Budapest University of Technology and Economics

Faculty of Electrical Engineering and Informatics

Department of Networked Systems and Services

Customer Case Analysis with Machine Learning

Scientific Students' Association Report

Author

Anna Lotz

Advisor

Gábor Horváth

2019

Contents

Kivonat

Az adatok manapság mindenütt jelen vannak. Ezek elemzése és összegyűjtése segíthet olyan gépi predikciók létrehozásában, amelyek könnyíthetik az üzleti életet.

A multinacionális cégeknek számos ügyfelük van, akik nap mint nap a termékeiket használják. A partnerek bármiféle felmerülő problémával az ügyfélszolgálathoz fordulnak. A probléma nehézsége lehet egyszerű, pár perc alatt megoldható, de akad olyan, mely megoldásához több hónapra is szükség van. Ha nem tudják az ügyfélszolgálaton helyben megoldani az esetet, akkor vonják be a mérnöki csapatot, vagyis eszkalálják az ügyet, de még így is hónapokig elhúzódhat, mire a problémát lezárják. Ezek az eszkalációval járó esetek mind a cég, mind az ügyfél számára rendkívül drágák.

A projekt célja egy olyan rendszer létrehozása, mely az ügyfélszolgálati adatokat tároló adatbázisra támaszkodva, gépi tanulás segítségével meg tudja becsülni, hogy egy adott eset megoldásához várhatóan szükség lesz-e a mérnöki csapat bevonása, illetve hogy az eset előreláthatólag hány napig lesz nyitva. Ezekkel a predikciókkal korábban felismerhetjük a problémásabb ügyeket, ezekre felhívhatjuk az ügyfélszolgálati csapat figyelmét, akik célzott módon, magasabb prioritással kezelhetik a várhatóan nehéz ügyeket, korábbiaknál gyorsabban le tudják zárni azokat, így rengeteg erőforrást megspórolva.

Abstract

Data is everywhere nowadays. Collecting and analyzing data can help us discover hidden patterns in the data set, making it possible to create predictions that can make business life more manageable.

Multinational companies have a lot of customers who use their products every day. In case the customer runs into any problems related to these products, the support team of the company addresses the issues. The complexity can vary dramatically from simple cases requiring just a couple of minutes to resolve to very complex ones stretching over several months to get resolved. If the support team can not solve the case, they escalate it to engineering for help. These complex cases can remain open for a potentially long period of time. These situations can be very expensive and frustrating both for the customer and the company.

My research goal is to develop a machine learning tool based on the database storing the historical customer cases. This tool predicts whether the resolution of a case will involve the engineering team, and also predicts the expected time till the resolution of the case. These predictions help us to recognize and assign high priority to the problematic situations earlier. Concentrating more effort towards the resolution of these problematic cases leads to faster resolution times, implying fewer resources in terms of money and time.

Chapter 1

Introduction

1.1 Concept

The amount of data we produce every day is truly amazing. Collecting and analyzing data can help us discover hidden patterns in the data set, making it possible to create predictions even for business situations.

Multinational companies have a lot of customers who use their products every day. In case the customer runs into any problems related to these products, the support team of the company addresses the issues. The complexity can vary dramatically from simple cases requiring just a couple of minutes to very complex ones stretching over several months to get resolved. If the support team can not solve the case, they escalate it to engineering for help. These complex cases can remain open for a potentially longer period. These situations can be costly and frustrating both for the customer and the company.

My research goal is to predict

- if a case will be escalated,
- and how long it will be open.

The first one is a classification, and the second is a regression problem. With these predictions, we can recognize the problematic cases earlier and can save resources.

The Cloudera software company helped me by providing the necessary data and software as well as industry knowledge to develop this idea in a real business environment. Cloudera is an enterprise data cloud company that offers a software platform for data engineering, data warehousing, machine learning, and analytics [1]. Its main commercial products are CDH, a software distribution consisting of many cloud-related open source components, and CM (Cloudera Manager), the software responsible for managing the components of CDH.

1.2 Execution plan

The project got divided into the following phases:

1. Preparation
2. Data Collection
3. Data Analysis
4. Creation of a Machine Learning models

In the preparation phase, I studied the related literature, and I found that this project is very company-specific. Thus, I had to learn how the support team and the engineering team work together at Cloudera. Next, I have discovered the data that exists, and that could be important for the Machine Learning (ML) model. Then, I have analyzed the data set, created the feature set for the model, and investigated the correlation between the features and the target variables. I created the final data set after the data cleaning for both Machine Learning models. The last step was to find the suitable Machine Learning model for both prediction goals, to try several models, then compare them and select the best algorithm.

Chapter 2

Related work

There are not too many studies available in the literature that are analyzing support cases and whether we can make predictions with the help of Data Science techniques. We found an MSc thesis made by Lloyd Montgomery in the topic of *Feature Engineering* [11]. He worked within IBM's Ecosystem and created a Machine Learning model to predict support ticket escalations. His main project goal was to predict whether a case will be 'CritSit' ('Critical Situation' process that involves third-party employees to help completion faster). Classification of incoming support tickets is a method to assist support analysts in recognizing which tickets require the most attention. He tried out different Machine Learning models, including CHAID (Chi-square Automatic Interaction Detector), SVM (Support-Vector Machine), but the Random Trees (a different name for Random Forest) was the most suitable model. For that research, he used 10-fold cross-validation to provide more consistent and robust results. 10% of the data was labeled as 'testing' and roughly 90% of the data was marked as 'training'. The thesis did not mention the data cleaning part, so it is not known whether the data set is valid. In my case, cleaning the data could lead to significant improvements in the model, as without cleaning the model has learnt from false data. The feature set he proposed for his Machine Learning model is an excellent base for our investigation. I added some other features to the collection based on the different operation model, process, and the opinion of the support team. There are significant differences between IBM's and Cloudera's operation model and tools for support tickets. Therefore adjustments and extensions to the features set are required. As opposed to his model, for me, it is very important that my model is scalable.

2.1 Operation model at IBM

At IBM, support process involves multiple levels: L0, L1, L2, and L3. L0 is the first level where the support collects the necessary information about the issue, but the contact has no technical knowledge to address the issue. L1 Support has the basic technical knowledge, and they can solve simple problems. L2 Support is the first level where they specialize in IBM products, which means that they have more experience to resolve the issues. L3 Support is the last level, which is a mediator between the support and the engineering team, but there could be developers in the L3 support team as well.

2.2 Operation model at Cloudera

Contrarz to the support organiyation of IBM, Cloudera support team does not distinguish the traditional four levels at Cloudera support team. The team consists of two levels: the frontline and the backline team.

The members of both teams are engineers, and we call them Customer Operations Engineers. Based on their knowledge, they try to solve cases that are related to different components. Every frontline member knows the basics of 3-4 components. If they can not solve an issues, they can escalate it to the backline team, who knows much more about that component. Usually, the members are good at one component, but not every component has a backline member. If the situation is more complicated or there is no backline member for that component, then the support team escalates the case to the development team of the given software component. When they identify and fix the problem, the customer operations engineer can close the case. The solution of the problem could take several weeks because it is not easy to find the source of a problem, reading the log files could require a lot of time, and the communication is not always fluent.

Chapter 3

Preparation

In this section, I would like to present what platforms and tools I used to collect data. A necessary part of the project is the learning phase. For the data collection, I have learned to use two platforms, one to obtain the information (Hue) and one where I can visualize them and create a ML model (CDSW). I had to learn how the database structure looks like, and from where I can get the necessary data.

3.1 Hue

Hue is an open source workbench for data analytics and discovery. With intelligent query assistance and query recommendations, it is an easy-to-use SQL development tool [10]. Hue provides a very simple interface to run SQL queries. We can easily search and discover tables and views across all databases. The editor is pluggable and compatible with any databases or query engines, for example, Apache Hive, Impala, MapReduce, or SparkSQL. I used Apache Impala as the editor. If we have complex queries, then we can make a view, which makes it easier to have the results updated without rerunning the query. Hue can do basic visualization (bar plot, line plot, or pie chart), but I never used it because I wanted to make more complex diagrams; therefore, I used Cloudera Data Science Workbench.

3.2 Cloudera Data Science Workbench - CDSW

Cloudera Data Science Workbench is a web-based platform for collaborative data science. In it, we can work in multiple computer languages - including Python, Scala, and R - and we can use it directly from a web browser. Built using Docker containers, CDSW offers data science teams per-project isolation and reproducibility, in addition to the easier collaboration.

In the workbench, I have used the Python programming language and imported the Python modules necessary for data analysis including:

- **Numpy**, a module supporting fast array operations,
- **Pandas**, a module implementing DataFrame, which is a two-dimensional tabular data structure. Pandas DataFrame consists of three principal components, the data itself, rows, and columns [9].
- **Matplotlib** and **seaborn** has rich tools to visualize DataFrames as graphs (including scatter plot, line plot, bar plot),
- **NLTK** means Natural Language Toolkit, which is a leading platform for building Python programs to work with human language data,
- **Scikit-learn** (sklearn in short) to prepare data for Machine Learning models and to run the Machine Learning models themselves.

3.3 Database structure

For every machine learning project, we need to collect all available data that can influence the prediction, and from these data, we need to create a feature set. However, this requires knowledge of the structure of the database. I used five data tables whose relationship is shown in Figure 3.1. This diagram shows a simplified version of the tables because the entire database structure may contain sensitive information. All tables are linked to the case table as the purpose of the model is to make a prediction to every case. I used the users, history, comments, and escalations tables. The users table represents every person that can create a comment for a case, even the customers and the support team members. The history table records all changes for one case, for example, changes in priority, in the owner, or status. The comments table contains all comments, the created date, and the user id that created them. Every escalated case can be found in the escalations table with the type, created date and the id of the case. There are four types of escalation: Customer, Management, Backline, and Engineer.

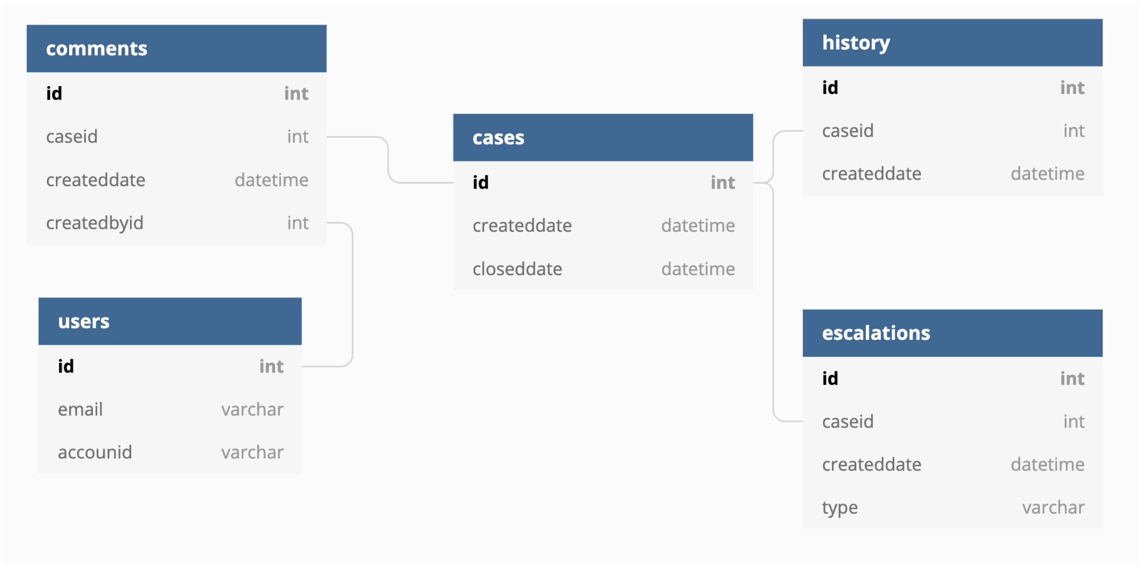


Figure 3.1: Database Relationship Diagram

Chapter 4

Data Discovery

For every Machine Learning (ML) project, a very important part of the project is data collection and analysis because every ML model requires good features. The data collection and analysis is often 90% of the work.

In this section, I would like to outline the collection of features and the processing of the data.

4.1 Feature set

A necessary step for every ML model is to create a feature set which is a collection of relevant information that can make an impact on the prediction. The starting point was the feature set identified in [11]. Through Hue, I'm able to reach all databases storing the information related to the support cases and the customers. I created SQL queries to extract these features from the database of the company and saved the results to SQL views.

The feature set based on IBM's model can be separated into four main categories:

1. Basic properties
2. Perception of process
3. Perception of time
4. Customer profile

4.1.1 Basic properties

This category contains all the basic features about a case. It includes the following features that I could directly obtain from the tables.

Days open (*type: numeric*) Number of days while the case was actually open. Cases that are still open are ignored.

Number of actions (*type: numeric*) Action means that something happens with the case (for example customer responded, the solution proposed, etc.) and it will be recorded in the database as a status update. I count how many times this status changes to get this feature.

Component name (*type: string*) Cloudera's platform consists of many different components and every case is in relation to one or several of those components. I used a Label Encoder to convert the labels into a numeric value and thereby bring them into a machine-readable form.

CDH major/minor/maintenance version (*type: numeric*) Cloudera's platform (CDH) has been released in several different versions over the years. We believe that the different CDH versions are important because different versions have different levels of quality. The version can be divided into three parts: major, minor and maintenance versions.

4.1.2 Perception of process

During the process (as the case evolves over time), there are a lot of factors that can determine the outcome of the cases.

Number of people in contact on Cloudera side (*type: numeric*) The number of people involved in the case is relevant because the more complicated a case is, the more people are involved in the solution, thus the more time is necessary for the communication.

Increase/Decrease in priority (*type: numeric*) The increase of the priority indicates that the case is more important. Over time, as the solution gets near, the priority is often decreased. In complicated cases the priority is increased and decreased many times until the case gets solved.

4.1.3 Perception of time

The cases can be open for a long period of time that is why the time is one of the most important factors.

Time until first contact in minutes (*type: numeric*) The time until the first contact after case opening is determinate because this is the first step of the process. I studied how much time it takes until the first comment created on Cloudera side.

Average support response time (*type: numeric*) We also looked at the average of support response time. To obtain average support response time, I have collected the case comments, and every time a support member answered a customer message, I calculated the time difference between the comment creation dates. Then I took the average of these response times.

Difference between avg and expected support response time (*type: numeric*) Every customer has an expected support response time based on the average support response time of all its cases. The difference between the average time for a specific case (`Average support response time`) and the average time of all the cases of a customer could be different and could refer to the complexity of the case.

4.1.4 Customer profile

We believe that each customer is unique and therefore the behavior of their cases will be unique too. Therefore we want to capture this in a customer profile by looking at:

Number of closed/open cases/escalations (*type: numeric*) How many closed and open cases and escalations the customer has.

Ratio (*type: numeric*) If a customer has many escalations but not so many cases, then the rate between the number of escalations and cases can affect the prediction. The ratio is calculated in the following way:

$$ratio = \frac{number\ of\ escalations}{number\ of\ cases}$$

Number of open/closed cases/escalations in the last X months (*type: numeric*) Number of closed/opened cases and escalations in the last X months (X=0,1,2,3,4,5,6).

Expected support response time (*type: numeric*) Expectation of support response time based on the average response time considering all their cases.

4.2 Survey

The Cloudera frontline support team is working every day with cases. They therefore have probably a good feel on which features could be important for my predictions. I therefore wanted to ask them for their opinions by creating a survey. With it we wanted to achieve the following goals:

1. Validation of our assumptions
2. Validate whether the feeling expressed by support people matches with our findings in the model
3. Discover new features that should be added to the feature set

In the survey, the question was ‘Do you think the following features are important?’ and there were four possible choices available: I do not know/ Not important/Maybe important/ Important. In the survey, I separated my feature set into two groups because the first is about a case and its process and the other group is about the customer.

I also asked the following free form questions: ‘What other factors do you think might help to predict how long a case will be open?’ and ‘What other factors do you think might help to predict whether a case will be escalated?’.

4.2.1 Results

In Figure 4.1, we can see the result of the survey (basic properties, perception of time and process). The survey was filled out by ten support team members. Most of the support team members think that the features are important or maybe important except the increase and decrease in priority and the CDH version.

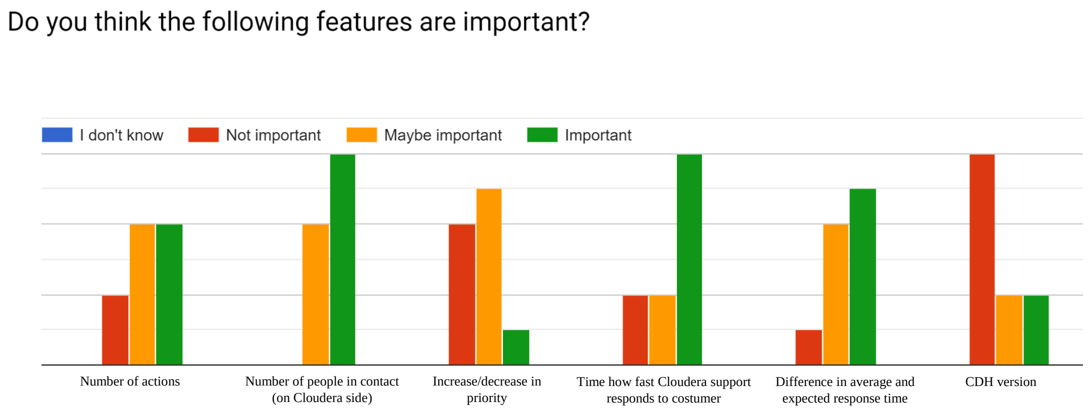


Figure 4.1: Features about the case

Figure 4.2 depicts the features about the customer profile. The responses for these features were not so uniform than the results in Figure 4.1. After running the ML model (discussed later) we are going to evaluate the feature importance and we can validate whether the feeling expressed by support people matches with our findings in the model.

Do you think the following features are important?

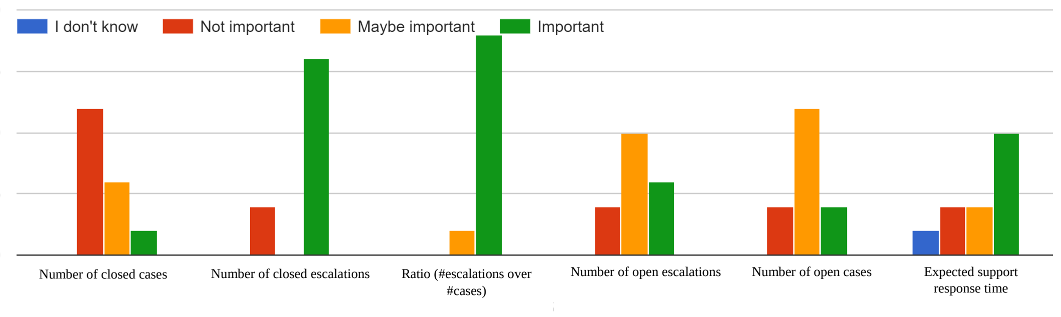


Figure 4.2: Customer profile

I added four new features to my model based on the responses of the support team.

Number of people in contact on customer side (*type: numeric*) Besides the number of people in contact with the customer on Cloudera side, they thought that the number of people on the customer side in contact could be important too. It happens that the case is open for many days because support is waiting for the customer answers so I counted the average customer response time.

Average customer response time (*type: numeric*) I counted the average customer response time for each case in the same way as in average support response time.

Expected customer response time (*type: numeric*) I calculated the expected customer response time for each customer, that includes all the cases of the customer.

Difference in average and expected customer response time (*type: numeric*) Every customer has a response time based on the average response time of all its cases. The difference between the average time for a specific case (`Average customer response time`) and the average time of all the cases of a customer could be different and could refer to the complexity of the case.

We received a lot of promising answers to the free form questions. Somebody said that we should study the keywords: 'intermittent', 'sometimes', 'occasionally', 'not reproducible'. Other said that the keywords should be 'ASAP', 'immediate'. I would like to discuss the free form answers in the next section.

4.3 Textual features

Based on the suggestion from the survey, I started to examine if there is a difference in the words used in escalated and non-escalated case comments. To check this assumption, I made word clouds, one from the escalated case comments and the other from the non-escalated case comments. A word cloud is a cloud of words, where more frequent words are displayed in a bigger font. Before feeding any textual data to the word cloud, first, I preprocessed my data by taking out stop words (for example, I, you, we, that, what, be, again, and too). I expanded the stop words with frequently occurring words, for example, Cloudera, help, thanks, regards, issue, work, etc. For calculating word frequency, I used the nltk module, which includes the common stop words.

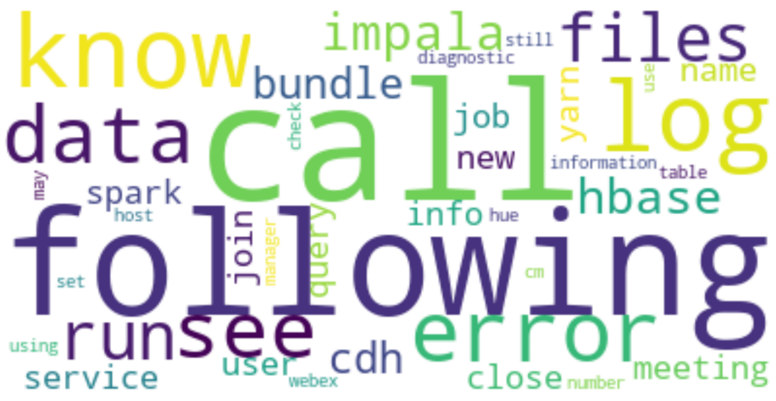


Figure 4.3: The 40 most common words of the comments of escalated cases

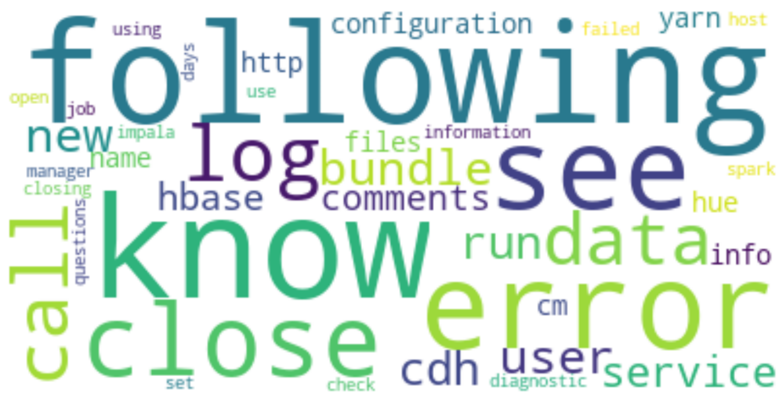


Figure 4.4: The 40 most common words of the comments of non-escalated cases

If we look at the Figure 4.3 and the Figure 4.4, we can realize that the words are almost the same, but there is a difference in frequency. The words ‘call’ and ‘files’ are more common in the escalated case comments than in the non-escalated case comments. In contrast, the words ‘close’ and ‘error’ occur more often in the non-escalated case comments. We can observe some different words in the two charts. In Figure 4.3, there are words like ‘meeting’, ‘still’ or ‘may’. The escalated cases are more sensitive; therefore, it is understandable using more polite words like ‘may’. In Figure 4.4 there are words like ‘open’, ‘failed’ or ‘configuration’. I expected that ‘failed’ would be more frequent in the escalated comments. Probably when the customers describe their issues more detailed with using the words ‘configuration’ and ‘failed’, the support team understands the problem immediately, and solving the case does not require the engineering team.

TF-IDF

I need a method for adding text features to my feature set. Based on the word cloud, I can calculate the word frequency and count the most common words in the comments. That is not the most effective way because we do not know whether the most common words are important.

TF-IDF is a way to score the importance of words in a document based on how often they appear across multiple documents [12]. If a word appears frequently in a document, it is important. But if a word appears in many documents, it is not a unique identifier. TF-IDF stands for "Term Frequency, Inverse Document Frequency." Where tf is the number of times a word appears in a document divided by the total number of words in the document. Idf is the log of the size of the corpus divided by the number of documents in which the word appears. TF-IDF is tf multiplied by idf.

I used TfidfVectorizer() to find the important words in the case comments. TfidfVectorizer() function is a part of the scikit-learn feature extraction module. We can set a lot of parameters, for example, stop words, the maximum number of features, or vocabulary. First, I selected the function parameters with a maximum of 20 features and the stop words that were extended as described earlier. Then I applied the transformation to the escalated case comments. So I identified the 20 most common and relevant words in the escalated case comments. These words were: attachments, call, check, close, data, diagnostic, error, files, following, information, know, log, new, processed, run, see, server, service, still, and use.

After that, I needed to calculate the TF-IDF for every case and word. I changed the TfidfVectorizer() parameters because the maximum feature number is not required anymore. With the setting called vocabulary, I could set the feature matrix based on the previous 20 words. Finally, I ran the fit_transform function to determine the tf-idf values of every case comment.

4.4 Final feature set

By combining the initial feature set and the result of the survey the final list of features is provided by Table 4.1.

Table 4.1: Final feature set

Basic properties
Daysopen
Number of actions
Component name
CDH version
Perception of process
Number of people in contact on Cloudera side
Number of people in contact on customer side
Increase in priority
Decrease in priority
Perception of time
Time until first contact
Average support response time
Difference between average and expected support response time
Average customer response time
Difference between average and expected customer response time
Customer profile
Number of closed cases
Number of open cases
Number of closed escalations
Number of open escalations
Ratio over number of escalations and all cases
Number of opened cases in the last X months
Number of closed cases in the last X months
Number of opened escalations in the last X months
Number of closed escalations in the last X months
Expected of support response time
Expected of customer response time
Text features - 20 words
attachments, call, check, close, data, diagnostic, error, files, following, information, know, log, new, processed, run, see, server, service, still, use

4.5 Generation of the query

At the beginning of the project, I focused on closed cases. Writing the query that collects every feature into one database was not an easy task. I calculated the various average times and calculated the specific entries to obtain the feature set. I made four SQL views for the feature categories because I was not able to collect them into one. There are a lot of mistakes that we can make while writing a SQL query. A study made by Navita Kumari about query optimization techniques lists the most important of these [6]. Below I would like to outline what I did to optimize my database queries.

I tried to avoid using SELECT COUNT(*) statement because it makes a full table scan, and my table is huge. Instead of COUNT(*), I always specified the column name. I never used the HAVING clause in select statements because I could filter everything with WHERE. Where it was possible, I used EXISTS rather than DISTINCT.

Initially, I had more than one sub query in the main queries. I used several nested select statements, which resulted in slow performance. That often led to timeout.

The machine learning models require a database with intermediate states of the cases. In the beginning, the model was trained on closed cases. But in the real-life, I would like to tell about a 10-day-old case whether it will be escalated. If my model is trained in closed cases, it can not make prediction on open cases. So the next goal was to have a database where each row is one day. To produce these records, I needed to rewrite my queries, but this time, the main goal was also to optimize it. I tried to minimize the number of JOINs. Instead of COUNT something WHERE condition, I used SUM(term) that allows counting based on several conditions. With analytic functions, I was able to write almost everything in one query. These are also known as window functions, and they examine the contents of multiple input rows to compute each output value. Instead of being limited to one result value per GROUP BY group, they operate on windows where the input rows are ordered and grouped using flexible conditions expressed through an OVER() clause. I also used the optional window clause, which makes the function analyze only specific rows "around" the current row. For the intermediate record, I could calculate the moving average with the window clause RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW. In the following code snippet, we can observe a part of the query that can create the necessary data set.

```
WITH comments as(
    SELECT
        createddate as valid_from,
        LEAD(createddate) OVER(PARTITION BY caseid ORDER BY
           createddate) AS valid_to,
        MIN(createddate) OVER(PARTITION BY caseid ORDER BY
           createddate) = createddate AS first_set,
        SUM(cust_response) over (partition by caseid order
           by createddate rows between unbounded preceding
           and current row) AS sum_of_cust_resp_time,
        COUNT(cust_response) over (partition by caseid order
            by createddate rows between unbounded preceding
           and current row) AS count_of_cust_resp_time
        ...
        FROM table
        JOIN (SELECT
            IF(customer,(LEAD(createddate) over(partition by
                caseid order by createddate) -
               casecreateddate)/60, NULL) AS cust_response
               ...)
    ),
    history as(...),
    escalations as(...)

SELECT
    day,
    max(sum_of_cust_resp_time)/max(count_of_cust_resp_time)
       as avg_of_cust_resp_time
    ...
FROM
    days
    LEFT OUTER JOIN comments on day BETWEEN
        IF(first_set, casecreateddate, valid_from) AND
        NVL(valid_to, NVL(closeddate, to_utc_timestamp(now()
           )))
    LEFT OUTER JOIN ...
GROUP BY day,caseid
```

Chapter 5

Data Analysis

Based on the feature set I collect every possible relevant data together into SQL views and in the end, I joined them together to form a pandas DataFrame.

In the data analysis phase we have two goals. First, we clean the data to remove everything that leads the model to wrong predictions. Then, we need to check if these features are relevant, thus can we find correlation between the features and the prediction target.

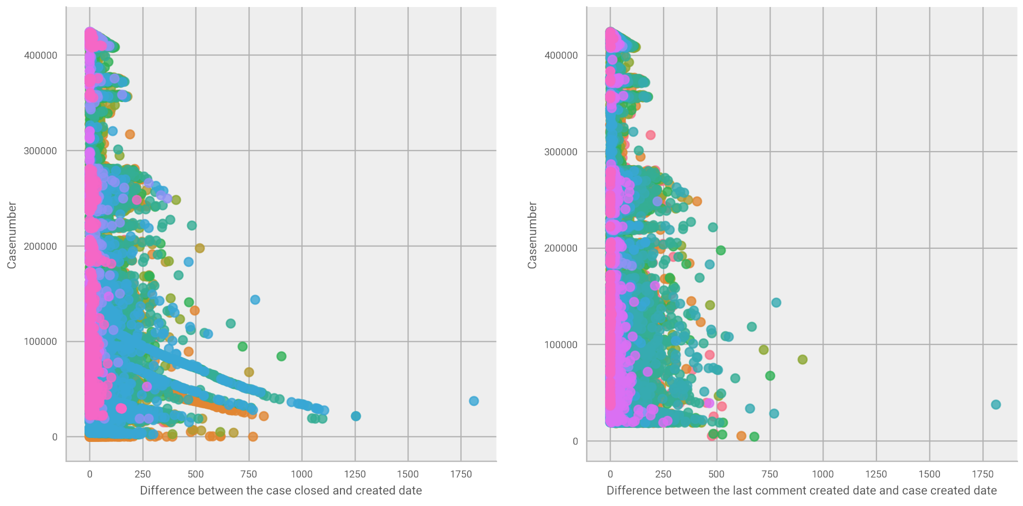
The prediction target is a binary variable, indicating whether the case will be escalated, or a non-binary variable, quantifying how many days the case will be open.

5.1 Data cleaning

During the Data Analysis, we made a lot of validation and data cleaning. For example, a member of the support team suggested ignoring every case where the case number is below 50 000 because these cases came from an old system and the support team worked at that time according to a very different operation model.

As for the customer profile, we only investigate the cases which were created by the customer and not by Cloudera. (Cloudera can also create cases proactively to prevent possible issues, but since these cases have a different life cycle, we ignore them at the moment.)

The strangest phenomenon I found in the data was the number of days a case is open because there are a lot of cases that have not been closed for years. I visualized all of the cases based on the days open. In Figure 5.1a, each point is a case and we can see how long the case was open. On this chart, there are three lines that look very unnatural. For finding the reason of these lines I randomly selected cases on that line and visualized all of their comments during that time (see in Figure 5.2a). If we look at this Figure, there are big gaps between some comments for each case. I have discovered that these comments are often made by a Cloudera bot, that notifies the participants about changes and takes care of the forgotten cases.



(a) Days open (b) Valid days open

Figure 5.1: The number of days open without and with validation

If we filter out the email address of the bots then we get Figure 5.2b, where it is easy to see that a lot of comments disappeared. After this filtering, the unnatural lines in Figure 5.1b also disappeared. That means we found the source of the phenomenon and now it represents the information that we wanted to extract from it.

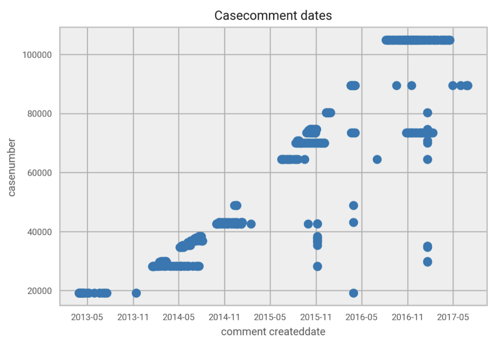
If we are not sure that this method is working, then we can look at Figure 5.3. In the chart, the vertical axis represents every case. The horizontal axis shows the difference (in days) between the last comment created date and the case closed date. If this result is zero, then the last comment was made on the closing day. We can notice that the lines appear in the Figure; this means our validation is correct.

5.2 Correlation analysis

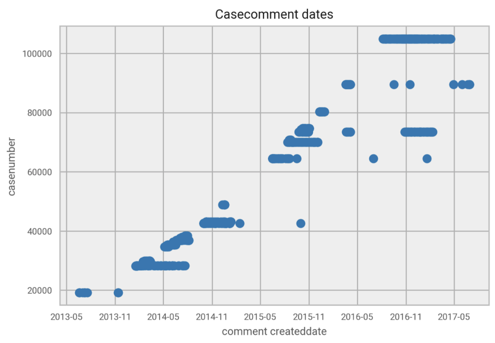
For the ML model, there should be a correlation between the features and the target variable. I intend to make two models; one is a classification that answers if the case will be escalated. The other one is a regression that will predict how many days the case will be open.

5.2.1 Classification

For the classification model, I visualized if there is a correlation between the features and the outcome of the case. The target variable is whether a case will be escalated.



(a) All comments



(b) Comments without the bots

Figure 5.2: The difference in the comments

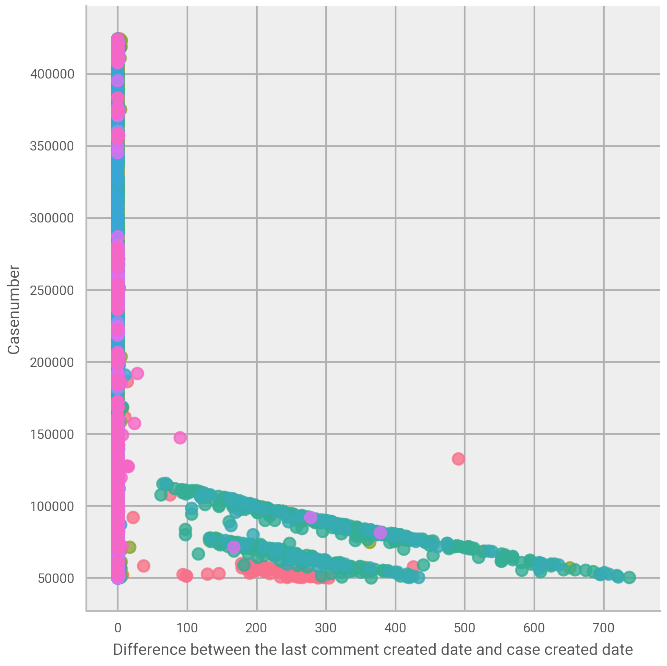


Figure 5.3: Difference between last comment date and case closed date

Among all the plots I include only those, where the correlation is clearly visible.

In Figure 5.4 we can see normalized histogram of the number of status changes for escalated and non-escalated cases, both on linear, and in logarithmic scale, to make the body and the tail of the distribution also visible. The Figures show that there are more status changes in the escalated cases (green bars) than in the non-escalated cases (purple bars). We suspect that this feature will be useful for the classification task.

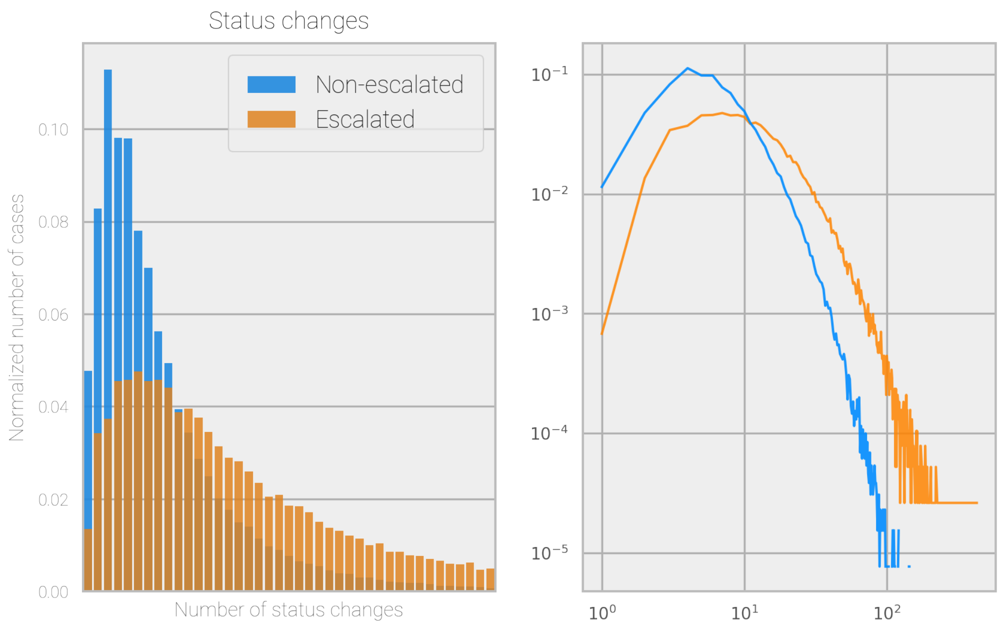


Figure 5.4: Number of status changes - correlation with escalated and non-escalated cases

The analysis of the number of days the case is open (Figure 5.5) leads to the same conclusion.

5.2.2 Regression

For the regression model, I also visualized the correlation between the features and the target variable, which is, in this case, how long a case will be open. When investigating a relationship between two variables, the first step is to show the data values graphically on a scatter diagram. On the chart, the closer the points lie to a straight line, the stronger the linear relationship is between the two variables. I selected two features: number of actions and average support response time. In Figure 5.6 is the number of actions feature, in Figure 5.7 is the average support response time. We can conclude that these features, while there is a significant correlation, are not in a linear relationship with the target variable.

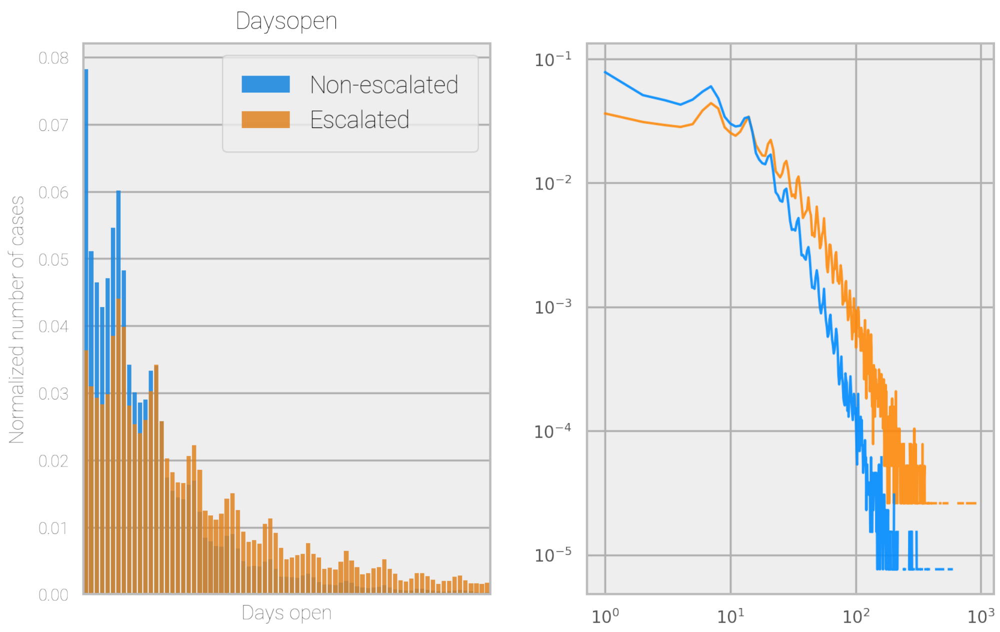


Figure 5.5: Number of days open - correlation with escalated and non-escalated cases

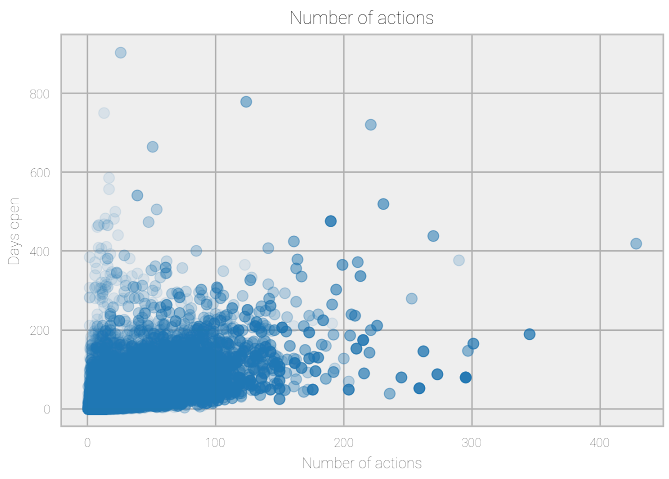


Figure 5.6: Number of actions - correlation with number of days open

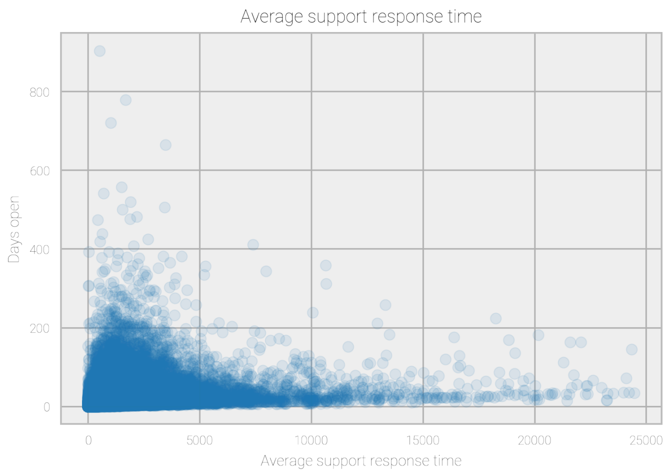


Figure 5.7: Average support response time - correlation with number of days open

Chapter 6

Creation of Machine Learning models

6.1 Model selection

In this section, I would like to present the machine learning models that I have examined for the project. To achieve the two main goals, I need classification and regression algorithms. I used only decision tree-based algorithms for both models. I did not try out Neural Networks because my data set is not large enough for that. A decision tree is a decision support system that uses a tree-like graph decision and their possible after-effects, including random event results, resource costs, and a utility. Take the decision tree as a logical function. The input to the function is the subject or all attributes of the situation, and the output is a 'yes' or 'no' decision value. In Figure 6.1 we can observe the general structure. In the decision tree, each tree node corresponds to a property test, each leaf node corresponds to a logical value, and each branch corresponds to a possible value of the test attribute [2].

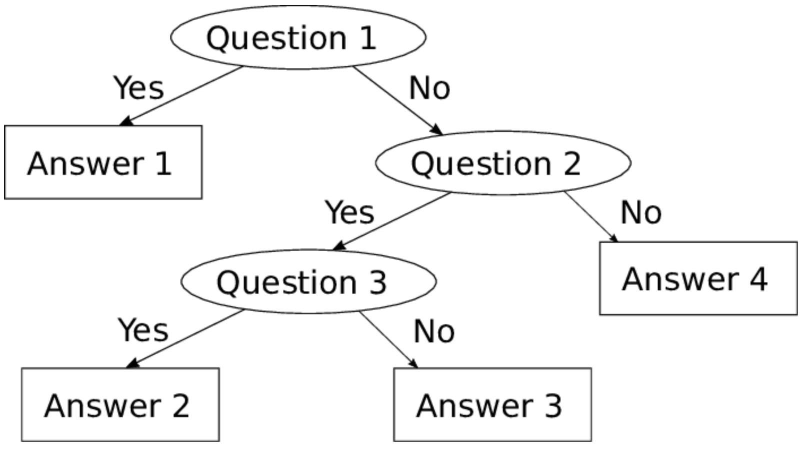


Figure 6.1: A general decision tree structure

6.1.1 Classification

For the classification, I selected Random Forest Classifier and Gradient Boosting decision tree algorithm.

Random Forest Classifier

Random Forest is an easy-to-use, flexible, simple Machine Learning algorithm that gives often successful prediction [3]. The formula was developed by Leo Breiman. As the base components are tree-structured predictors, and since each of these is constructed using an injection of randomness, the method is called 'random forest'. The algorithm builds multiple decision trees and then aggregates the votes from the decision trees to get the final vote. We can observe the structure of the algorithm in the Figure 6.2.

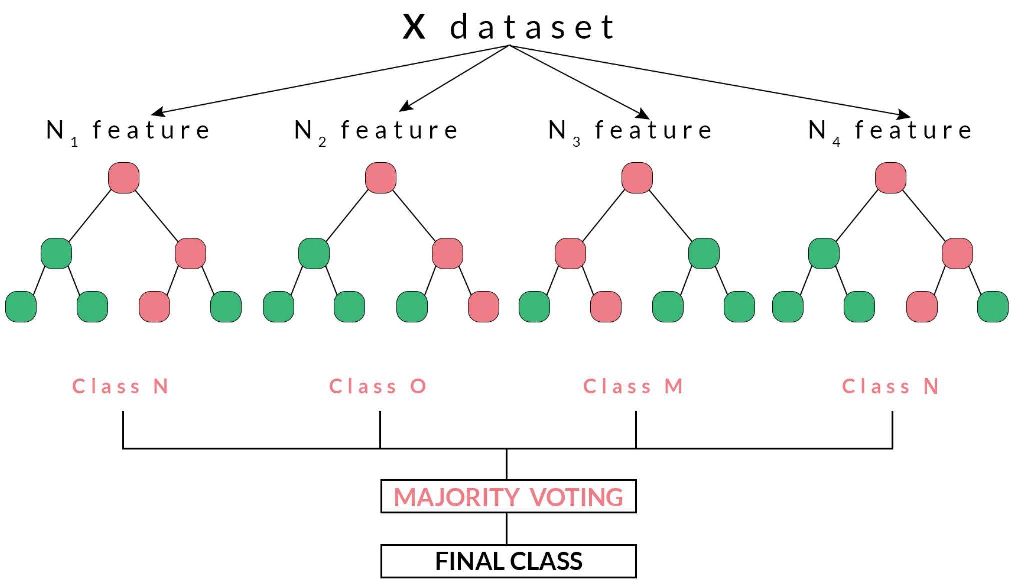


Figure 6.2: Random Forest algorithm

The algorithm steps are the follows:

1. Draw n bootstrap samples from the original data.
2. For each of the bootstrap samples, grow a classification tree, with the following modification: at each node, rather than choosing the best split among all predictors, randomly sample m of the predictors and choose the best separation from among those variables.
3. Predict new data by aggregating the predictions of n trees majority votes for classification [8].

We can optimize the performance of the model with hyperparameter tuning, which means that we can set parameters before training. Scikit-learn implements sensible default hyperparameters, but these are not always optimal for the problem. To determine the best collection of hyperparameters is a complex, time consuming task. The most important settings are the following:

- n_estimators, the number of trees in the forest (ideally, the more trees there are, the better the performance is);
- max_depth, maximum depth of the trees (max. number of levels in each decision tree);
- max_features, maximum number of features considered for splitting a node
- random_state is the seed used by the random number generator.

Gradient Boosting decision tree

The other model I examined was the Gradient Boosting algorithm. That is a widely-used machine learning algorithm, due to its efficiency, accuracy, and interpretability.

The random forests rely on simple averaging of models, which is called bagging. The boosting methods are based on a different strategy. The main idea is to add new models to the ensemble sequentially. At each particular iteration, a new weak, base-learner model is trained concerning the previous error of the whole group [13]. If the error function is the classic squared-error loss, the learning procedure would result in consecutive error-fitting. The researcher can choice the loss function.

The algorithm steps are the follows:

1. Calculate the predicted probability distribution based on the current model
2. Calculate the difference between the true probability distribution and the predicted probability distribution. (The goal is to minimize the total loss and for each data point, we wish the predicted probability distribution to match the true probability distribution as closely as possible)
3. Construct a decision tree
4. Predict the target label using all of the trees within the ensemble
5. Repeat until the number of iterations matches the number specified by the hyperparameter (i.e. number of estimators)
6. Once trained, use all of the trees in the ensemble to make a final prediction as to the value of the target variable [7].

To optimize the Gradient Boosting algorithm, we could also apply hyperparameter tuning. We try to choose the parameters to avoid overfitting. The following listing shows the most important parameters of Gradient Boosting:

- learning_rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators.
- n_estimators, the number of trees in the forest;
- max_depth, maximum depth of the tree (max. number of levels in each decision tree);
- max_features, maximum number of features considered for splitting a node.

6.1.2 Regression

Since the classification and the regression are not so different in the decision tree algorithm, I used Random Forest for the regression as well. This selection can handle thousands of input variables without variable deletion. It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data is missing. Gradient Boosting is also available to create regression models.

6.2 Metrics

We need metrics to compare different models. In this section, I would like to introduce the methods for evaluating the models.

I calculated the accuracy, precision, recall, and F1 score, which can all be expressed from the number of true positive (tp), false positive (fp), total number of positive (p) results, and the number of true negative (tn), false negative (fn) and the total number of negative (n) results.

Accuracy is defined by

$$\frac{tp + tn}{p + n},$$

precision is given by

$$\frac{tp}{tp + fp},$$

recall is expressed by

$$\frac{tp}{tp + fn}$$

and the F1 is

$$\frac{2 \cdot tp}{2 \cdot tp + fp + fn}$$

[4].

Another technique is the precision-recall curve, which shows us the possible combinations of the precision and recall (the precision – recall trade-off), which can be used to check the recall corresponding to a given precision.

The ROC curve (receiver operating characteristic curve) is a graphical method for showing a ratio between the true positives and the false negative. The Area Under the ROC Curve (AUC) provides another way for evaluating which model is better [5]. If the model is perfect, then the AUC is one, and ROC curve jumps to one right at zero, while a bad performing model (a no-skill predictor) leads to a diagonal ROC.

For the regression model, there are some other metrics to evaluate the right model. Loss functions like mean square error and mean absolute error measure how good a model can predict the outcome. Mean square error (MSE) is the sum of squared distances between our target variable and predicted values. Mean absolute error (MAE) is the same measurement but with the sum of absolute differences between the variables [14]. The root mean square error (RMSE) is the square root of the MSE.

6.3 Results for the Classification Model

I used scikit-learn to split the database into 70% training and 30% test set. The final database contained more than one million rows, which means that the test set was 300 thousand rows long. I selected the model that is implemented in scikit-learn, and trained it in CDSW. I built the model in three times: create a prediction for all cases, for the customer and management escalated cases and one for the only engineer escalated cases.

6.3.1 Random Forest Classifier

The classifier had 50 different decision trees in the algorithm. After training the model with the training set, I validated it with the test set.

Having the metrics calculated, I visualized the precision-recall curve in Figure 6.3, which shows us the possible combinations of the precision and recall (the precision – recall trade-off), which can be used to check the recall corresponding to a given precision. The perfect precision-recall curve shows two straight lines from the top left corner to the top right corner and down. The results point out that there is not a big difference in these predictions from this point of view.

In Figure 6.4 are our ROC curves, which are great because they are bending towards the upper left corner. We can notice that the AUC in the engineer escalated

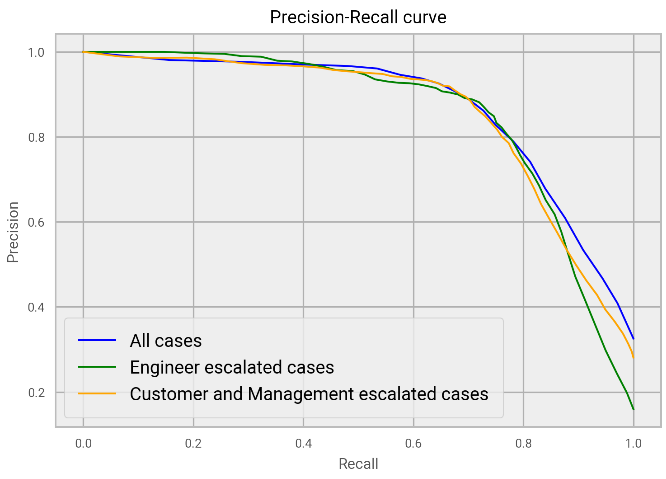


Figure 6.3: Precision - recall curve - Random Forest Classifier

cases has a higher value, which means that the ratio between the true positives and the false negatives is better.

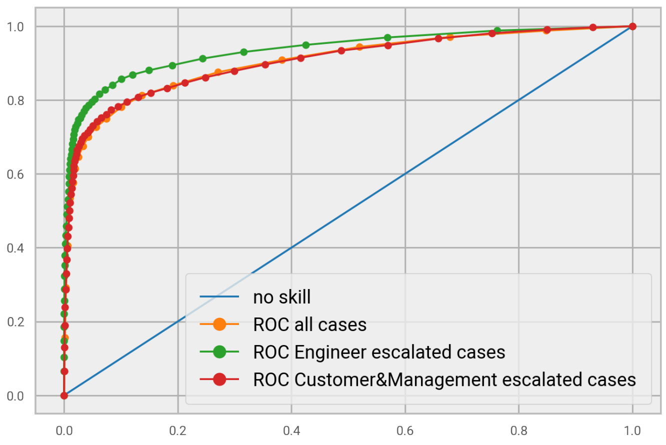


Figure 6.4: ROC curve - Random Forest Classifier

With the Random Forest algorithm, it is easy to measure the feature importance and we can quickly select which feature should be dropped. We can see the 15 most important features in Figure 6.5. Interestingly, there are no textual features in the top 15 significant variables. That shows us that the most important feature is the time until the first comment was made from Cloudera side. If we look back at the Figure 4.1, we can see that the support team felt the same way that it is very important. They also felt that the decrease in priority and the CDH version do not have such a significant impact, which can be confirmed, too. In the feature importance, there is no considerable divergence except in days while the case is open. In the model, which predicts whether an issue will be escalated to the engineering team, the days are almost the most important feature in contrast to the customer/management escalated cases where it is the ninth in the influential line.

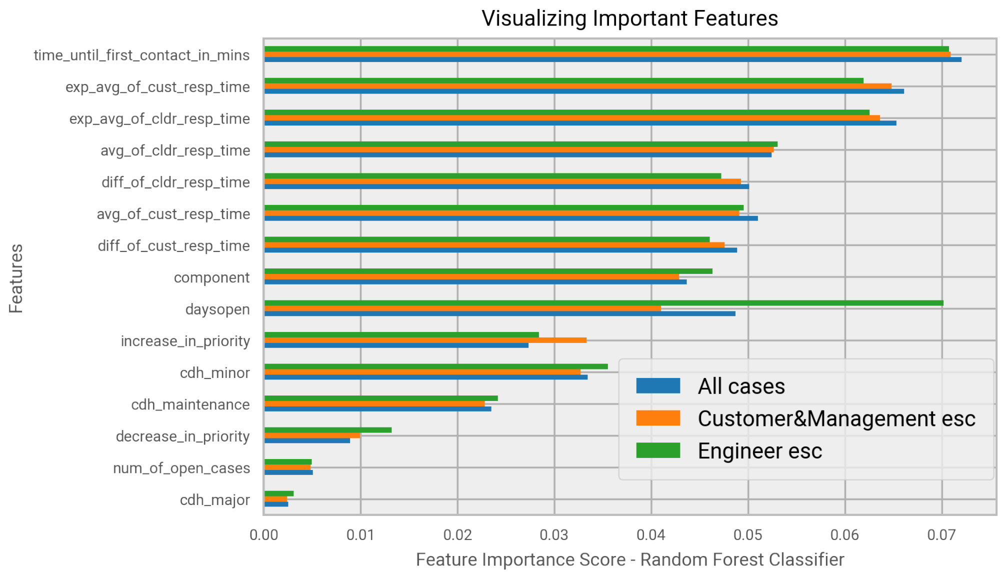


Figure 6.5: The 15 most important features - Random Forest Classifier

6.3.2 Gradient Boosting decision tree

The Gradient Boosting Classifier had 100 trees, maximum 3 features considered for splitting a node, and the maximum depth of the tree was 3 levels. I have examined more options for the learning_rate. The chosen learning_rate was 0.75. After splitting the training and test set, I ran the algorithm. For the results, I visualized the feature importance, the precision-recall curve, and the ROC curve.

On the precision-recall curve in Figure 6.6, there is a slight dip at the start of the green curve but it can be normal. Precision is looking at all the examples that the model flags positively, and of those the truly positives. When reducing the threshold, all newly appearing positives are false, which decreases the precision.

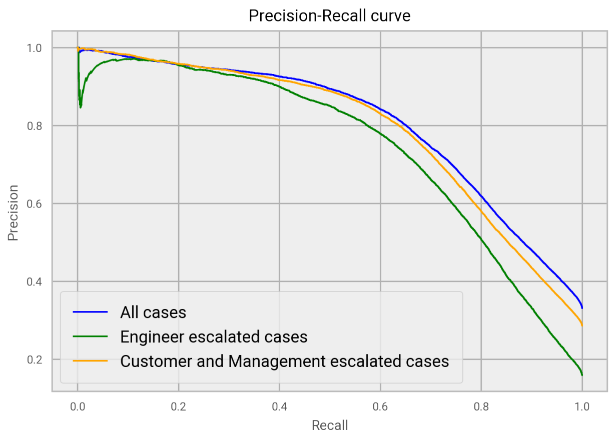


Figure 6.6: Precision - recall curve - Gradient Boosting Classifier

In Figure 6.7, there are the Gradient Boosting Classifier's ROC curves. The three ROC lines shapes are almost the same, but the smallest AUC belongs to the engineer escalated cases. The lines are bending towards the corner but not to the same extent as with the Random Forest Classifier. If the two models were compared based on this metric, the better model would be the Random Forest Classifier.

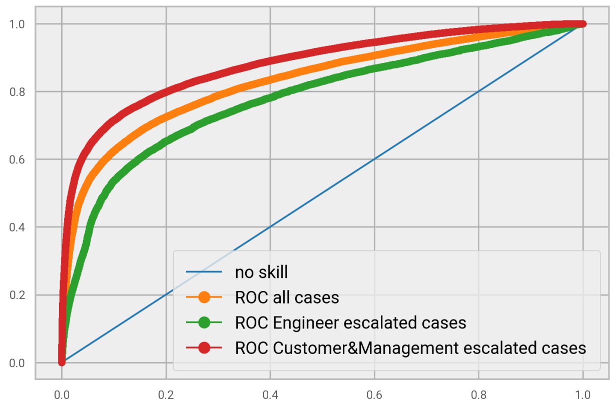


Figure 6.7: ROC curve - Gradient Boosting Classifier

We can observe the feature importance in Figure 6.8. The number of escalated cases and the days open have much more influence than the other features.

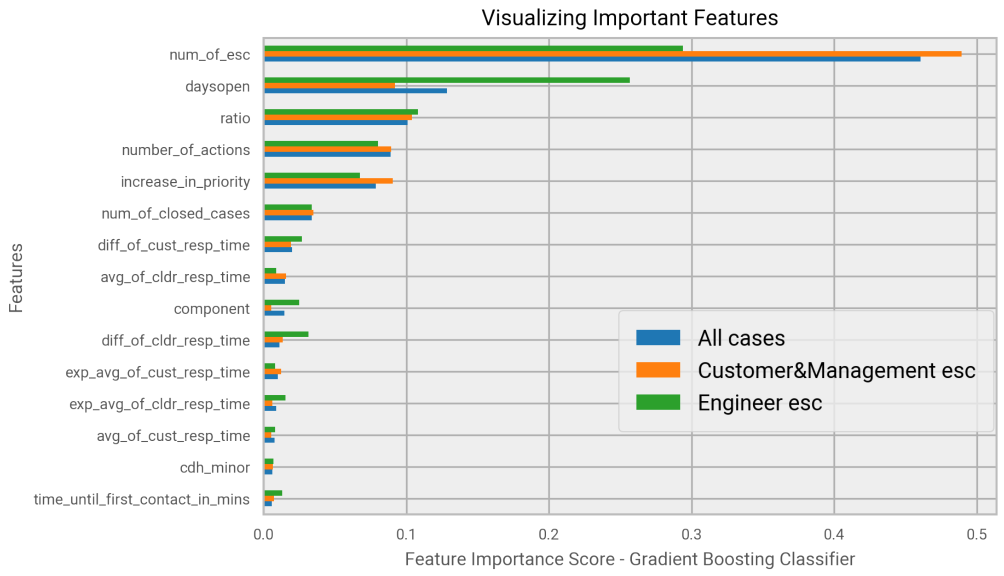


Figure 6.8: The 15 most important features - Gradient Boosting Classifier

To select the better model from these two classification algorithms, I compared all charts and evaluating metrics. The Random Forest Classifier results in a more flexible, successful model. The model could be extended with features and rows; it is scalable, which is extremely important in business situations.

6.4 Results for the Regression Model

For the regression model, I ran a Random Forest Regressor with 100 trees. In Figure 6.9, there are all cases from the test set. On the horizontal axes are the actual values, and on the vertical axes are the predicted values.

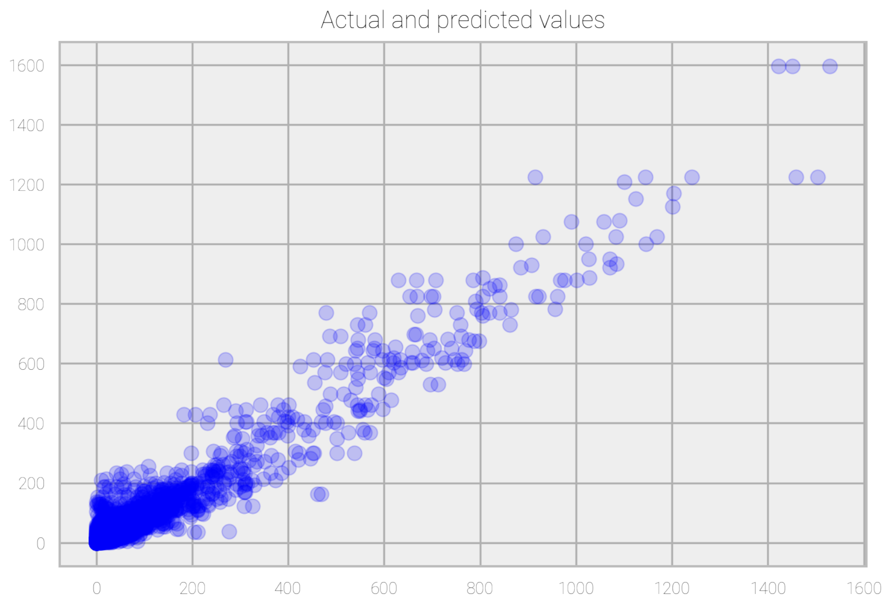


Figure 6.9: Actual and predicted values - Random Forest Regressor

About the calculated metrics, the Table 6.1 shows the mean square error, the mean absolute error, and the root mean square error. These measurements means that the error between the actual data and the predicted data is around 20 days. These errors are also visible in Figure 6.9.

Table 6.1: Loss functions for Random Forest Regressor

	MAE	MSE	RMSE
All cases	11.636	642.012	25.338
Customer and Management escalated cases	10.5036	505.062	22.474
Engineer escalated cases	10.726	571.566	23.907

Figure 6.10 depicts the feature importance for the Random Forest Regressor. The average customer response time is the most important variable. This feature was not originally part of the feature set, but after analyzing the survey results, I added it to the ensemble. The same situation is with the expected average customer response time; it was also added later which turned out to be a good decision.

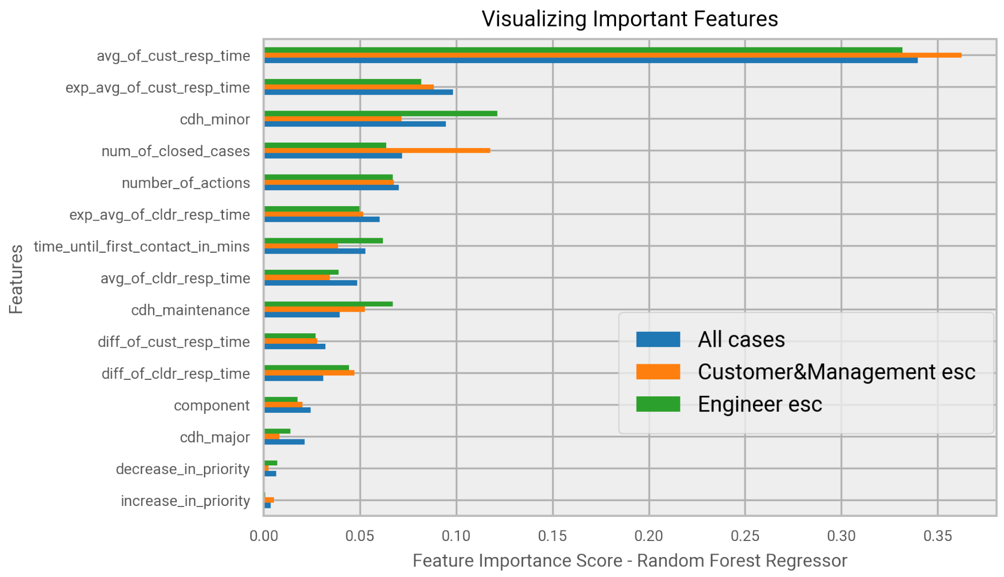


Figure 6.10: The 15 most important features - Random Forest Regressor

Chapter 7

Conclusion

In my research, I presented a procedure based on data processing, analysis and machine learning. I was looking for an approach to two different problems. One classification and one regression problem. I have a predictive model to decide whether the case will be escalated and an other model to predict how long a case will be open. For these models, I had to go through all the machine learning steps.

I collected every important feature for my machine learning model and after validation and analysis, I created a clean and accurate data set which is essential for the robust prediction. I experienced that query optimization and scalability are time-consuming, but for the future application opportunities, they are worth the work. I learned that data could be dirty but if we would like to have a successful model, then it is a necessary step to clean it. I validated every assumption with the support team, I took their opinion into consideration and thereby expanded the feature set. They had really accurate feelings about the feature set but everything that they suggested was proven by the results of the models.

For the machine learning problems, I tried out Random Forest Classifier and Regressor, and Gradient Boosting Classifier. These algorithms are based on decision trees, flexible algorithms with many hyperparameter tuning opportunities. In my case, the Random Forest was better than the Gradient Boosting, with it I have good predictive models. These predictions offer solutions to business situations as well, because the companies could save human and financial resources with using it. My models could make a big impact on the company's processes about the cases.

I proved that these concepts are feasible, we can make correct predictions with the two models. The models are not perfect, but with more data, feature and hyperparameter tuning we could have more precise predictions.

Bibliography

[1] About Cloudera, Inc. *https://www.cloudera.com/about/enterprise-data-cloud.html. Accessed: 2019-09-23.*

[2] Qing-Yun Dai, Chun-ping Zhang, and Hao Wu. Research of decision tree classification algorithm in data mining. International Journal of Database Theory and Application*, 9(5):1–8, 2016.*

[3] Niklas Donges. The random forest algorithm (2018). URL https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd*, 2018.*

[4] Peter Flach and Meelis Kull. Precision-recall-gain curves: Pr analysis done right. In Advances in neural information processing systems*, pages 838–846, 2015.*

[5] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. Radiology*, 143(1):29–36, 1982.*

[6] Navita Kumari. Sql server query optimization techniques—tips for writing efficient and faster queries. International Journal of Scientific and Research Publications*, 2(6):1–4, 2012.*

[7] Cheng Li. A gentle introduction to gradient boosting. URL: http://www. ccs. neu. edu/home/vip/teach/MLcourse/4_ boosting/slides/gradient_boosting. pdf*, 2016.*

[8] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. R news*, 2(3):18–22, 2002.*

[9] Wes McKinney. Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. " O'Reilly Media, Inc.", 2012.*

[10] Rohit Menon. Cloudera administration handbook*. Packt Publishing Ltd, 2014.*

[11] *Lloyd Robert Frank Montgomery. Escalation Prediction using Feature Engineering: Addressing Support Ticket Escalations within IBM's Ecosystem. Master's thesis, University of Victoria, Victoria, 2017.*

[12] *Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In* Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. Piscataway, NJ, 2003.*

[13] *Robert E Schapire. The boosting approach to machine learning: An overview. In* Nonlinear estimation and classification*, pages 149–171. Springer, 2003.*

[14] *Mark R Segal. Machine learning benchmarks and random forest regression. 2004.*