In this document I will try to explain my process and thoughts and my time management.

- Understand the case

I downloaded and decompressed the attached files, read the task and understand it well, and open the data files to take a quick look at the data and get a basic idea of what they are like.

15min

- Labeler (1)

I understand that at some point I will have to classify the cases, so I start preparing a basic-level Python script with Tkinter that displays the text on the screen and has 3 buttons to decide the answer to the question: "Is the customer asking for the status of his order?" One button to say yes, one to say no, and one to say that I'm not sure and will look into it later. The script will finally save the given classification.

45min

- Data analysis

I decide to go with Google Colab because I'm quite used to it and I think it's good for commenting on the text and separating it into sections that you can later hide. Also, you can have the data uploaded to GitHub and not depend on the machine installation or anything else. This helps with replicability.

I start creating the Google Colab, load the data into pandas, and check how many messages I have and how many fields of information. My first idea, taking into account that I'm more used to dealing with texts and Deep Learning, is that I will probably have to train a classification model with the text as input. In this case, 1433 messages may be not enough to train the model.

Just by looking at them visually, I realize that there are some fields that seem to always have the same value. I check it in the code and decide to remove all the fields that only have one value since they don't give any information about the type of message it is. So, I'm left with the fields of "Description", "tags", and "CreatedAt". Considering that the latter is self-descriptive, it does not provide any information about the message content, so I also remove it. I think I can use the words from the text and the tags as the input for the model.

20min

- Split tags

However, the tags consist of an array of variable length. Therefore, I decided to have all the arrays of uniform length, keeping the maximum length that we have, in this case 5, and filling the arrays of shorter length with None values. After that, I separate each tag in the array into a different column of the Dataframe while maintaining the order in case it is important for classification.

30min

- Clean text

Initially thought that training a language model, probably with a neural network, a transformer, would be the way to go. So firstly I dedicated myself to cleaning the text of capital letters, punctuation, numbers, and strange symbols. I tokenized the text and lemmatized the words. After some bug fixing, I finally analyzed the length of the messages to normalize them to a length of 370 by adding NaN elements.

1:30h

- Investigation

Cleaning the input text is a very important part in the case of the transformer, but even so, I suppose that the number of messages will not be enough, so I investigate a bit about data augmentation in this case. During this, I also realize that spelling is a very important point. A misspelt word can affect tokenization and subsequent identification of the word. So I dedicate a significant amount of time to researching possibilities for a spelling corrector and which would be the best option. After investigating and reading the paper https://aclanthology.org/2020.lrec-1.228.pdf, I decide on LanguageTools because it seems to be able to deal with both cases of errors for joined and concatenated words and seems easy to use. I think the best way would be, once I have the dictionary of word occurrences in my training text if a word does not appear in it, to first pass it through the corrector and see if it detects it as an error, and if so, correct it, otherwise mark it as unknown.

1:15h

- Train and test data

Before starting to use the spell checker, in order to change things up a bit, I decided to split the set into training and testing. I opted for a distribution of 75%/25% and selected the sentences randomly since I am not sure if there is any internal distribution that I am unaware of and I want to avoid it.

10m

- Onehot encoding

I want to convert the tags to one-hot encoding in order to normalize the distance between them. I'm thinking about how to combine them since passing them to the input consecutively could result in a significant increase in the length of the input. So I think it would be best to sum them up instead of concatenating them one after the other. With this way it will reduce significantly the length, but we will lose the information on the order, but I checked previously that as the examples we have, we always receive the tags in the same order, so probably there is some sorting previously to us receiving it.

I also decide that tags that appear only once will be combined into the Unknown value, so they will not be considered separately in training, as they are too specific and can affect a worse result and generalization.

Here there is an important change as I realize that instead of applying a complicated text classification process, I can try to use the tags as input for classification and make it work correctly. From now on I will focus on this way on solving the problem.

1:30h

- Sampling

In order to train classification models, I decided to randomly reduce the number of samples to half in order to reduce the labelling time.

5min

- Labeling (2)

I just fixed the Tkinter labelling program to adapt to how I currently have the data stored, and I started labelling them.

In addition, in case it turns out that I need to classify according to the text again, this will give me visual information about typical cases that I could clean up from the text.

I notice that there are some cases where the classification is a bit ambiguous, but I try to do my best.

2:15h

- Train and eval data

I separate the data into training and evaluation sets in an 80/20 ratio, in order to evaluate the next models. This leaves a final distribution of 60% for training, 15% for evaluation, and 25% for testing.

5m

- The models

Here, I considered the different model possibilities that I have to train, in order to test how well they perform and choose the best one in the end. I decided to train an SVM, a Random Forest, and a Neural Network. Intuitively, I think that the SVM will work better since it usually requires fewer data than other models to obtain good results, and the classes are well-defined. I also think that the Neural Network probably will not perform very well since we have little data, especially after sampling.

While preparing the SVM, I think that since I have a limited amount of training data, I should apply k-folding during training, so I do that. I started preparing the SVM and the Random Forest.

1:10h

- The situation

Here, I take a moment to define the situation clearly and determine the best metric for evaluation.

The situation that I have in mind is that there is a classifier that classifies whether a message is "where is my order" or not, and in case it is not, it will be sent to another classifier (probably a human) that will say where this message needs to go. With this in mind, I am clear that the critical error is when the model classifies a 0 as a 1, as in this case, I assume that following there is an automatic process that can cause problems if the classification is incorrect. In contrast, if a 1 is classified as 0, the human can reclassify it as the correct class and send it to the appropriate process.

Based on this, I define that the most important metric is precision, as it values the number of true positives among those that the model has classified as positive. To avoid cases where the model directly classifies all cases as 0, I will train the models using accuracy as the primary metric, and take it into account as a secondary metric in the evaluation.

20m

- The models(2)

After defining the situation, I decided to set a confidence threshold for the output of the models. If the confidence is not higher than a certain value that I will define, the prediction will be considered invalid and sent to 0 (human classifier). I continue implementing the random forest and the NN, the latter based on the model in https://machinelearningmastery.com/building-a-binary-classification-model-in-pytorch/. Additionally, I apply a specific seed to all models in order to have consistent results if I train them again.

1:40h

- Evaluation data

I use the evaluation data to check that the models are working correctly and to decide which confidence threshold level would be the best. I display the data as a confusion matrix to make it more visually understandable. I realize that all three models obtain the same results at their best threshold. I look at which phrases these are where they fail, in case it might be that they are not labelled correctly according to the text, but even though they are a bit ambiguous, I believe that the cases are well labelled.

25min

- Some minor problems

In order to maintain independence between sections, and to be able to load the models in the future to evaluate the test set, I am trying to download the models and be able to load them again, as well as the same for the labelBinarizer used to make the one-hot encoding for the tags. This results in giving me a little more trouble, and that's why I had to dedicate some time to it.

45min

- Test data

I am dedicating myself to cleaning the test data, in order to have them in the same format as the training data. I pass them through the models and observe the results. In the case of SVM and random forest, they continue to give the same results, but the neural network is now performing worse than the rest. Above all, I notice that we are dealing with a critical case and I examine it. According to the text, it seems to be well labelled, although I noticed that in the labels that appear the "where-is-my-order" which I suppose normally defines positive cases, so I consider it to be an error when they added the tags.

I tried to visualize the model's decisions, but in order to not drag more time the task I decided to not do it. But if I had more time, it could be done using PCA or similar techniques.

45min

- Best model

Finally, I select the best model and decide to go with SVM. Both SVM and Random Forest obtain the same results, so the processing time is a very important factor to consider. SVM takes 0.008s to process the 179 test cases, while RF takes 0.0224s, which is not very significant with these numbers of cases. But if we wanted to apply it in a real situation where many more cases could arrive, this time difference could represent a significant increase.

I decided to not label all the rest of the data, as it will increase the time and I suppose it would probably not affect that much on the results. Although with more time it could be a thing to be tested.

15min

- Extras

As an extra step, I check the code and finish cleaning it. And write this document from the notes I have taken.

2h