Analysis of Patterns and Similarities in Service Tickets using Natural Language Processing

Dumitru-Tudor Tolciu, Christian Săcărea, and Cristian Matei

***Abstract*—In this paper we propose an approach for classifying documents, embedding documents into feature vectors and using these embeddings for finding similarities between them. Our chosen domain for applying this method is the IT-Service Support branch, where the documents we try to analyse are support tickets and the potential of classifying and finding patterns between tickets is huge for optimizing the service process. We aim to tackle the problem with multiple methods of text classification and recognition, and data analysis, followed by comparison and interpretation of the results. Following our previous work in this field, we propose further means of validating our models, so we can describe and visualize several methods of feature extraction and recognition for service tickets that help the business process for Service Support.**

I. Introduction

Data analysis in an informal context, especially in the Natural Language Processing (NLP) [1] branch, has been proven a difficult subject, in which humans still outperform machines. Our contribution to this domain mainly lies in the method of extracting embedding from ticket documents and using these to analyse our data pool and derive concepts: ticket embeddings allow us to analyse tickets through mathematical methods. Their key advantage is projecting a text from the informal human language into a formal one, that can be manipulated by machines.

In our previous paper [2] we describe the steps we took for developing ticket classifiers along with extracting and analyzing features. Continuing our work, we focused on interpreting our achieved results through different means, and ways we can aid our business domain into making an educated decisions, based on these interpretations.

In the following sections we will discuss the *problems* we tackled in the context of service process optimization, and aspects relevant to the problem domain, our *approach*, our *implementation* of the agent(s) for pattern recognition and feature extraction, the achieved *results* and means of interpretation together with *output enhancements* that better suit the practical domain of our scope.

Manuscript received March 8, 2020; revised December 7, 2020. Date of publication February 24, 2021. Date of current version February 24, 2021. The associate editor prof. Dinko Begušić has been coordinating the review of this manuscript and approved it for publication

Authors are with the Computer Science Department, Faculty of Mathematics and Computer Science at the Babeș-Bolyai University in Cluj-Napoca, Romania (e-mails: tudor.tolciu@gmail.com, csacarea@math.ubbcluj.ro, mateicristiansb@gmail.com).

Digital Object Identifier (DOI): 10.24138/jcomss.v17i1.1024

II. The Service-flow Optimization Problem

As discussed in [2], we want to aid the HelpDesk process through augmenting it with two main automated features: classifying tickets into service categories and finding similar tickets for a new incoming one.

The problems we tried to tackle came into a cascade format, but to reiterate shortly, these were the steps we took towards a solution for optimizing the HelpDesk process:

Firstly we had to classify tickets into *Service Requests* and *Incidents*. A service request is an application someone makes when they need a service, a functioning item or a Helpdesk clearance. This category includes network access, phone and laptop requests, admin rights, and others. Whereas the Incident is a type of ticket that the user usually makes when he experiences some kind of *problem* or *failure*, it is denoted by the fact that something is not working properly. Common records for this type of event are hardware or software failures that occur most of the time. This classification type (Service vs. Incident) is called *Procedure Type*.

Our usecase subsequently developed into a more detailed ticket type hierarchy: *Request Type*. This form of classification has several more label forms (around 120), with random variations in frequency in all groups. So we took the top 50% tickets from the most commonly seen groups and were left with 12 separate courses for our training so to avoid coping with the distorted classes [3]. The following sections include: E-mail and exchange queries, VPN communication, orders or enquiries for hardware and accessories, password resets and much more.

After ticket classification, came another opportunity to optimize the Service process: by finding similarities between tickets. Similarly to the Frequently Asked Questions (FAQ) concept, routine requests or problems appear more often and share similarities in text, thus having the same solutioning process.

However, the linguistic barrier imposes a noticeable difficulty: mails are sent to the HelpDesk team in multiple languages, including English, German and Romanian. Moreover, roughly 95% of the inquiries in our context are made in German. This means that it was imperious for us to find natural language processing tools that can operate on the German corpus and vocabulary. Also, our solution should be able to work for English and Romanian texts as well, making the task even more complicated.

Related work tackles a classical classification problem: spam filtering. For instance, [4] presents a wide range of machine learning methods for e-mail filtering and classification. Other

authors use Support Vector Machines [5] and Artificial Neural Networks (ANN) [6]. The efficiency of Multilayer Perceptrons regarding spam filtering has been studied in [7]. Through yielding noticeable results, a particular weakness of these models is their incapacity to perceive the context: the human language is strongly context sensitive, especially when it comes to order. This key problem is addressed in [8], where the authors also propose a solution for this: Recurrent Neuronal Networks [9]. Numerous other works in the field of Natural Language Processing [10], [11] are using Recurrent Neuronal Networks for text-document classification, since they represents the state of the art in this field due to their ability to process data in a time-dependent form. Since the core of our problem relies on text classification using NLP methods for ticket sorting, we attempted to achieve similar results by using these types of neural networks, as will be described in the following sections.

III. Our Approach

For our usecase we researched a particular branch of tools form the field of Natural Language Processing, capable of capturing features and classifying a text into categories: Word2Vec [12], and GloVe [13] word representations. Another representation of words in a text we used was the Bag Of Words Model (BoW) [14] that basically represents a word counter vector of all the words of a text in relation to the vocabulary of the corpus used. BoW was used as well as an independent encoder, as in conjuncture with Word2Vec embeddings.

As described in [2], the above mentioned methods were used for coding the input of our neural networks in two ways: a BoW vectorial input, which simply represents an one-dimensional tensor the size of the vocabulary, and sequences of indexes, in which each word is represented by it's index in the corpus vocabulary. A sequence is a vector of indexes that represent words of the vocabulary. The advantage is, these indexes appear in the order of the words in text, thus preserving the structure of the message. The texts were also preprocessed before encoding them, eliminating redundant parts and recurring formulas.

For the models we decided to use neural networks with different types of architectures and layers. We used the Rosenblatt Perceptron [15] as a binary classifier for the Procedure Type problem, as it only has 2 classes. Afterwards we extended our networks with additional layers, to achieve a finer model with better performance, and output sizes, to fit the Request type classification. Long Short Term Memory [16] layers were used, that specialize in processing ordered sequences of input. As our task is to classify texts, the order of words is tremendously relevant for the context of the given message, thus we chose to use these types of layers as they capture patterns from an ordered set of inputs. For these layers, the sequential vectors described above is essential, as they process the input element-wise and their final prediction is influenced by the order of elements as well.

Our idea of finding ticket similarities was based on Word2Vec: we wanted to encode an entire document with a vector, much like Word2Vec embeds one single word into a vector (In here a document is referred to as a text from a ticket). This method is known as Doc2Vec and it has many options of implementation (ours will be explained in the next sections), as it is our customized way of doing so. The reason why Doc2Vec helps us for finding similarities, is that after embedding our texts into vectors, we basically built a hash-function that mapped our tickets in an n-dimensional vector space, so they can be *compared*. To establish a measure of similarity between them, we decided to use the same measure that is used for the similarity of Word2Vec embeddings,i.e., Cosine similarity.

$$sim(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \tag{1}$$

The cosine distance of two vectors (1), A and B, is calculated based on their scalar product and is used as a similarity measure. The idea behind this is that the closest two vectors are to each other, the more similar their meanings in our vector space would be. Ergo, the closer the two vectors are to each other, the smaller the angle between them, the greater their cosine distance. Since the embedding vectors are fairly normalized and do not have significant variations in their norms, this cosine value is a valid unit for their similarity score calculation.

A. Additional Output Enhancements

Furthering our agenda, we needed a more practical way to interpret our results, so we adopted different metrics and visualization techniques, such as three dimensional T-SNE [17] and Receiver Operating Characteristics curve [18], described in the sections to follow. Also, in an effort to put the classification model's results into a practical perspective and augment it's value for the HelpDesk department, we devised a method to threshold it's output, in order to filter potential mistakes. This approach was meant to restrict the model's predictions under a certain threshold, artificially introducing a third class (*not classified*) to our model, and increasing the overall value our project might add to the service department, as will be explained in the results section.

IV. Implementations

During the course of our project we adopted an iterative model of development. We had multiple phases which took place sequentially, each contributing to upgrading the solution to its final state. All phases are described in detail in [2]. In the following we will describe a short summary iterations, to better understand the evolution of the project.

In the *Proof of concept* phase (or experimental phase) we wanted to test our NLP tools and methods and to do so we implemented a basic Support Vector Machine [5] or a Perceptron for the classification of movie reviews, from the Internet Movie Database (IMDB) dataset.

After our first experiment with NLP techniques was successful, we received the first batch of data to be classified based on the Procedure Type. The data consisted of around 10000 mail tickets, that were received and solved from the Helpdesk. The first models developed in the proof of concept phase,

were used with the same configurations. The main difference however, was the language of the datasets. If on our IMDB reviews were all written in English, the received tickets were all in German. For lemmatization in German, we relied on the GermaNet [19] database, from the University of Tübingen.

The difference of context between the two sets (movie reviews are more fluent then support tickets) made it harder to distinguish between Service Requests and Incident. The performance of the models were thus accordingly weaker, so we had to use more developed technologies for our models. This is where Keras [20] and LSTM-layers come into discussion. Keras is an open-source neural-network library for Python, designed for easy user-experience, as well as fast experiments with deep neural networks and customizability. We used Keras as a wrapper for the TensorFlow backend. The full architecture of the network can be seen below in Figure 1.

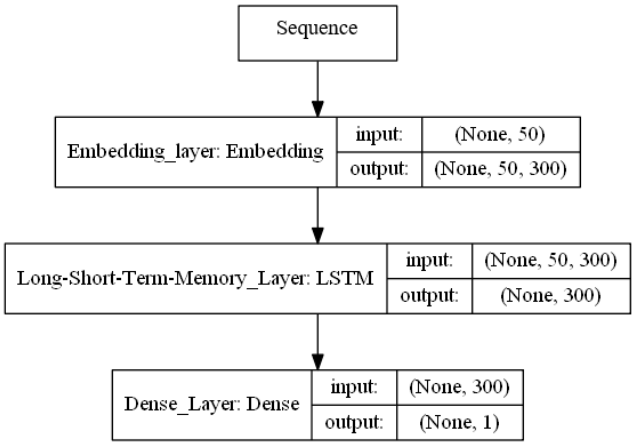

Fig. 1. NN architecture

Following our success with the first batch, we received a second batch of data and concatenated them both, resulting in a total of 48775 tickets on which we ran our Procedure Type classifier again, and in addition a Request Type classifier with the same architecture. In order to avoid skewed classes training [3] for the Request Type model, we selected only those classes with a frequency of appearance of more than 1000 times, thus we were left with 22134 entries for our dataset, spread around 12 different classes. Additionally, o enhance resolution, we also added an extra LSTM-layer to each of the models, identical to the first one.

After classifying tickets, tended to the embedding extraction matter, used for computing similarities. We took the penultimate layer output from the Request Type model (the added LSTM-layer, right before the output layer) for representing vector embeddings. After these embeddings have been extracted, the cosine distance formula can now be used to calculate ticket similarities. With the help of the T-Stochastic neighborhood [17] algorithm, we plotted all extracted ticket embeddings to visualize them . TSNE is a way to reduce multi-dimensional data sets to 2 or 3-dimensional vectors, to be seen in a vector space that is humanly perceptible. This reduction in dimensionality is calculated based on likelihood between multi-dimensional vectors.

In order to asses that our embeddings were working and the similarity measure used was correct, we used alpha-clustering algorithms to create ticket hubs in our plot to visually illustrate similarities: two tickets were connected via a line each which had the similarity (based on their network extraction) greater than a predefined threshold α. We could thus see how refined the classifiers got through training and how useful the cosine distance measure can be in order to find similar tickets.

V. Results and Analysis

The results and performances of all the models in each iteration are presented thoroughly in [2]. We are going to mention only the best performing models.

A. Procedure Type classifier

After augmenting it with a second LSTM-layer and training on an extended dataset of roughly 48000 entries, the performance of the Procedure Type classification model can be viewed below in Table I, with it's validation accuracy peaking at 86.5%.

TABLE I
METRICS SCORES OF LSTM MODEL ON PROCEDURE TYPE

Class	Precision	Recall	F1 score	Accuracy
Service Request	91%	91%	0.91	86.5%
Incident	72%	71%	0.72	

B. Request Type Classifier

For the Request Type classification we used the same model in the same training context. The performance results can be viewed in Table II.

TABLE II
METRICS SCORES OF LSTM MODEL ON REQUEST TYPE

Class	Precision	Recall	F1 score	Accuracy
Other Installations	79%	89%	0.84	67%
VPN Renewal	66%	85%	0.74	
Outlook/Exchange	58%	77%	0.66	
Mail	71%	38%	0.50	
Hardware Order	65%	63%	0.64	
New Installation	67%	48%	0.56	
Onsite Support RO	79%	79%	0.79	
VPN Client Notebook	49%	53%	0.51	
Password Recovery	85%	76%	0.80	
Standard Accessory	53%	50%	0.51	
Lync	72%	82%	0.77	
VPN Client	50%	31%	0.38	

C. Similarities Plots

Based upon the Procedure type model, we extracted embeddings as described above and plotted the TSNE graph as illustrated below. In Figure 2, each dot represents a ticket from a sample of 1000 entries from our data set, that has been reduced to a 2-dimensional point. Through the clustering algorithm, we connected each 2 tickets with a similarity score greater than 0.6 with a line, thus creating two big hives of data. Upon a

close look we discovered that these clusters represented the accumulating points of our two classes: Service Request and Incident, and our network would most of the time place an entry inside the vectorial space of these 2 hives.

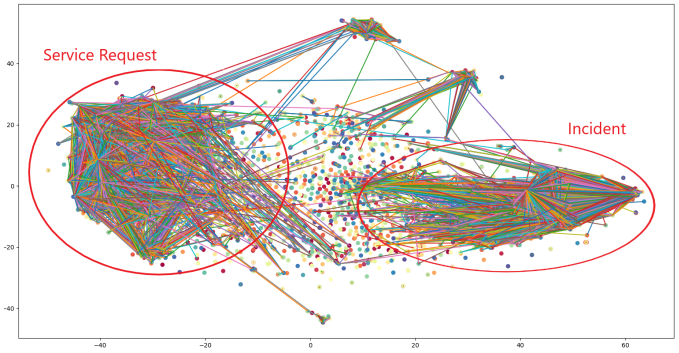

Fig. 2. Service Request vs. Incident clustering visualization

For the Request type, we applied the same algorithm leading to similar results. We can see more clusters forming in Figure 3 and numerous similarity connections between tickets of the same cluster.

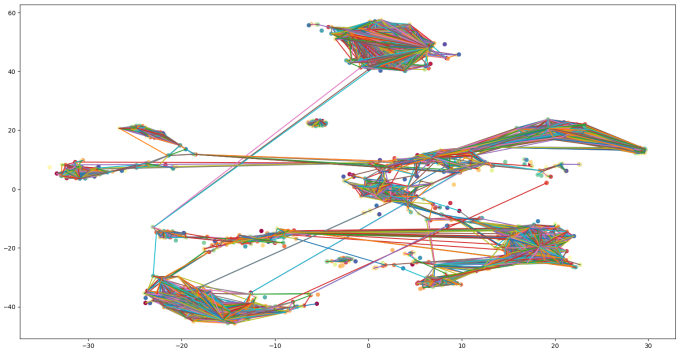

Fig. 3. Request Type clustering visualization

D. Request Type Clustering

We also plotted all tickets from a class to a specific color, in order to better identify the forming of clusters for the Request Type agent. Upon closer inspection we noticed an interesting phaenomenon: tickets from classes of similar semantic grouped together. This can be visualized in Figure 4, where strongly defined clusters contain points from 2 similar classes that merge together: Outlook / Exchange with Mail, Hardware Order with Standard Accessory which basically represent hardware and accessories requests, VPN Client Notebook with VPN Client and New Installation with VPN Renewal, which have to deal with overall authorization certificate renewal and creation.

This anomaly inspired us to retrain our model in a different way: to combine every grouping class in one and so refine the data in the hope of improving its performance. As shown in Table III, the newly trained 8-class model achieved 80.8% precision and much higher measurements in the preliminary unmerged classes. The result was impressive. After this success we computed the clustering analysis again, this time with

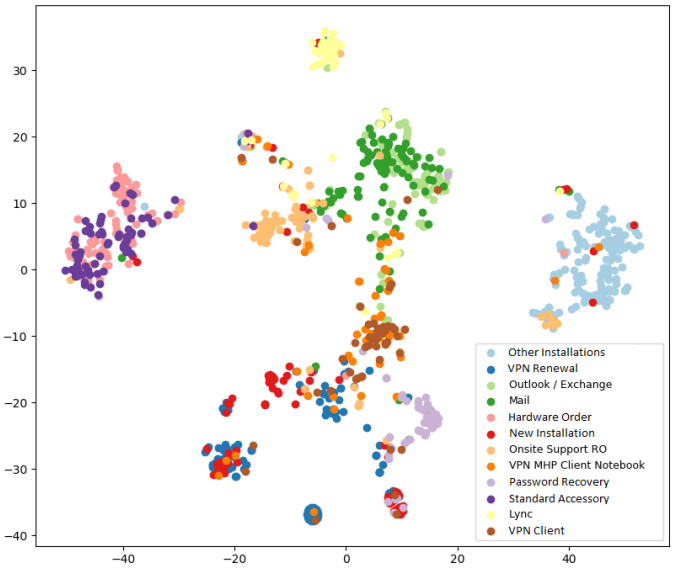

Fig. 4. Request Type clustering visualization: each class highlighted

merged classes. The results were significantly better, as seen in Figure 5.

TABLE III
METRICS SCORES OF LSTM MODEL ON REQUEST TYPE MERGED

Class	Precision	Recall	F1 score	Accuracy
Other Installations	85%	88%	0.87	80.8%
VPN Renewal	82%	86%	0.84	
Mail	78%	83%	0.81	
Hardware Order	88%	95%	0.91	
Onsite Support RO	80%	67%	0.73	
VPN Client Notebook	69%	58%	0.63	
Password Recovery	79%	76%	0.77	
Lync	81%	68%	0.74	

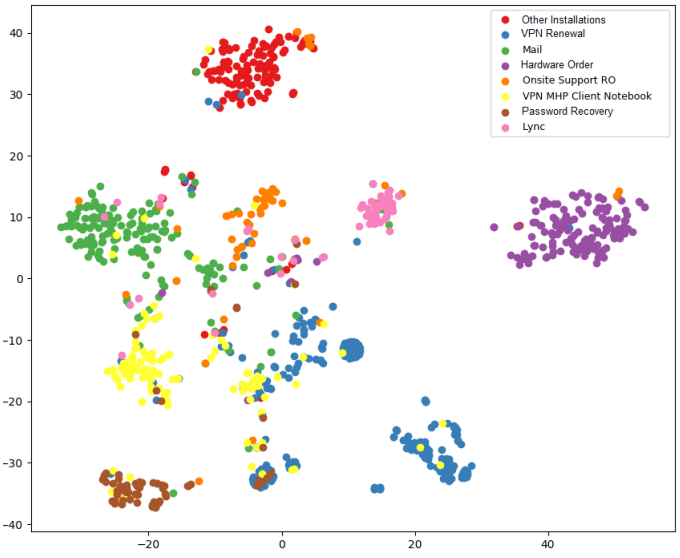

Fig. 5. Request Type clustering visualization: merged classes

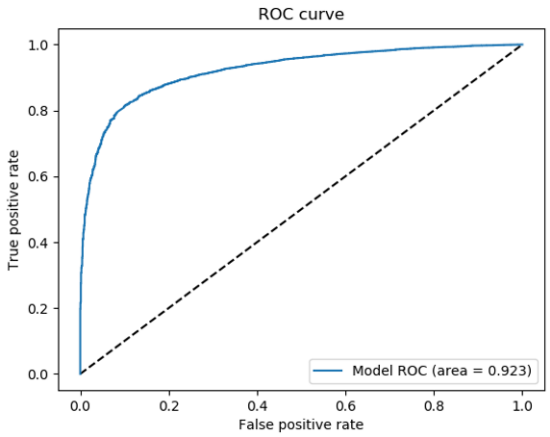

Fig. 6. Receiver Operating Characteristics Curve

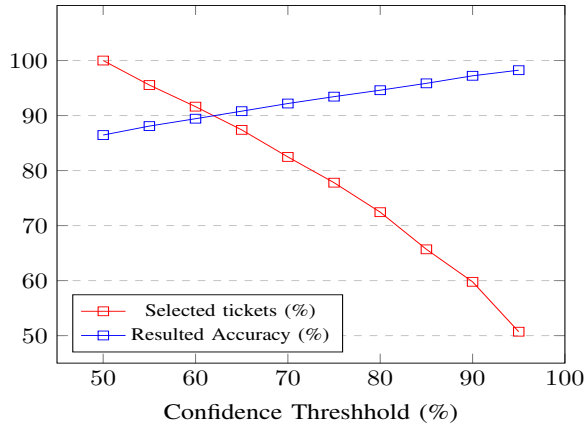

Fig. 7. Confidence score trade-off

VI. Enlarging The View: Additional Features

Succeeding our work from [2], we aimed to provide a more detailed perspective of what our models where capable of, and how we could use them in our business scenario more effectively. To do so, we extended our project with different illustration and performance measurement techniques.

A. Receiver Operating Characteristics Curve

In addition to the metrics presented in [2], there is one additional metric suitable for measuring the performance of our binary classifier: the Receiver Operating Characteristics curve [18], or ROC. ROC is a plot that measures the diagnostic ability of a binary classifier. It does so by illustrating the trade-off between the True positive rate (TPR) and the False positive rate (FPR). These rates are calculated as follows, where P represent all the positives in our data, TP the true positives and FP the false positives. Also, this trade-off is a helpful interpretation input for the business domain, as it gives an in-depth prediciton of the model's performance in a real-life scenario.

$$TPR = \frac{TP}{P}$$

$$FPR = \frac{FP}{P}$$

The curve is computed by changing our classifier's threshold of prediction (0.5 by default) to all values of the interval $[0, 1]$ and calculating TPR and FPR on the validation data, based on this new thresholds. Figure 6 illustrates the plot of these values for the model. The ideal ROC curve will be very close to hugging the upper-left corner of the plot ($(1, 0)$ point), so the better the classifier, the closer this curve will be to this corner. To measure the diagnostic performance of a model based on it's ROC curve, we have the Are Under the Curve (AUC) [21] measure, computed by integrating the function of the ROC curve on the $[0, 1]$ interval, and represents the probability that our classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one. After calculating and integrating our curve, we were left with an AUC value of 0.923, which is pretty close to 1, thus showing remarkable performance of the model.

B. Confidence Threshold

After obtaining an accuracy of 86.5%, we we're informed that this number is comparable to the actual efficiency of the Helpdesk department in sorting tickets (empirically measured). This is due to the fact that working in the context of a real life problem, sometimes definitive separation decisions can not be taken. Semantics are sometimes vague in real life, and this tells us that not all tickets are completely separable in 2 or more classes. Following our work from [2], this is the reason we came up with the idea of building a *confidence threshold*.

A confidence threshold is a margin of estimation we impose to our predictor: every input that yields a prediction probability under said confidence threshold, gets left unclassified. Until now, as the model is a binary classifier with one output, everything predicted over 50% was assigned to one class, and everything under 50% was assigned to the other. This means that our basic predictor is making decision with a confidence threshold of 50%: so everything gets classified. We wanted to see what happens when we raise this margin: consequentially, the total number of classified tickets will decrease, but the total accuracy will increase. We plotted the evolution of these two numbers in Figure 7, that represents the trade off of classifying more tickets vs. obtaining a higher accuracy. Here we can see the effects this confidence threshold will have on the classification process, and the Service department can take an informed business decision of what threshold to use, in order to optimize the process.

C. Comparing Results

By adding this concept of a confidence threshold, we are artificially adding a new class to our model, so we now have 3 classes: *Service Request, Incident and Not Classified*. To put these numbers from Figure 7 into perspective, let's visualize the distribution of classification impacted by this confidence threshold. Figure 8 compares the standard, or conventional output of our previous model (confidence threshold 50% - so just the binary model) with the improved, confidence-conditioned model (confidence threshold 60% and 70%).

We can see the percent of incorrectly classified tickets falling, although we now encounter unclassified entries: it is preferable to have less wrongly classified tickets, with the drawback of

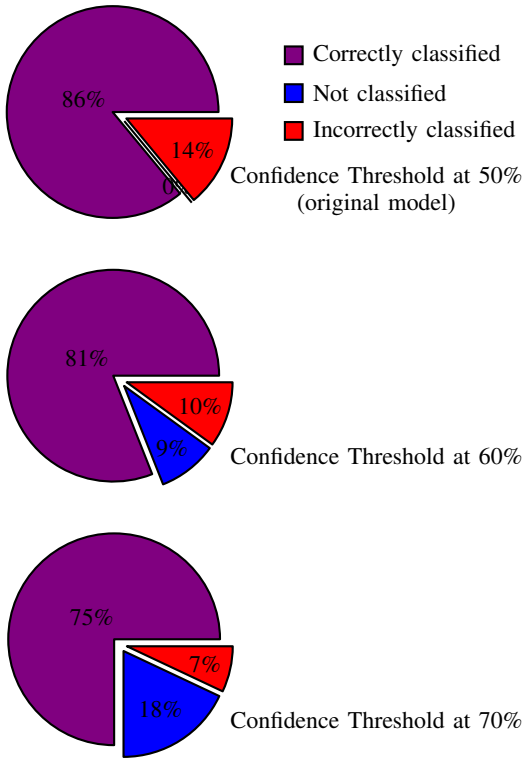

Fig. 8. Ticket classification based on confidence threshold

unclassified tickets, rather then having classified all tickets, but with many labeled incorrectly. This is because the workload for wrongly labeled tickets greatly exceeds the workload for simply unclassified tickets, since the former imply returning to previous service-desk levels.

D. 3D Clustering Representation

Following our goal to extend the interpretation of our results from [2], we plotted the data in 3D as well, for a better point of view. This allowed us to have a more detailed look as to how our categories clustered together, and consequently how well was our Doc2Vec embedding method able to capture the semantics of the tickets. These plots can be seen in Figures 9 through 12.

VII. Conclusion

Our numerous results achieved in these experiments are a good indicator that there are plausible ways of aiding the ticket solving process. By classifying tickets in their corresponding categories, we can considerably reduce the time a ticket spends in pending status, thus allowing the Service department to focus more on solving requests instead of sorting them. Furthermore, by computing ticket similarities we can provide a connected platform of solved requests that will benefit operators in finding the solution quicker. Our idea of extracting output from hidden layers proved to be a trustworthy method of computing Doc2Vec embeddings, and are able to correctly represent relevant features of our data. We think the proposed methods of automation are very viable and can be successfully and sustainably integrated in the workflow of the Helpdesk department.

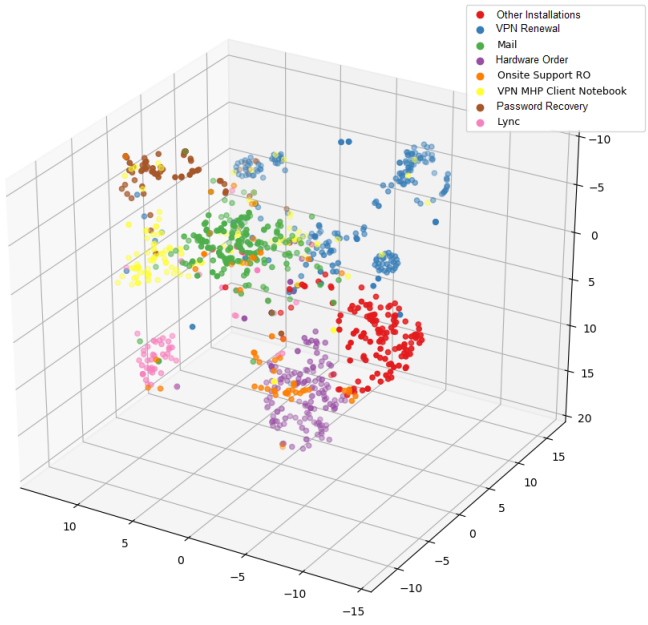

Fig. 9. Request Type clustering visualization: merged classes

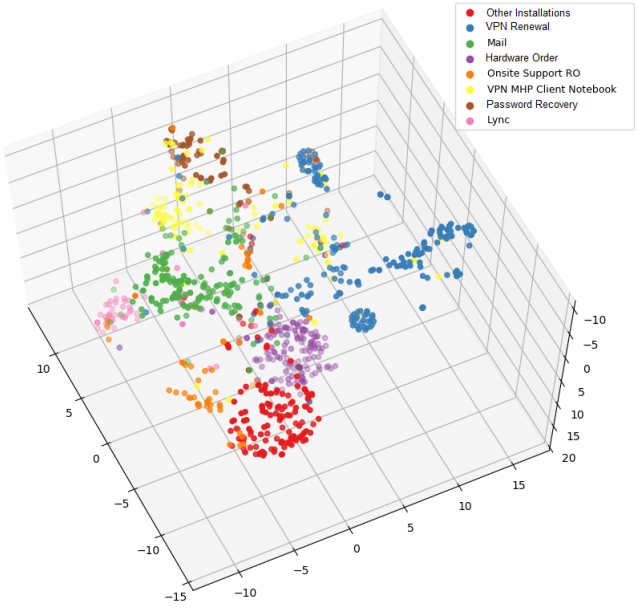

Fig. 10. Request Type clustering visualization: merged classes

References

[1] Manning, C. D., Manning, C. D., & Schütze, H. (1999). Foundations of statistical natural language processing. MIT press.

[2] M. Cristian, S. Christian and T. Dumitru-Tudor, "A Study in the Automation of Service Ticket Recognition using Natural Language Processing," 2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 2019, pp. 1-6. https://doi.org/10.23919/SOFTCOM.2019.8903676

[3] Ricamato, M. T., Marrocco, C., & Tortorella, F. (2008, December). Mcs-based balancing techniques for skewed classes: An empirical comparison. In 2008 19th International Conference on Pattern Recognition (pp. 1-4). IEEE. https://doi.org/10.1109/ICPR.2008.4761359

[4] Awad, W. A., & ELseuofi, S. M. (2011). Machine learning methods for spam e-mail classification. International Journal of Com-

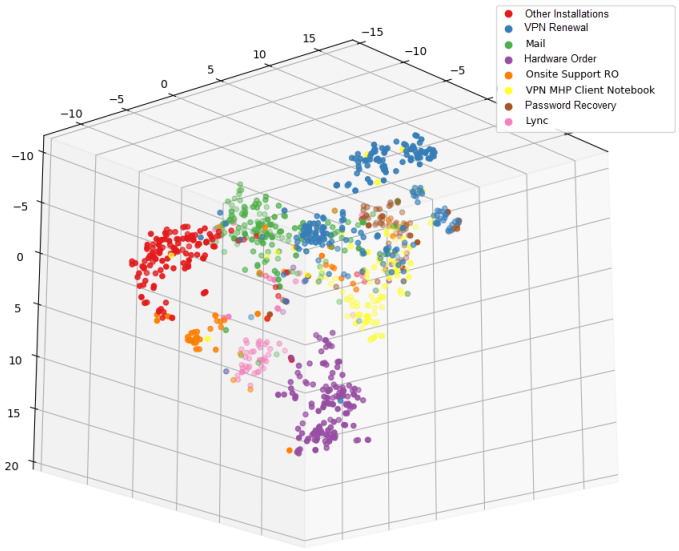

Fig. 11. Request Type clustering visualization: merged classes

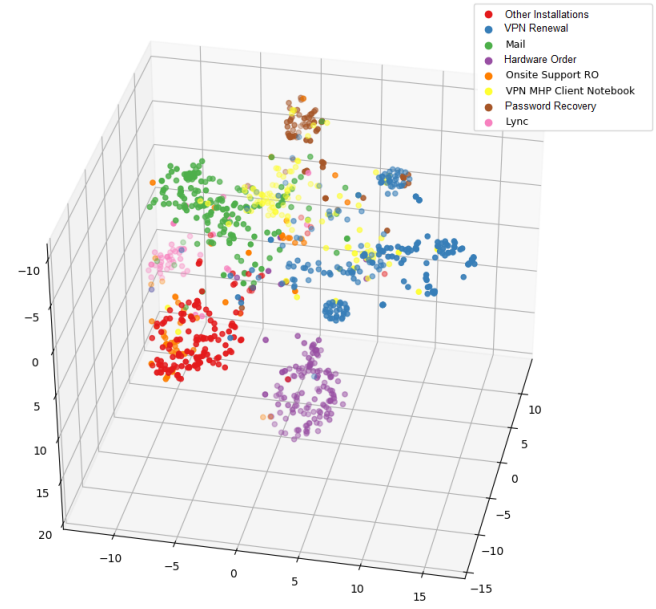

Fig. 12. Request Type clustering visualization: merged classes

puter Science & Information Technology (IJCSIT), 3(1), 173-184. https://doi.org/10.5121/ijcsit.2011.3112

[5] Suykens, J. A., & Vandewalle, J. (1999). Least squares support vector machine classifiers. Neural processing letters, 9(3), 293-300. https://doi.org/10.1023/A:1018628609742

[6] Schalkoff, R. J. (1997). Artificial neural networks (Vol. 1). New York: McGraw-Hill.

[7] Carpinteiro, O. A., Lima, I., Assis, J. M., de Souza, A. C. Z., Moreira, E. M., & Pinheiro, C. A. (2006, September). A neural model in anti-spam systems. In International Conference on Artificial Neural Networks (pp. 847-855). Springer, Berlin, Heidelberg. https://doi.org/10.1007/11840930_88

[8] Lai, S., Xu, L., Liu, K., & Zhao, J. (2015, February). Recurrent convolutional neural networks for text classification. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 29, No. 1).

[9] Williams, Ronald J.; Hinton, Geoffrey E.; Rumelhart, David E. (October 1986). "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536. doi:10.1038/323533a0. ISSN 1476-4687

[10] Arevian, G. (2007, November). Recurrent neural networks for robust real-world text classification. In IEEE/WIC/ACM International Conference on Web Intelligence (WI'07) (pp. 326-329). IEEE. https://doi.org/10.1109/WI.2007.126

[11] Halgaš, L., Agrafiotis, I., & Nurse, J. R. (2019, August). Catching the Phish: Detecting Phishing Attacks using Recurrent Neural Networks (RNNs). In International Workshop on Information Security Applications (pp. 219-233). Springer, Cham. https://doi.org/10.1007/978-3-030-39303-8_17

[12] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[13] Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543). https://doi.org/10.3115/v1/D14-1162

[14] Harris, Z. S. (1954). Distributional structure. Word, 10(2-3), 146-162. https://doi.org/10.1080/00437956.1954.11659520

[15] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386. https://doi.org/10.1037/h0042519

[16] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

[17] Maaten, L. V. D., & Hinton, G. (2008). Visualizing data using t-SNE. Journal of machine learning research, 9(Nov), 2579-2605.

[18] Streiner, D. L., & Cairney, J. (2007). What's under the ROC? An introduction to receiver operating characteristics curves. The Canadian Journal of Psychiatry, 52(2), 121-128. https://doi.org/10.1177/070674370705200210

[19] Hamp, B., & Feldweg, H. (1997). Germanet-a lexical-semantic net for german. In Automatic information extraction and building of lexical semantic resources for NLP applications.

[20] Chollet, F. (2018). Keras: The python deep learning library. Astrophysics Source Code Library, ascl-1806.

[21] Hanley, J. A., & McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology, 143(1), 29-36. https://doi.org/10.1148/radiology.143.1.7063747

Dumitru-Tudor Tolciu is a student at the Babes-Bolyai University of Cluj-Napoca. He received his Bachelor Degree in Computer Science at the Babes-Bolyai University, Faculty of Mathematics and Computer Science where he graduated as valedictorian. He currently studies for his Masters degree, and his fields of research include Artificial Intelligence, Natural Language Processing and Computer Vision.

Christian Săcărea is Assistant Professor at the Babes-Bolyai University of Cluj-Napoca. He received his MSc. Degree at the Babes-Bolyai University and his PhD degree (Dr. rer. nat.) at the Darmstadt University of Technology, Germany. He is the Head of the Formal Concept Analysis Lab at the Babes-Bolyai University. His current research interests are Knowledge Discovery, Knowledge Processing and Representation.

Cristian Matei is a Computer Science graduate from the Babeș-Bolyai University Cluj-Napoca and currently a Master's student at the same university, studying Advanced Computer System. He has been working for one and a half years in the field of Data Science, with projects in the areas of medical AI and digital reenactment