Efficient Service Request Categorization with Stacked LSTM Networks

Kosti Koistinen

September 21, 2023

Abstract

In this excercise, a multi-layer LSTM network was built for a classification task. I constructed the original code on summer 2023 but modified further on autumn 2023 for the purposes of this course. The classificator was trained with data owned by *Siun sote* (Pohjois-Karjalan hyvinvointialue). I had permission to use and distribute data as long as it does not contain sensitive data. For security reasons. I shall not distribute the dataset. If you want to test my code with original data, send me email at kokois@utu.fi or kosti.koistinen@siunsote.fi.

This document has 5 sections: Data description, Data transformation, Model description, Results and Discussion. The .ipynb - notebook file is attached separately.

1 Introduction and data description

Siun Sote (Pohjois-Karjala Welfare Area) uses a support service *Meittari*, controlled by *Meita* (Meidän IT ja talous Oy). It handles different service requests carried out by organizations labourers. The requests include, for example, password requests, hardware problems and orders, software license handling and so on. The service requests are registered and classified manually (see Table 1). Classification is slow and services are often flagged to wrong groups. My task originally was to develop a machine learning model that learns to label requests to given categories automatically.

Table 1: A few examples of service requests and their classes

Date	Request	Class
03.04.2023	Mikrofoni ei toimi	Oheislaitteet
03.04.2023	Käyntisyy otsikot eivät ui malleista	Järjestelmäpalvelut
11.04.2023	Verkkotunnuksen salasanan nollaus	Käyttöoikeudet

Over 60 000 requests from years 2022-2023 with 14 labeled categories were chosen for training and testing the model. Three of the categories contained only few hundred requests. Therefore, the categories containing less than 1000 requests were combined to a group *"Muut"* (Other). In total, all the requests fell within 11 groups. List of the categories and requests are shown in Table 2.

Categories are skewed. The biggest group has one third of all requests. Figure 1 shows the distribution of classes. It also contains distribution of test and training sets. They were split 80/20. There are 49 178 rows for training and 12295 for validating (testing) the model. In total there was 61474 rows.

Table 2: Categories with requests

Category	Requests	Category weight	Category Number
Käyttöoikeudet	20183	0.3	0
Järjestelmäpalvelut	14210	0.4	1
Työasemasovellus	7257	0.8	2
Työasemat	6436	0.8	3
Oheislaitteet	2887	1.9	4
Mobiililaite	2644	2.1	5
Muut	1918	2.9	6
Puhelinpalvelut	1815	3.1	7
Etäyhteydet	1752	3.2	8
Elinkaari	1288	4.3	9
Tietoliikenne	1084	5.2	10
Total	61474		

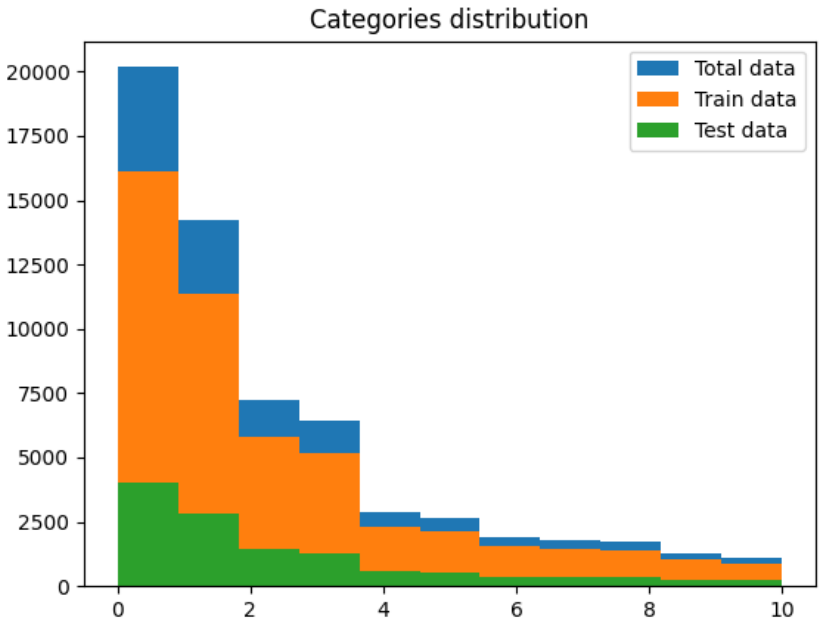

Figure 1: Table 2 categories on x axis with total, training and test split counts. The service requests are not uniformly distributed.

2 Data transformation

First step of the process was the deletion of sensitive information, i.e. cell phone numbers and social sequrity numbers. The data was then lowercased and any serial codes were deleted. The text was then tokenized with *nltk* tokenizer. It gave some statistics of the dataset: Data contained 24419 unique words (vocabulary), with total word count being 185904.

Word2Vector (W2V) was used for vectorizing the words. The vector size, also known as embed-

ding dimension, was chosen to be 300. The value typically is between 50-300. Big vector size requires more computation time. On the other hand, more fine grained details can be aquired from the text. Furthermore, the requests have rather short lengths: a typical length is around 5 words. A large vector size was thus not required. It was nevertheless held at 300 as it did not affect drastically to computational time. For long corpuses, a large vector size is necessity.

The W2V model was then saved and used for embedding matrix. The size 5 was chosen for maximum sequence length, because most of the requests are shorter than 5 words (Figure 2). Maybe this reasoning is faulty, but increasing the size quickly resulted in longer execution times of training and no improvement of the model.

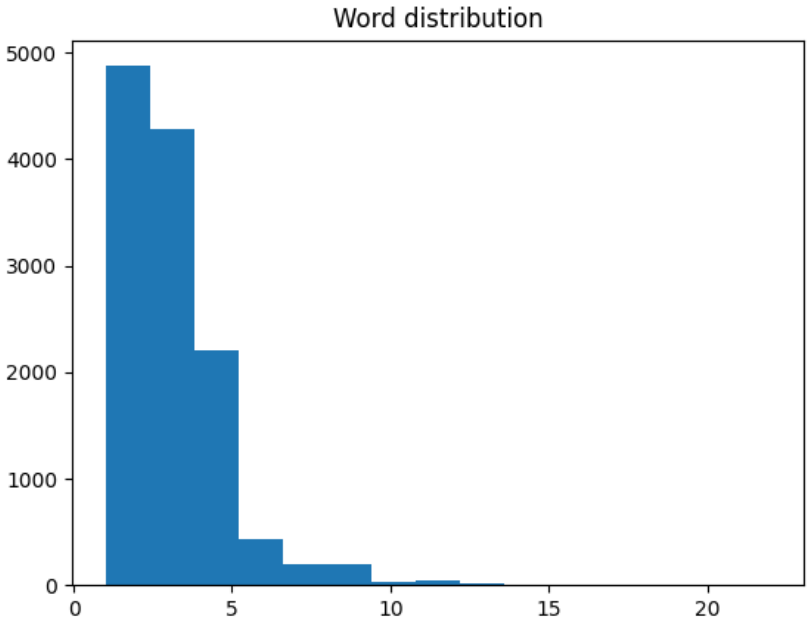

Figure 2: The distribution of request lengths. Typically the requests are very short, between 2–5 words.

Next, the imbalance of categories had to be dealt with. The weights of the categories had to be issued since The largest group has one third of data. The category weights help the model to handle the data more evenly. The weights for each category are in Table 2.

The vocabulary was rather short, and often, the model runs into words that are not in there. This results in crashing of the program. Therefore, a function had to be written for W2V model to ignore any word it does not have in vocabulary yet. A more sophisticated model would apply new words into the vocabulary. Adding was left for a future exercise.

Finally, the vectorized texts had to be padded. Padding is a crucial step for each vector to have same dimension (or uniform length, if you will).

3 Model Description

The code was implemented using *Tensorflow Keras* library and its tools. An LSTM network was chosen because of its efficiency and low computational costs.

3.1 The original model

A lot of different models were tested during the summer. For original purpose, an embedding layer, followed by 10 unit LSTM with a dense output layer with 7 groups were used. This model resulted in quite good performance, achieving a 71% accuracy (Figure 3). The model was very simple, but still worked very fine for decreased dataset having only 7 categories. The problem was that the models performance decreased a lot when the aimed 11 categories were chosen. Wider network did not solve the issue, and a lot of variations of hyperparameters was tested. Therefore, a more complex model was required for the task.

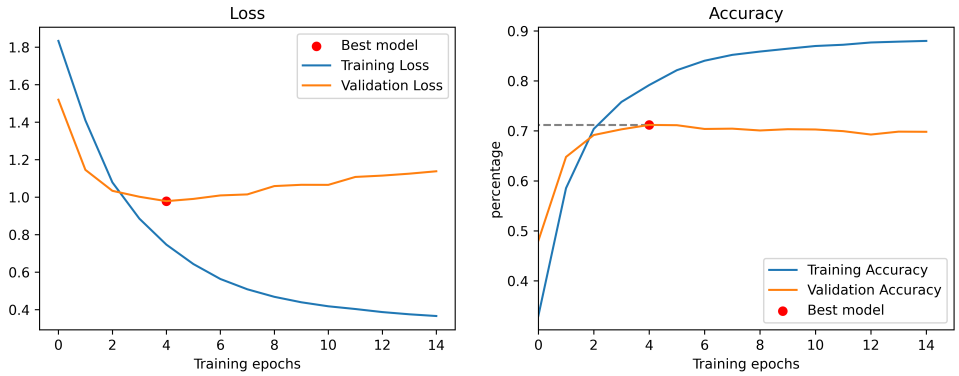

Figure 3: Old model loss (left) and performance (right) by training epochs. The model reaches the smallest loss in 4 epochs and then start to overfit. A 71% accuracy is obtained. Unfortunately, some of the detailed results were misplaced while new model was designed.

3.2 A New Model

A Multi-layer perceptron was definetly needed. The goal was to make the model deal with 11 categories as efficiently as the single LSTM - layer dealt with 7 categories. The model was constructed as follows:

1. The data was ingested by an embedding layer with input of vocabulary size and output of vector size. The weights were assinged by W2V model.

2. A bidirectional 128 unit LSTM network with dimension of vocabulary length to maximum sequence length. Relu was used as activation function.

3. Another bidirectional, 64 unit LSTM network of same properties.

4. A dropout layer of 64 units that randomly drops 40% of weights assigned. This artificial manoeuvre reduces the overfitting.

5. A single LSTM layer of 22 units that maps the results of previous steps into one dimensional output.

6. An output layer. A dense layer, containing the number of nodes corresponding to number of categories. Activation function "softmax" was used.

For compilation, an Adam optimizer was chosen with default settings. The learning rate was the only parameter tuned, from 0.0001 to 0.001. A more eager learning rate seemed to work best. Categorical cross entropy was assigned for loss function. It is suitable for non binary categorizing problems. For detailed summary of the model, see Appendix A.

4 Training and results

Finally, with batch size 128, 15 epochs were assigned for model to train. The history and a result of training and testing is shown in Figure 4. A more detailed scoreboard of training and testing is in Appendix B.

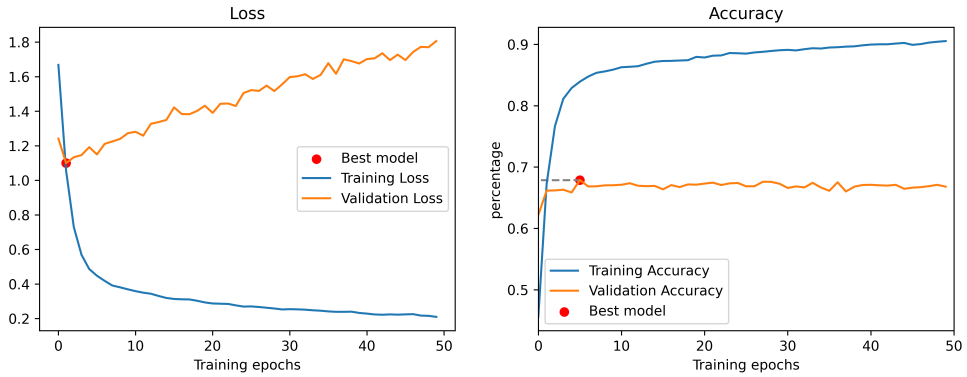

Figure 4: New model. The evolution of training and testing sets loss (left) and accuracy (right). The model quickly learns the characteristics, and then starts to overfit.

5 Results and Reflection, Future Work

Results of performance of the model are in Table 3. In Appendix B, there are more detailed information about the training and testing sets. Group 9 had the best precision with 0.91, while Group 6 had the worst, with 0.29.

Table 3: Accuracy

Group number	Precision	f1 score
0	0.59	0.71
1	0.85	0.68
2	0.79	0.61
3	0.77	0.48
4	0.61	0.55
5	0.81	0.71
6	0.29	0.36
7	0.59	0.65
8	0.41	0.51
9	0.91	0.91
10	0.62	0.48
Accuracy:		0.65

A model was unnecessarely complicated. Complexity resulted to quick overfitting. After trying multiple combinations, a fairly simple model achieved almost similar results. The original model obtained only 52% accuracy, but with adding just a single lstm layer in between the accuracy rose to 65%, which is really close to the best accuracy obtained by the complex model. **Tuning of parameters or model architechture had minimal effect after the optimal complexity**.

The reason why the model starts to overfit and does not improve does not depend solely on the complexity of the model or tuning of hyper parameters. The question of data quality needs to be assessed as well. The confusion matrix reveals the problem of data. A confusion matrix is a a map of false positives and false negatives of the models predictions. In diagonal are True positives. It represents how often the model assignes the correct label. As seen from Figure 5, the largest group "leaks" a lot of false positives and negatives. This behaviour was tried to prevent by assigning the weights for the groups. Unfortunately, it did not work very well. The group 1 reached only 59% accuracy, meaning that nearly half of the results are mislabeled. This has a major effect for model performance as the first group is the largest. Then again, there are some groups that the model regognizes very accurately: Group "Elinkaari" reaches a 91% accuracy. This is remarkable as the group is very small.

The confusion matrix reveals that data is poorly labelled. They are probably not done in uniform manner. Maybe the categorization of requests is not chosen wisely. **The quality of data sets the limits of learning, not the amount of data or complexity of model**. If the data was more uniform, fine tuning of model would probably improve the model. One solution to the problem would be re-categorizing the requests to more uniform groupings, but this would be an enormous, yet immersive task. I am planning to inject these requests to open AI (Chat GPT4) in Microsoft Azure environment to come up with better groups. After that, the model will be compiled again with new categories.

Although with minor effects to results, I have to admit, I was greedy with layers and model complexity. I worked a lot with the Keras packages and had to do a lot of tuning before the multi layer LSTM started to operate. I did not understand a one dimensional mapping layer is needed. On the other hand, I did not understand how bidirectional layer works and that the dimensions need to be adjusted for those layers. This all, in short, led me to experiment with over complicated models when I finally got them working.

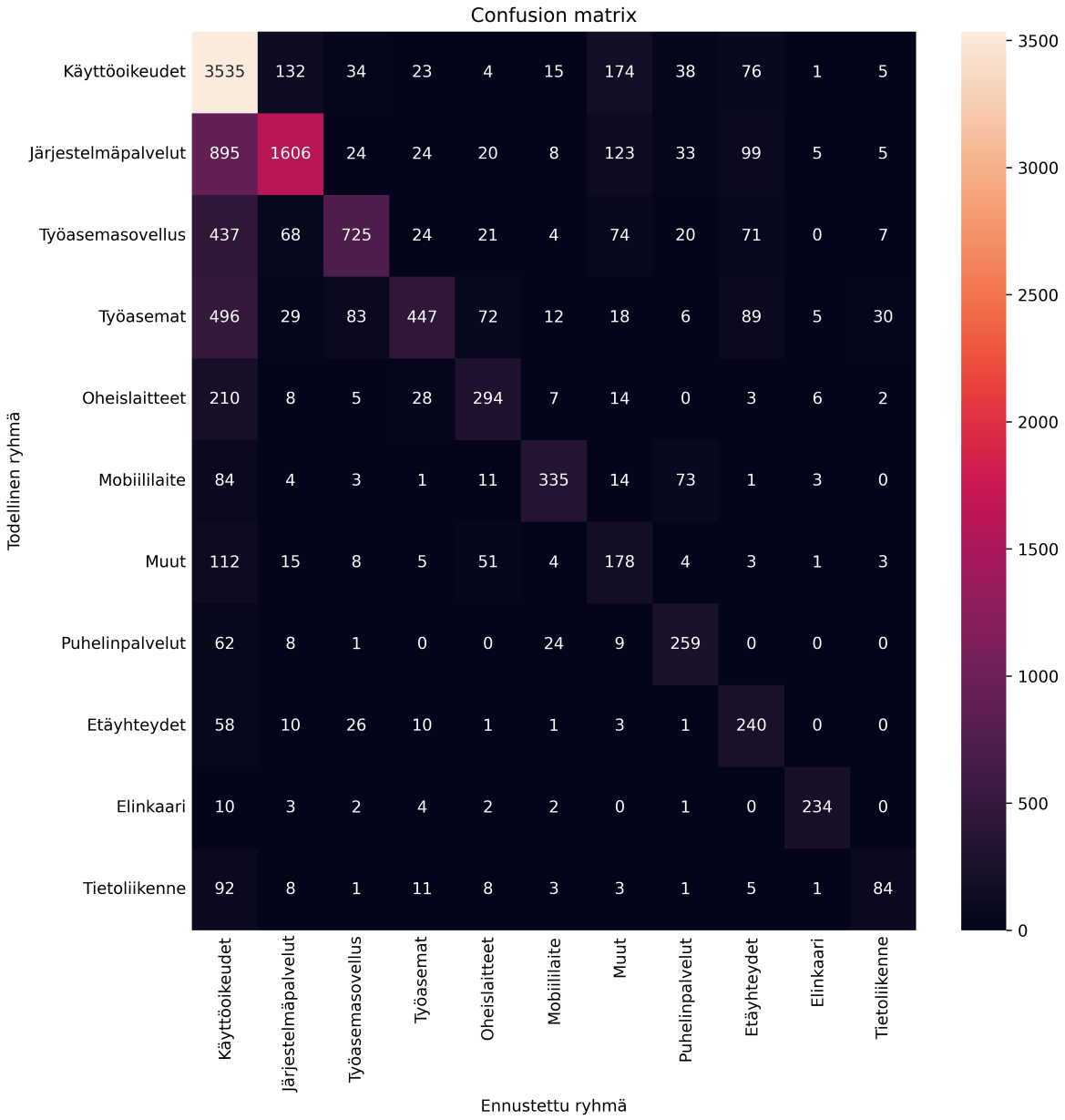

Figure 5: Confusion matrix. On y-axis, the true group, and predicted group on the x-axis. The color coding emphasises the occurences. Group "Järjestelmäpalvelut" has a leakage.

As a final thought, I was too lazy to carry out statistical testing. A statistical test would be quite easy to conduct by overruling the null hypothesis based on probabilities of selecting groups by random. This time I relied on f1 score. It is a value calculated by following formula:

$$\text{f1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall})}. \tag{1}$$

An f1 score between 0.5-0.7 would indicate moderate performance, and anything above good performance. As shown in Table 3, only groups 3,6 and 10 fall under 0.5, which is considered bad model behaviour. Moderate behaviour, however, suggests there are room for improvement.

Appendix A: Model Summary

```
Model: "sequential_67"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_55 (Embedding)    (None, 5, 300)            5532900

 bidirectional_60 (Bidirect  (None, 5, 256)            439296
 ional)

 dropout_5 (Dropout)         (None, 5, 256)            0

 bidirectional_61 (Bidirect  (None, 5, 128)            164352
 ional)

 lstm_92 (LSTM)              (None, 22)                13288

 dense_50 (Dense)            (None, 11)                253

=================================================================
Total params: 6150089 (23.46 MB)
Trainable params: 6150089 (23.46 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Appendix B: Scoreboard

```
--------------------------------------------------
Training set
              precision    recall  f1-score   support

           0       0.63      0.89      0.74     16146
           1       0.90      0.67      0.77     11368
           2       0.80      0.54      0.65      5806
           3       0.86      0.41      0.55      5149
           4       0.70      0.60      0.65      2310
           5       0.84      0.66      0.74      2115
           6       0.41      0.58      0.48      1534
           7       0.65      0.84      0.73      1451
           8       0.48      0.76      0.59      1402
           9       0.90      0.91      0.91      1030
          10       0.72      0.58      0.64       866

    accuracy                           0.70     49177
   macro avg       0.72      0.68      0.68     49177
weighted avg       0.75      0.70      0.70     49177

--------------------------------------------------
Test set
              precision    recall  f1-score   support

           0       0.59      0.88      0.71      4037
           1       0.85      0.57      0.68      2842
           2       0.79      0.50      0.61      1451
           3       0.77      0.35      0.48      1287
           4       0.61      0.51      0.55       577
           5       0.81      0.63      0.71       529
           6       0.29      0.46      0.36       384
           7       0.59      0.71      0.65       363
           8       0.41      0.69      0.51       350
           9       0.91      0.91      0.91       258
          10       0.62      0.39      0.48       217

    accuracy                           0.65     12295
   macro avg       0.66      0.60      0.60     12295
weighted avg       0.70      0.65      0.64     12295
```