_partner_country	Country of where the main partner is situated.
PO_sum_po_taken	Sum of values of all partner offers if ¿1 partners taken.
PO_last_po_taken	Date when the last partner was taken.
PO_number_of_taken_po_for_cr	Number of partner offers for trips that were actually sourced 1:1.
PO_SUFFICIENCY_partner_price_net_min	Minimum price among the received partner offers.
PO_SUFFICIENCY_partner_price_net_max	Maximum price among the received partner offers.
PO_SUFFICIENCY_partner_price_net_avg	Average price among the received partner offers.
PO_SUFFICIENCY_number_of_po	Total number of partner offers for the entire customer request.
PARTNER_REQUESTED_number_of_pr	Total number of requested partner offers for the entire customer request.
PARTNER_REQUESTED_first_pr_sent	Date when first partner offer was requested for the entire customer request.
PARTNER_REQUESTED_last_pr_sent	Date when last partner offer was requested for the entire customer request.
TUM_po_profit	Sum of all sourcing benchmarks prices minus sum of all values of the partner offers.
TUM_po_revenue_margin	Profit divided by the sum of all sourcing benchmarks.
TUM_days_from_pr_sent_to_last_po_taken	Time between date when the first partner offer request was made and when the last offer was taken.
TUM_days_from_pr_sent_to_departure	Time between when a first partner offer was requested and the date of departure.
TUM_days_from_last_po_taken_to_departure	Time between the when a last offer was taken and the date of departure.
TUM_partner_response_rate	Total number of partner offers taken divided by the total number of partner offers requested.
TUM_dist_from_dep_to_closest_partner_depot	Distance between departure point and closest partner depot.

Table 5: Variable descriptions on trip level

Variable	Variable description
trip_id	Trip id number, unique identifier for every trip.
cr_id	Customer request id number that belongs to the trip id.
Trip characteristics	
CR_cr_status	Last status about the customer request that belongs to the trip.
DEPOT_closest_partner_depot_id	Id of closest depot.
DEPOT_closest_partner_depot_longitude	Longitude of closest depot.
DEPOT_closest_partner_depot_latitude	Latitude of closest depot.
TRIPS_IN_RO_sourcing_type_of_CR	If the trip was sourced with a partner (PO), rental order (RO), mixed (MIX), or not at all (NOT_SOURCED) (because the trip was declined or canceled).
Other price variables	
price_all_net	Price the customer pays for the customer request.
sourcing_bench	Market price for the customer request.
TUM_sum_of_trip_sourcing_benchmarks	Sum of all market prices of the trips in the customer request.
passengers	Total passengers of the trip.
region	Region where the trip was made/departs ?.
trip_departure_latitude	Departure latitude of the trip.
trip_departure_longitude	Departure longitude of the trip.
trip_arrival_latitude	Arrival latitude of the trip.
trip_arrival_longitude	Arrival longitude of the trip.
trip_departure_time	Date of departure of the trip.
trip_end_time	Date of arrival of the trip.
trip_distance	Total distance in km of the trip.
trip_segment_duration	Total driving time of the trip.
trip_break_duration	Total break time of the trip.
SEGMENTS_number_of_segments	Total segments within the trip.
TRIPS_IN_RO_sourcing_type_of_trip	If the trip was sourced with a partner (PO), rental order (RO), or not at all (NOT_SOURCED) (because the trip was declined or canceled).
Price variables	
trip_sourcing_benchmark	Target revenue of the trip.
trip_revenue	Actual revenue of the trip.

Table 6: Variable descriptions of customer request statuses

Variable	Variable description
cr_id	Customer request id number.
change_date	Date of the status change.
status	Status of the customer request: reserved, booked, canceled, declined, carried out, done.