UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Data Science Curriculum

Õie Renata Siimon

Patient Treatment Trajectories Using Vector Embeddings

Master's Thesis (15 ECTS)

Supervisor: Sven Laur (PhD)

Tartu 2023

Patient Treatment Trajectories Using Vector Embeddings

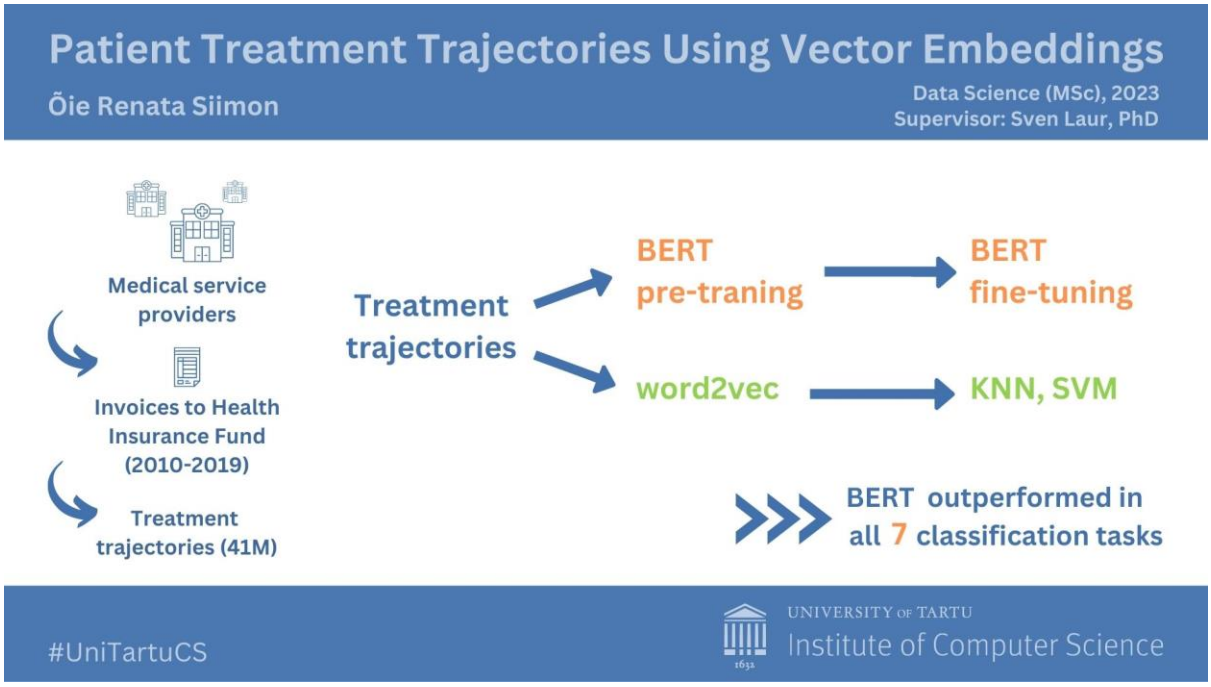
Abstract

In this thesis, data from Estonian Health Insurance Fund (Haigekassa) in 2010–2019 was used to construct vector representations of patient treatment trajectories with BERT, and for comparison, with word2vec. The goal was to see how well such natural language processing (NLP) models perform when sequences of medical services are used as input instead of sentences, and if BERT performs better than word2vec. So far, research on how well NLP models work with non-natural language sequences is limited, and this thesis contributes to filling this gap. In this thesis, treatment trajectories were built as sequences of service codes appearing on 41 million medical invoices. Models in this thesis were constructed in two stages. First, service code embeddings were trained with BERT and word2vec. Then, classification models were built by fine-tuning BERT and training KNN and SVM classifiers on top of word2vec embeddings. Results showed that despite poor performance of BERT in pre-training stage, it outperformed models built on top of word2vec embeddings in all seven classification tasks. The highest accuracy (0.9918) was achieved in classifying treatment types (5 classes) and the lowest (0.4121) in classifying diagnosis (174 classes). It was concluded that BERT indeed proved useful with this type of non-natural language input data, and that the contextual embeddings of BERT worked better than non-contextual ones of word2vec. From among the four BERT models built in this thesis, the second largest was the overall best, showing that if the 'language' used is simpler than natural language, then BERT models with reduced dimensions might work better.

Keywords: machine learning, treatment trajectory, medical bill, word2vec, BERT

CERCS: P176 – Artificial intelligence, B110 – Bioinformatics, medical informatics, biomathematics, biometrics

Graphical abstract



Ravitrajektooride koostamine sõnavektorite abil

Lühikokkuvõte

Magistritöös konstrueeritakse Haigekassa andmete (2010–2019) põhjal patsientide ravitrajektooride vektoresitused, kasutades selleks BERTi ja võrdluseks word2vec'i. Töö eesmärk on näha, kui hästi need loomuliku keele töötluse (NLP) mudelid töötavad, kui lausete asemel on sisendiks raviteenuste aegread, ning kas BERT on tulemuslikum kui word2vec. Seda, kui tulemuslikud on NLP mudelid mittekeeleliste andmete peal, on siiani vähe uuritud, ja käesolev töö annab panuse selle lünga täitmisse. Siin magistritöös koostati ravitrajektoorid 41 miljonil raviarvel olnud teenuskoodidest. Seejärel treeniti mudelid kahes etapis. Kõigepealt treeniti BERTi ja word2vec'iga teenuskoodide vektoresitused. Seejärel treeniti klassifitseerimismudelid, ühelt poolt peenhäälestades selleks BERTi ja teiselt poolt treenides word2vec'i vektoresituste peal KNN ja SVM klassifitseerimismudelid. Tulemused näitasid, et vaatamata BERTi kehvale sooritusele eeltreenimise etapis olid peenhäälestatud BERTi mudelid kõigis seitsmes klassifitseerimisülesannetes word2vec vektoritel treenitutest tulemuslikumad. Suurima täpsuse (0.9918) saavutas BERT raviliikide (5 klassi) ja vähima (0.4121) diagnooside (174 klassi) klassifitseerimisel. Töös järeldati, et BERT osutus ravitrajektooride klassifitseerimisel kasulikuks, vaatamata sellele, et tegu ei olnud loomuliku keele andmetega. Samuti järeldati, et BERTi konteksti arvestavad vektoresitused töötavad ravitrajektooridel word2vec'i konteksti mitte arvestavatest paremini. Neljast BERTi mudelist osutus kokkuvõttes parimaks suuruselt teine, mis näitab, et kui kasutatav 'keel' on loomulikust keelest lihtsam, siis võivad paremini töötada vähendatud mõõtmetega BERTi mudelid.

Võtmesõnad: masinõpe, ravitrajektoor, raviarve, word2vec, BERT

CERCS: P176 – Tehisintellekt, B110 – Bioinformaatika, meditsiiniinformaatika, biomatemaatika, biomeetrika

Visuaalne kokkuvõte

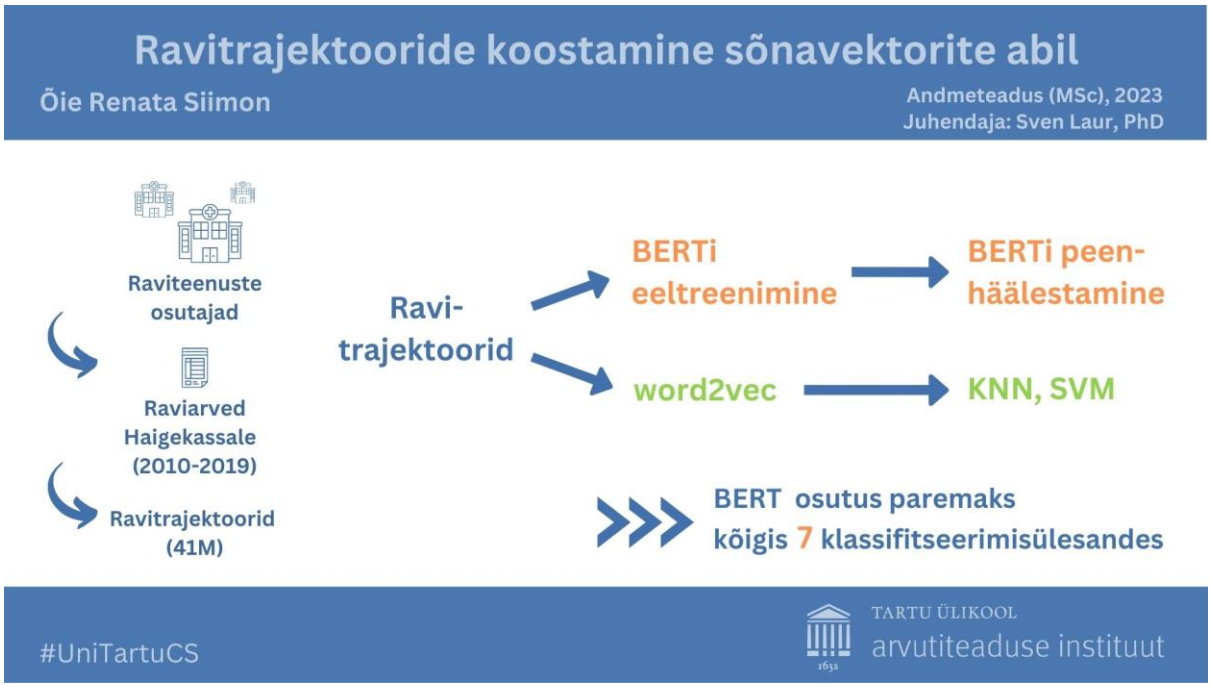


Table of contents

1. Introduction

In the last decade, the field of natural language processing (NLP) has been developing fast and has seen astonishing achievements. It has been shown that the advances in NLP can in some cases be transferred also to other fields, where the structure of the data shares some properties with natural language. For example, BERT (Bidirectional Encoder Representations from Transformers) [1], a state-of-the-art language model also used in this thesis, or its derivations have been successfully applied in modelling the 'language' of DNA as a sequence of nucleotides [2] and proteins as a sequence of amino acids [3]. In the field of electronic health records, its derivation BEHRT [4] was used for predicting future diagnosis based on sequences of previous visits, each of which could contain multiple diagnoses. In the same field, G-BERT [5] combined BERT with graph neural networks for modelling the hierarchical structure of the ontology of medical codes and for recommending medication, based on sequences of previous medications and previous and current diagnosis. In all those examples, the input data shared some similarity to natural language, since much like words in a sentence, the input sequences consisted of temporally ordered discrete events, with some inner logic governing how items in a sequence were ordered. This thesis contributes to the above research by looking at how well BERT performs on patient treatment trajectories, consisting of sequences of services provided by medical institutions to a patient, and by comparing its performance to word2vec [6], which is a simpler NLP model.

More specifically, the aim of this thesis is to evaluate the suitability of applying BERT to patient treatment trajectories, by measuring its performance in various tasks related to both its pre-training and fine-tuning stage, and also comparing its performance with word2vec and classifiers built on top of it. The task in pre-training stage, where BERT learns general language representation through masked language modelling (MLM) approach, was to predict service codes that were masked out in the trajectory. Its performance in this task is compared to the performance of word2vec in predicting the target 'word' (service code) in the middle of its context window. In the fine-tuning stage, performance of BERT is measured in seven classification tasks (incl. predicting diagnosis, treatment type, discharge status, type of service provider, and if the invoice was for emergency services). Its performance in this task is compared to k-nearest neighbours (KNN) and support vector machine (SVM) classifiers built on top of vector representations of treatment trajectories obtained from word2vec.

The reason for comparing BERT with word2vec is that both learn vector representations of words (also called 'word embeddings') during training. The difference is that embeddings learnt by BERT are contextual (the same word has different embedding depending on the context where it occurs), while the embeddings learnt by word2vec are static (for each word in its vocabulary, the model learns a fixed embedding) [7]. Therefore, comparing BERT and word2vec allowed to see if contextual embeddings work better with treatment trajectories or not.

Since the 'language' of medical services is likely to be less complex than natural language, four BERT models of different dimensions were built in this thesis in order to see if models smaller than the one proposed by the authors of BERT [1] work better with treatment trajectories.

Data used in this thesis were invoices submitted by medical institutions for reimbursement to the Estonian Health Insurance Fund (Haigekassa) in 2010-2019, from which 41 million treatment trajectories were constructed. The features predicted in the classification models were obtained from an accompanying dataset containing further details on a small subset (62,000) of those invoices.

The results of the models are further analysed from a number of aspects. The results of pre-trained BERT and word2vec models are first further analysed by breaking accuracies down by trajectory length. Secondly, principal component analysis (PCA) is performed on embeddings of 10 selected services of one of the pre-trained BERT models in order to better understanding its results. Thirdly, the possible effect of masking strategy on performance is discussed. The results of fine-tuning BERT are further analysed by discussing prediction errors of the best BERT model based on confusion matrices. Also, a small case study is performed on two fine-tuned BERT models to see if having more training data could have helped improve performance of the classification models.

The models were implemented in Python programming language, using Hugging Face Transformers library[1] (vers. 4.7.0) for BERT models and scikit-learn library[2] (vers. 0.24.2) for other models. Link to the GitLab repository of this thesis is given in Appendix I.

The rest of this thesis is organized as follows. In Chapter 2, the background is explained, including the description of BERT and word2vec models, as well as related works where BERT has previously been applied to non-natural language data. In Chapter 3, the datasets used, as well as pre-processing and descriptive statistics of both treatment trajectories and predicted features are described. Chapter 4 contains methodology, explaining how the models were built and which performance metrics were used. The results are set out in Chapter 5 and conclusions in Chapter 6.

[1] https://huggingface.co/docs/transformers/index
[2] Scikit-learn User Guide: https://scikit-learn.org/stable/user_guide.html

2. Background

In this Chapter, the two main models used in this thesis – BERT and word2vec – are described (see Sections 2.1 and 2.2), followed by description of some related works where BERT has previously been applied to non-natural language data.

As regards the main models, BERT is a state-of-the-art language model developed by Devlin *et al.* [1] in 2019. In that article, the authors demonstrated that BERT can achieve excellent performance in a wide range of NLP tasks, like question answering, named entity recognition and language inference. Word2vec is an older and less complex model, developed by Mikolov *et al.* [6] in 2013. Both models have been extensively used by the natural language research community[3].

Both BERT and word2vec learn word embeddings during training. An embedding is a real valued vector, usually of length 50-1000, which represents the meaning of a word [7]. The idea that word meanings can be represented with real valued vectors relies on distributional hypothesis, which states that words occurring in similar contexts tend to have similar meanings [*Ibid.*]. Therefore, a language model can learn word embeddings by going through the text and looking at which words tend to occur nearby each other, iteratively adapting embeddings along the way, so that embeddings of words that appear near each other become more similar and those that do not become less similar. In this process, BERT looks at the whole sentence at a time and also takes word order into account [8], while word2vec looks only at words in a fixed sized context window and treats them as a bag-of-words, i.e. without considering word order.

An important difference between word2vec and BERT is that embeddings learnt by BERT are contextual (the same word may have different embedding depending on the context where it occurs), while the embeddings learnt by word2vec are static (for each word in its vocabulary, the model learns a fixed embedding) [7]. In both models, vocabulary is a dictionary that maps tokens (i.e. words, and in case of BERT, also some special tokens) to their indices. Both word2vec and BERT use a fixed sized vocabulary. For example, the BERT model developed in [1] had vocabulary size 30,000. Less frequent words that do not fit in the vocabulary are treated as out-of-vocabulary or unknown (UNK). In this thesis, the vocabulary contained service codes instead of words, while patient treatment trajectory (i.e. sequence of medical service codes on an invoice) was treated as a sentence.

2.1 Word2vec

In word2vec, the inputs are gathered by moving a fixed sized window through the text, for each sample storing the word in the middle of the window as the target word and the words around it as context words. Word2vec has two architectures: CBOW (continuous bag-of-words) and skipgram (see Figure 1), both of which are log-linear models [6]. In the figure, *w(t)* is the target word and *w(i-2)*, *w(i-1)*, *w(i+1)* and *w(i+2)* are the context words. In CBOW, the goal is to predict the target word based on the context words, and in skipgram, to predict context words based on the target word [6]. In this thesis, both architectures were considered, but finally only CBOW was used, because it showed slightly better performance in initial experiments.

[3] In Google Scholar, there are 15,400 papers mentioning "Bidirectional Encoder Representations from Transformers" and 87,200 papers mentioning "word2vec" in their text.

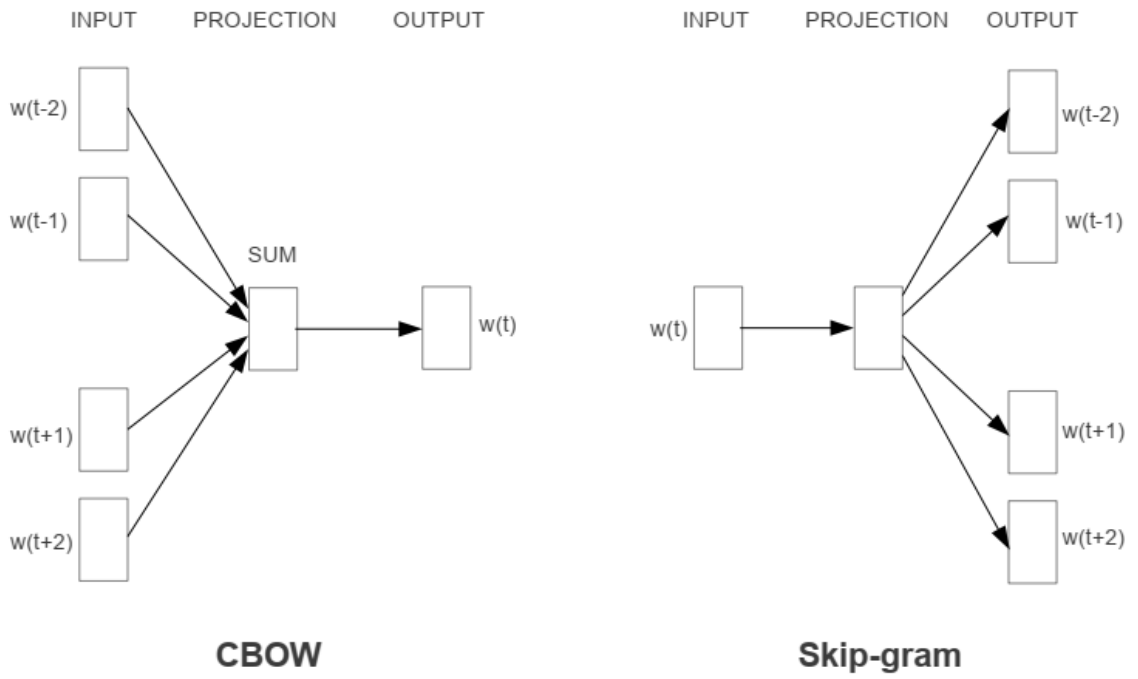


Figure 1. CBOW and skipgram architectures of word2vec [6]

Word2vec stores two weights matrices, one containing embeddings for words when used in context, and the other one containing words when used as target word [7]. This means that in fact two embeddings are learnt for each word. There are two alternative ways for obtaining the final embeddings – one is to add up the two embeddings for each word, and the other one is to simply disregard the context word embeddings and only use the target word embeddings *[Ibid]*.

In case of CBOW, the model selects the vectors of context words from the context words matrix, then either averages or concatenates [9] or sums [7], [10] them, and sends the resulting aggregated vector to the classifier (see Figure 1). In case of skipgram, the model selects the vector of the target word from the target words matrix and sends it to the classifier (see Figure 1).

The classifier in word2vec is logistic regression [7]. There are two alternative optimization objectives that can be used: negative sampling or hierarchical softmax [10]. The models in this thesis used negative sampling. As explained in [*Ibid.*], the idea of negative sampling is that for each positive sample (actual word-context pair), the classifier also needs *k* negative samples (target word that does not appear together with the given context words in case of CBOW, or *vice versa* in skipgram), which are randomly sampled from the input text. The classifier is then trained to distinguish the positive samples from the negative ones.

As explained in [7], the model starts learning with randomly initialized weights matrices. During training, as it walks through the data, it adapts the weights using stochastic gradient descent to maximize the similarity of positive samples and minimize the similarity of the negative samples. In general, the similarity of two words can be calculated as cosine of the angle between their word vectors $\boldsymbol{u}$ and $\boldsymbol{v}$:

$$cosine\ similarity\ (\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{v} \cdot \boldsymbol{u}}{|\boldsymbol{v}||\boldsymbol{u}|} = \frac{u_1 v_1 + \ldots + u_n v_n}{\sqrt{u_1^2 + \ldots + u_n^2}\ \sqrt{v_1^2 + \ldots + v_n^2}} \quad (1)$$

However, the similarity calculated when training word2vec is simply dot product between the word vectors (i.e. numerator in the formula above), without normalizing the vectors to unit

vectors (of length 1) by dividing them by vector lengths [7]. As explained in *[Ibid.]*, dot product still reflects similarity between vectors (similar vectors have higher dot product, since they tend to have high values in the same dimensions). At the same time, it has the disadvantage of favouring long vectors (dot product is higher for longer vectors, which tend to correspond to more frequent words, since those words co-occur more frequently with any other words).

2.2 BERT

The description of BERT in this section is based on the paper by Devlin *et al.* [1]. As regards model architecture, BERT is a multi-layer bidirectional transformer encoder model. According to the authors of BERT [*Ibid.*], the Transformer is implemented almost identically to the one proposed by Vaswani *et al.* [11]. In this thesis, BERT was used as a black-box model, and therefore its underlying Transformer architecture is not here described.

As explained in [1], BERT is trained in two stages: pre-training and fine-tuning. In the pre-training stage, general language representation is learnt by training the model on large amount of unlabelled data in a self-supervised manner. This is done though masked language modelling (MLM) approach, where 15% of input tokens are masked (replaced by MASK token), and the task is to predict vocabulary ids of those masked tokens. The MLM approach makes BERT a bidirectional model (i.e. it learns from both left and right hand context), as opposed to some earlier models like ELMo [12] and OpenAI GPT [13], which are unidirectional (i.e. trained to predict next token based on preceding ones). Inputs to BERT can be either single sentences or sentence pairs. The possibility to use sentence pairs as input is essential for some NLP tasks (e.g. question answering), while not that relevant for others (e.g. sentence classification or tagging). If sentence pairs are used as input, an additional objective besides minimizing MLM loss is also to minimize next sentence prediction (NSP) loss [1]. In this thesis, inputs were single sentences (treatment trajectories), since due to lack of patient ids in data, consecutive trajectories (i.e. invoices of the same patient) could not be identified.

As further explained in [*Ibid.*], in the fine-tuning stage, a classification layer is added to the model, and then the same model is trained further on a relatively small amount of labelled data to perform classification tasks. These can either be sentence level tasks, where a label is predicted for the whole sentence (e.g. sentiment analysis), or token level tasks, where a label is predicted for each token in the sentence (e.g. part-of-speech tagging). The classification tasks in this thesis were all at sentence level.

The two stages of training BERT are shown in Figure 2. This figure is from the original paper of BERT [*Ibid.*] and depicts the case where a sentence pair is used as input. In Figure 2, the input sequence (where Tok_i represents i-th token) is marked in pink, the input embeddings E_i in yellow and embeddings in the final hidden state T_i in green. Special token CLS is added to the start of each input sequence, and SEP token to the end of each sentence. The final hidden state corresponding to the CLS token is used for sentence level classification tasks (for NSP in pre-training and for different downstream classification tasks in fine-tuning). As shown in Figure 2, the outputs of pre-training are predictions for the masked tokens (Mask LM in the figure) and prediction of the second sentence following the first one or not (NSP). The same pre-trained model can be fine-tuned for different tasks (resulting in several fine-tuned models) [*Ibid.*]. Examples of tasks shown in Figure 2 are language inference (MNLI), named entity recognition (NER) and question answering (SQuAD). For example, in SQuAD the input sentence pair consists of a question (first sentence) and a paragraph (second sentence), and the task is to predict where the answer to the question can be found in that paragraph (i.e. the start and end of its span).

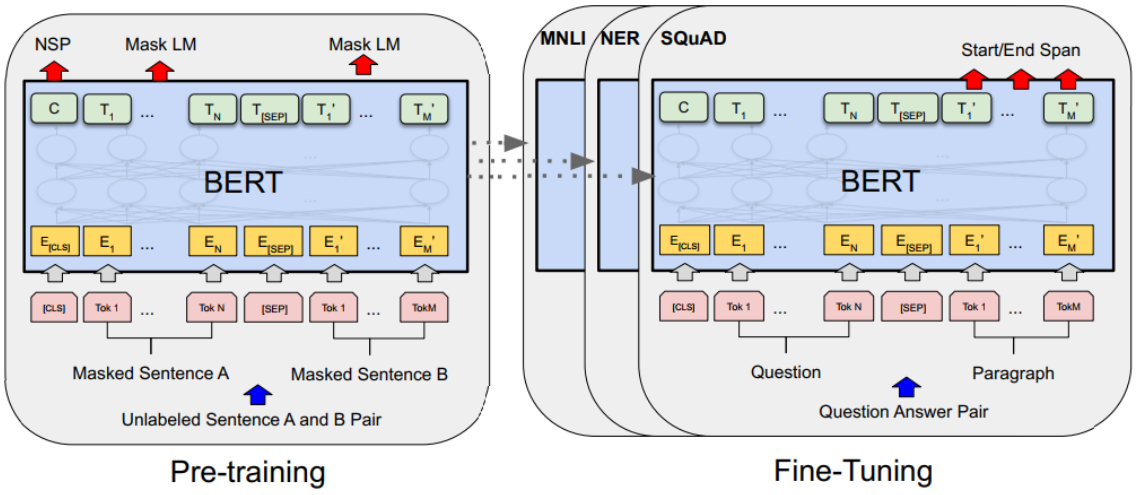


Figure 2. Pre-training and fine-tuning of BERT [1]

For each input sequence, the input embeddings (E_i in Figure 2) are constructed as shown in Figure 3 by summing up token embeddings, segment embeddings and position embeddings [1]. Token embeddings are simply the embeddings of each input token. Segment embeddings indicate which of the two sentences the token belongs to. Position embeddings encode absolute position of the token from 1 to maximum sequence length and show how a token in a given position attends to tokens in other positions [8]. A limitation of BERT with the type of inputs used in this thesis is that it does not allow encoding sets of input tokens as having the same position in the sequence. It can be seen as a limitation because it was not clear if services provided to the patient on the same date appeared on the invoice exactly in the same order as they were provided or not. It is also possible that in case of some services, the precise order is not important (e.g. if several lab tests are done on the same day, their order does not matter much).

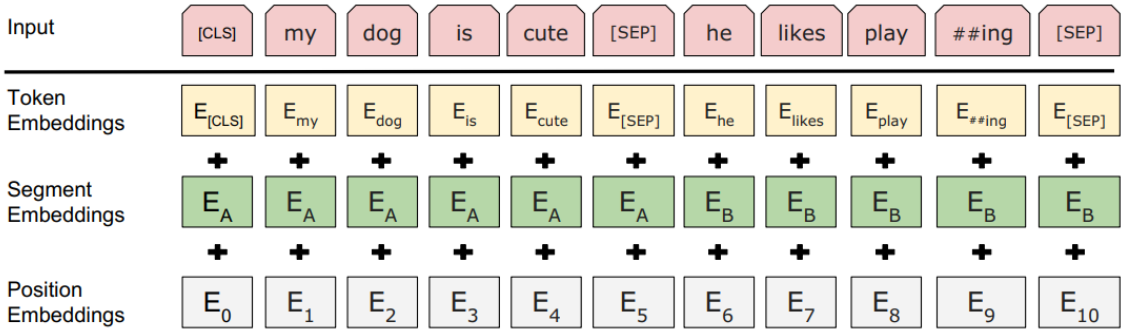


Figure 3. Representation of the input of BERT [1]

The tokenized inputs to BERT have a fixed maximum length [1]. As the authors note, a limitation in choosing the length is that time complexity of the attention mechanism in BERT is quadratic to the length of the input sequence, which makes using longer sentences disproportionately expensive. Their approach was to first pre-train BERT with sentence length 128 for 90% of the time and then continue training with sentence length 512. In this thesis, sentence (treatment trajectory) length used was 128. If a sentence is shorter than the maximum length, PAD tokens are added to the end, to ensure that all input sequences have the same length. This token, along with special tokens CLS and SEP, are ignored by the model during training [*Ibid.*]. Another special token used in BERT is UNK, which stands for out-of-vocabulary words.

In the BERT model developed in [1], words in the input sequence are split into subwords before feeding them to the model, using WordPiece tokenizer [14]. For example, in Figure 3, '*playing*' has been split into *'play'* and *'##ing'*. The advantage of using subwords in NLP is that it improves handling of rare words [*Ibid*.], but also helps the model to recognize that words having same roots or same endings are similar. However, in this thesis subword segmentation was not used. Splitting service codes into 'subwords' would not have made sense, because the codes, despite being treated as strings, were mostly numeric (a few also contained some letter) and unlike subwords in NLP, their subsequences did not infer similarity between the services.

2.3 Related works

Since data used in this thesis were patient treatment trajectories extracted from medical invoices, it is interesting to point out some examples of previous research where BERT (or its derivations) have been used on other than natural language data.

The most relevant example is BEHRT [4] – a BERT model for electronic health records (EHR), developed by Li *et al.* using data of 1.6 million patients. As opposed to this thesis, the inputs to BEHRT are sequences of diagnosis codes (301 in total) and corresponding sequences of ages of the patient, instead of codes of medical services. In BEHRT, diagnosis codes are acting as words and visits as sentences. Ages were added since they are a key risk factor in most diseases, and were also meant to give an indication of time intervals between visits. A difference from BERT is that instead of using just one sentence pair (i.e. two visits) as input, all visits of a patient, separated by SEP token, are concatenated into a single input sequence. The input representation was also modified compared to BERT (see Figure 3), by adding age embeddings, having segment embeddings (E_A and E_B in Figure 3) alternate between visits, and position embeddings being relative instead of absolute (same for the same visit, and increasing with successive visits). Similar to BERT, the final input representation was obtained by summing all the token (i.e. diagnosis), position, segment and age embeddings. The model was then fine-tuned to predict diseases in the next visit, and in the next 6 and 12 months.

Another derivation of BERT applied to EHR is G-BERT [5], developed by Shang *et al.*, which combines BERT with graph neural networks (GNNs). G-BERT can be used for representing the hierarchical structure of medical codes ontology and for recommending medication. Contrary to BEHRT [4], where inputs are diagnosis codes of successive visits, inputs to pre-training G-BERT are diagnosis and medications codes of a single visit, while successive visits are only accounted for in fine-tuning. Also, G-BERT does not have position embeddings (the authors justify their removal with medical codes within a visit not having any specific order). The first step in training G-BERT is learning two ontology embeddings – for diagnosis codes and for medication codes – using GNNs. Both are then used as input to BERT to obtain visit embeddings. The pre-training tasks in G-BERT are self-prediction (where similarly to MLM in BERT, masked codes are predicted based on other codes within a visit) and dual-prediction (predicting medications of the visit based on diagnosis of the visit and *vice versa*). G-BERT is then fine-tuned for medication recommendation. For fine-tuning, the means of diagnosis embeddings and medications embeddings of all previous visits, as well the diagnosis embedding of the current visit are concatenated. The result of this concatenation is then used as input to a classification layer to predict medications of the current visit – that is, to recommend medications, given the previous history and the current diagnosis.

Two other interesting examples of applying BERT to non-natural language inputs can be found from the field of bioinformatics. The first is DNABERT, developed by Ji *et al.* [2] for modelling the 'language' of human DNA. The inputs to DNABERT are *k*-mers (nucleotide sequences of length *k*). The authors trained a separate model for each value of *k* in range 3–6.

The architecture of the models was the same as $BERT_{BASE}$ proposed in the original BERT article [1]. However, similar to this thesis, the inputs were single sentences (not sentence pairs), and sequences longer than the maximum length (512) were split into pieces. During pre-training, the model learnt general understanding of the DNA 'language' through self-supervision. Then it was fine-tuned for three genomics tasks (predicting promoters, splice sites and transcription factor binding sites), where its performance surpassed previous models.

Another example is a paper by Min *et al.* [3], where Transformers and RNN were used for modelling proteins, with sequences of amino acids acting as 'sentences'. The pre-training of their models was otherwise similar to BERT (MLM with masking 15% of the amino acids), except that the NSP task was replaced by the task of predicting if the input protein pair belongs to the same protein family. However, besides fine-tuning the same Transformers model further, they also used the pre-trained embeddings as input to BiRNN (bidirectional recurrent neural network) and found that the latter performed better in 6 out of 7 downstream classification and regression tasks. The authors argued that an advantage of using RNN (Recurrent Neural Networks) instead of Transformers for modelling proteins was that while Transformers are good at learning long-distance relationships, RNN is better at learning local contexts, which are especially important in the structure of proteins. Also, they argued that Transformers cannot handle well inputs exceeding 512 tokens, while protein sequences are often longer.

3. Data

The datasets used for this thesis contained details of invoices submitted by medical institutions for reimbursement to the Estonian Health Insurance Fund (Haigekassa) in 2010-2019. The data was held in a PostgreSQL database, from where mainly two data tables were used. The first one (Dataset 1) contained billing data on all 72.6 million invoices, and the second one (Dataset 2) contained further details on 108,961 invoices randomly selected from Dataset 1. Dataset 1, except data on invoices also included in Dataset 2, was used for building patient treatment trajectories which were then used as input for training word2vec models and pre-training BERT models. Dataset 2 was used for building classification models on top of word2vec embeddings, and for fine-tuning BERT models for classification. The data was analysed in a secure and monitored environment according to the rules and restrictions posed by the data owner. In particular, all identifiers were pseudonymised (the data did not include patient ids).

3.1 Pre-processing

3.1.1 Treatment trajectories (Dataset 1)

Dataset 1 contained 293.8 million rows. Each row represented a service provided by a medical institution, with `bill_id` field indicating to which invoice the service in that row pertained. For extracting the treatment trajectories, first `bill_id` field was used to identify rows pertaining to the same invoice, which were then sorted by `service_date` and `row_number` fields. Finally, the values in `service_code` column for each invoice were extracted to form a temporally ordered sequence of service codes to be used as patient treatment trajectory represented by that invoice. As a result, each row (data point) in Dataset 1 contained a treatment trajectory formatted as a sentence, for example, `'3004 7004 66612.'` (corresponding to '*Repeated visit to specialist doctor' → 'Taking a biopsy (except during surgery)' → 'Determination of pathogen type or group with PCR method*').

Trajectories consisting of only one service code (31.3 million trajectories, or 43.1% of the data) were dropped, since they would not have been useful for training the models. For example, in the mask prediction task in BERT it would have resulted in this single code being masked, leaving no other codes for the model to base its prediction upon. Leaving out only invoices with a single service code was actually a conservative choice, which helped to limit selection bias (which arises from data not meeting some predefined criteria being left out from training a model [5]). For example, the authors of BEHRT [4] left out all patients whose trajectories contained less than five visits, where each visit could contain multiple diagnosis codes. It must be noted that while each invoice in this thesis included services provided to the same patient, patient ids were not given. Therefore, unlike BEHRT [4], different visits of the same patient (i.e. services provided to the same patient but included in different invoices) could not be concatenated in this thesis for making short trajectories longer.

The data was then shuffled and split into train, test and evaluation set and a set of long sentences. The latter were later split into shorter sentences and divided between train, evaluation and test set in python. This was done in a way that ensured that parts of the same sentence were either only in train, evaluation or test set. The reason for splitting long sentences was that the maximum sentence length used in BERT models in this thesis was 128 (125 service codes, plus special tokens CLS and SEP, and '.'). Long treatment trajectories of more than 125 service codes (30,556 trajectories) were therefore split into trajectories of maximum length 125 (resulting in 73,340 trajectories). Initially, using maximum sentence length 256 was also

considered, but was disregarded, because some initial trials of pre-training showed that this considerably increased training time.

Before splitting into train, test and evaluation set (and the set of long sentences), invoices also included in Dataset 2 were excluded, since the aim was to only use them in training classification models. Table 1 illustrates how the pre-processed Dataset 1 looked like before performing the train/test/eval split. In Table 1, `bill_id` column has been anonymised. The number of services (rows) on invoice in `rows` column was used for statistical purposes. The last column shows which split the row was assigned to.

Table 1. Example rows in pre-processed Dataset 1

	bill_id	service_codes	rows	data_128
1	XXXXXXX15	9064 9001 .	2	train
2	XXXXXXX20	3002 3004 7975 .	3	train
3	XXXXXXX22	66707 3004 66107 66707 66707 …	24	eval
4	XXXXXXX23	9001 9002 .	2	train

The pre-processing described above (except splitting long trajectories) was done with SQL queries directly in the server holding the database. Performing them locally would not have been feasible, considering the size of the dataset. Also, avoiding downloading all data was preferrable for data security purposes.

The sizes of train, test and evaluation sets are given in Table 2. In addition to the training set of 39 million samples, a reduced version of it was prepared by randomly sampling 1 million trajectories from it. This reduced version was used

Table 2. Sizes of train, test and evaluation sets

	Train	Test	Evaluation	Total
Dataset 1	39,294,731	1,000,000	1,000,000	41,294,731
Dataset 2	45,000	10,000	6,991	61,991

While obviously the whole training set of 39 million samples was used for training, also a reduced version of it was prepared by randomly sampling 1 million trajectories from it. This reduced training set was used for measuring performance of an already trained model also on training set (to compare it with test set). The reduced version was assumed to represent the large training set well enough for this purpose. Its purpose was to speed up testing the models (which was essential, given that testing the largest pre-trained BERT model even on the 1 million dataset took 23 hours).

After downloading the trajectories from the database, they were saved as .txt file with one trajectory on each line. For word2vec, no further pre-processing was needed. For BERT, the input trajectories were tokenized using WordLevel tokenizer from Hugging Face Tokenizers library[4], which (as opposed to the default WordPiece tokenizer) does not use subword segmentation. The tokenizer added CLS and SEP tokens to the start and the end of each trajectory (for example, `['[CLS]', '3002', '3004', '7907', '7941', '.', '[SEP]']`). Also, it replaced service codes appearing less than five times (207 codes) with UNK tokens. Then it encoded all tokens with their vocabulary ids (for example, the same trajectory was encoded as `[1, 2767, 1689, 1450, 2253, 2179, 2]`). The vocabulary

[4] https://github.com/huggingface/tokenizers

size (incl. service codes that appeared at least five times, special tokens CLS, SEP, PAD, MASK and UNK, and a token for '.') was 3,440. All trajectories were padded to length 128 by adding PAD tokens to the end, and then converted into Pytorch tensors.

Based on the input ids, also other inputs needed for BERT – token type ids, attention mask and special tokens mask – were generated. All these were also tensors of length 128. Token type ids indicate whether each token belongs to the first or second sentence in case sentence pairs are used as input. Since in this thesis the inputs were single sentences, token type ids were all zeros. The values in attention mask and special tokens mask indicate which tokens are paddings or special tokens, respectively, that should be ignored by the model during training (with '0' standing for padding in attention mask and '1' for special tokens in special tokens mask). For example, the final tokenized input of the same treatment trajectory as above was:

```
{'input_ids':        [1, 2767, 1689, 1450, 2253, 2179, 2, 0, ... 0],
 'token_type_ids':   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0],
 'attention_mask':   [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, ... 0],
 'special_tokens_mask': [1, 0, 0, 0, 0, 0, 1, 1, 1, 1, ... 1]}
```

The masking in pre-training was done using the internal implementation of BERT, by using the default option of masking 15% of the tokens. For mask prediction on test set, the masks were instead added before feeding inputs to the model, by masking one random token in each trajectory. This was needed for two reasons. Firstly, it was needed to save the locations of the masks so that exactly the same masked tokens could be predicted both with BERT and word2vec to ensure comparability of their performance. Secondly, the internal implementation of masking 15% of tokens resulted in longer trajectories having more than one mask and some short trajectories not having any masks at all. However, it seemed not make sense to have trajectories with no masks in the test set. Despite different approaches to masking, the proportion of service codes masked in test set (14%) was actually quite similar to the one in training set (15%).

3.1.2 Predicted features (Dataset 2)

Dataset 2 contained further details on 108,961 invoices randomly selected from Dataset 1, which were used for constructing features to be predicted in the classification models. Each row in this dataset represented an invoice and could be linked to the rows in Dataset 1 through the `bill_id` field. When pre-processing Dataset 2, invoices that were dropped from Dataset 1 due to having only one service code (43.2% of the invoices) were also dropped from Dataset 2. The rows corresponding to long invoices which were split into multiple parts (due to having more than 125 service codes) were duplicated in Dataset 2 so that there was a row corresponding to each of the parts. After those operations, the final number of rows in Dataset 2 was 61,991.

When choosing the columns to be used as features in classification models, the ones where classes were extremely unbalanced or numeric were disregarded. Among the features used, the binary feature `emergency_bill`, indicating whether the invoice was for emergency services or not, was the only one used as is. In other features, some related classes were merged as described below to have the classes a bit less unbalanced.

The column `treatment_type` contained eight categories which were merged into five by removing the breakdown into overtime work and normal working hours. The column `discharge_status` contained 16 statuses of releasing the patient from a medical institution. Some related categories in this column were merged as shown in Appendix II in order to have fewer very small categories.

The column `main_icd10_diagnosis` contained 37,928 unique diagnosis codes. Since it would have been unfeasible to build classification models with that many classes in a dataset of size 61,991, diagnoses codes were instead aggregated in two ways into broader categories. Aggregation was based on the hierarchical levels present in ICD10 classification available in the database and resulted in two features, `diag` (18 classes, see Appendix III) and `diag2` (174 classes, see Appendix IV). In the higher level of aggregation (`diag`), codes were aggregated based on their starting letters, with related categories A–B, C–D, S–T and V–Y combined, as is also done in ICD10 [15] chapters. For example, category A–B corresponded to Chapter I (*'Certain infectious and parasitic diseases'*) of the ICD10 classification. Categories that did not appear in data[5] were left out. The lower level of aggregation (`diag2`) was based on ranges provided in the classification (for example, A00–A09, *'Intestinal infectious diseases'*). In case of overlapping ranges, i.e. when a wider range was further broken down into narrower ranges, only the narrower range was kept. Categories not appearing in data (44 categories) were left out and categories with less than 16 samples (47 categories) were combined into category '*Other*'.

Finally, the column `tto_registry_code`, containing registry code of the medical institution submitting the invoice, was used to construct two features: `tto_type` and `largest4`. For `tto_type`, a mapping available in the database was used to map 784 registry codes into five types of service providers (see Appendix V). For `largest4`, four largest service providers (Tartu University Hospital, East Tallinn Central Hospital, North Estonia Medical Centre and West Tallinn Central Hospital) were determined by their number of invoices in Dataset 2, and the remaining ones were assigned to the category '*Other'*. The purpose of this feature was to see if trajectories in the largest hospitals and other service providers are distinguishable (i.e. if it is able to predict where the invoice originated from), which would have meant that also medical practices between the four largest and other service providers must differ.

An overview of all features from Dataset 2 after pre-processing that were predicted in the classification models is given in Table 3. Class distributions of the final features are shown on the histograms in Section 3.2.2.

Table 3. Features predicted in the classification models

Predicted feature	No. of classes	Description
`diag`	18	Main diagnosis of the patient (higher level of aggregation of ICD10 codes)
`diag2`	174	Main diagnosis of the patient (lower level of aggregation of ICD10 codes)
`emergency_bill`	2	Binary feature indicating whether or not the invoice was for emergency services
`discharge_status`	7	Statuses of releasing the patient from the medical institution
`treatment_type`	5	Treatment types
`tto_type`	5	Types of service providers
`largest4`	5	Largest four and other service providers

Finally, the data was split into train, test and evaluation sets of sizes given in Table 2.

[5] U (*'Codes for special purposes'*), V–Y (*'External causes of morbidity and mortality'*) and '?' (*'Missing or erroneous diagnosis code'*).

3.2. Descriptive statistics

3.2.1. Descriptive statistics of treatment trajectories

Services. Total number of service codes appearing in the treatment trajectories was 3,641, which is considerably smaller than the number of words in a language. The most frequent services (see Appendix VI) were initial or repeated visit to family doctor (5.12% and 6.57% of the services) or to specialist doctor (5.96% and 4.61% of the services), followed by some common lab tests. In that sense, treatment trajectories were indeed similar to natural language, where also some words (e.g. articles and prepositions) are highly frequent. At the same time, as many as 86 services only appeared once.

Service subtypes. Since some of the performance metrics in this thesis were based on subtypes (see Section 4.3), it might also be relevant to look at the distribution of service subtypes (see Appendix VII). Overall, the 3,641 services were divided between 45 subtypes. The distribution was far from uniform, though, with the largest subtype *('Surgeries')* containing 1,011 services and the two smallest subtypes containing only one service. Appendix VII also shows the proportion of service codes in data belonging to each subtype. We can see that *'Lab tests'* made up as much as 42.5% of all services provided, with *'Treatment by family doctor'* (24.0%) being next. However, the services in great majority of subtypes (35) made up less than 1% of all services provided.

Lengths of treatment trajectories. As mentioned in Section 3.1.1, as much as 43.1% of the invoices only included a single service. On the other hand, the longest invoice contained 5,304 services. In that regard, the trajectories clearly differed from natural language sentences, which are never that long and only rarely consist of a single word. The distribution of treatment trajectories of length 2-125 in pre-processed data, where invoices with a single row were left out and invoices with more than 125 rows were split into parts, is shown in Figure 4[6]. We can see that very short trajectories were dominating in the data. The mean length of trajectories in pre-processed data was 6.18.

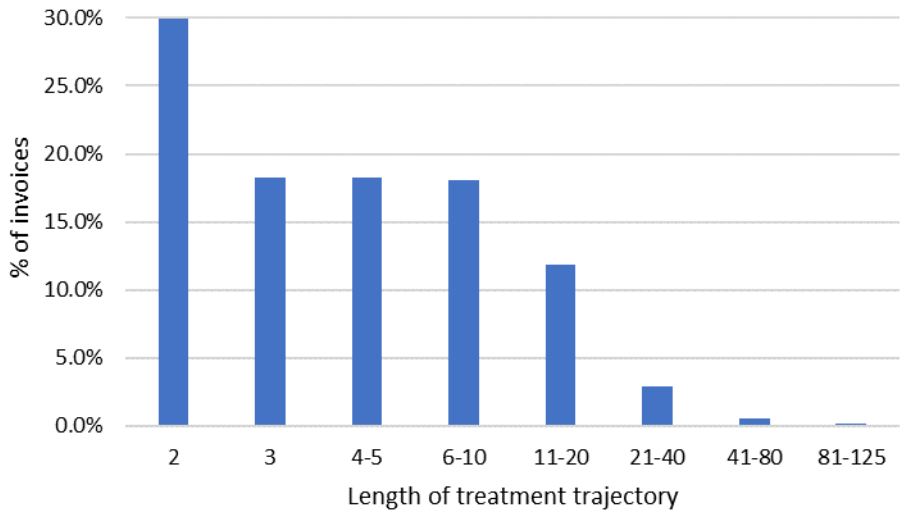


Figure 4. Distribution of treatment trajectory lengths

[6] Trajectory lengths reported in Figure 4 do not include full stop, which was added to the end of each trajectory to make it similar to a natural language sentence.

Repetitions. Another aspect in which patient treatment trajectories differed from natural language sentences was that they repeated a lot (see Table 4). While there were 41.3 million invoices, the number of unique trajectories was only 11.6 million. From Table 4, we can see that although 90.4% of the unique trajectories were non-repeating (i.e. appeared just once), they only made up 25.40% of the data. On the other hand, the repeating trajectories sometimes repeated a very high number of times. For example, while only 0.0002% of unique trajectories repeated more than 100,000 times, they made up as much as 12.3% of all trajectories (invoices). Although there are also certain short sentences (like greetings etc.) that repeat in natural language, they are by far less numerous and their repetitions are not that frequent.

Table 4. Repetitions of treatment trajectories

Times appeared	% of all trajectories	% of unique trajectories
1	25.40%	90.3631%
2-10	7.7%	8.3243%
11-100	8.7%	1.1314%
101-500	7.9%	0.1356%
501-5000	15.4%	0.0401%
5001-25,000	12.5%	0.0045%
25,001-100,000	10.3%	0.0008%
over 100,000	12.3%	0.0002%

Ten treatment trajectories that repeated the most are given in Appendix VIII. All ten featured visiting or consulting a family or a specialist doctor or a nurse as one of the services, with some also including another service (e.g. examination of eyes or prenatal ultrasound examination). The most frequent trajectory (1,342,397 repetitions) was `'9001 9002'` (‘*Initial visit to family doctor’* → *‘Repeated visit to family doctor*’). Examining longer trajectories revealed that some of them featured another type of repetition – the same service (e.g. a lab test) repeating numerous times. For example, the most frequent trajectories of length 8 and 9 (with frequencies of 7,708 and 7,830, respectively) consisted of *’Home nursing service’* repeating respectively 8 and 9 times. Still, usually the repeating code only formed a small subsequence in the trajectory.

Services provided on the same date. Initially, instead of predicting a single service, predicting the set of services provided on the same date was considered, since it would have allowed to ignore the order in which services offered on the same date appeared on an invoice, which seemed likely to be arbitrary in some cases. However, this idea was disregarded for the following reasons. First, 75.1% of all invoices included services provided on a single date, resulting in trajectory of length 1. Similarly to leaving out invoices with a single service from the final analysis, such invoices would have needed to be removed, which would have meant leaving out too large proportion of the data. Secondly, the number of unique sets of services provided in a single date was 6.6 million (or 4.9 million, if sets containing same services offered different number of times were considered as the same set). This would have meant vocabulary size of 6.6 million (or 4.9 million) in BERT models, while typically vocabulary size of at most 50,000 is used. Using that much larger vocabulary did not seem feasible, since it would have considerably increased the already very long training time. A possibility could have been to use vocabulary size of 50,000 to encode only the 50,000 most frequent service sets as distinct tokens and treat all others as UNKs. However, as seen from Figure 5, this would have led to 13% of the data consisting of UNKs.

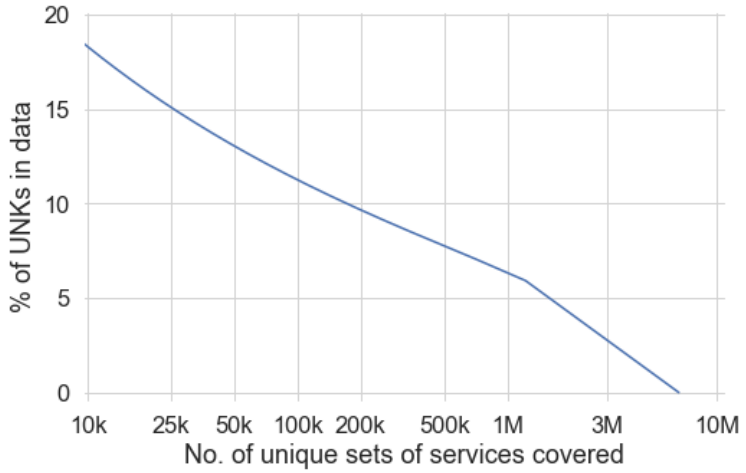


Figure 5. Percentage of UNKs in case of different sizes of vocabulary of service sets

The descriptive statistics given above showed that the properties of treatment trajectories differed from natural language sentences in a number of aspects. Therefore, before building the models, it was not at all certain how well BERT would perform on this type of data.

3.2.2 Descriptive statistics of predicted features

In this section, the histograms of the predicted features (see Table 3) are given, except for the binary feature `emergency_bill`. In this feature, 12.4% of the values were TRUE, indicating that the invoice was for emergency services.

The distribution of `diag` (diagnoses with 18 classes) is shown in Figure 6. In the figure, starting letters of ICD10 codes are given in brackets. The largest class contained 16.7%, and the smallest one 0.1% of the trajectories.

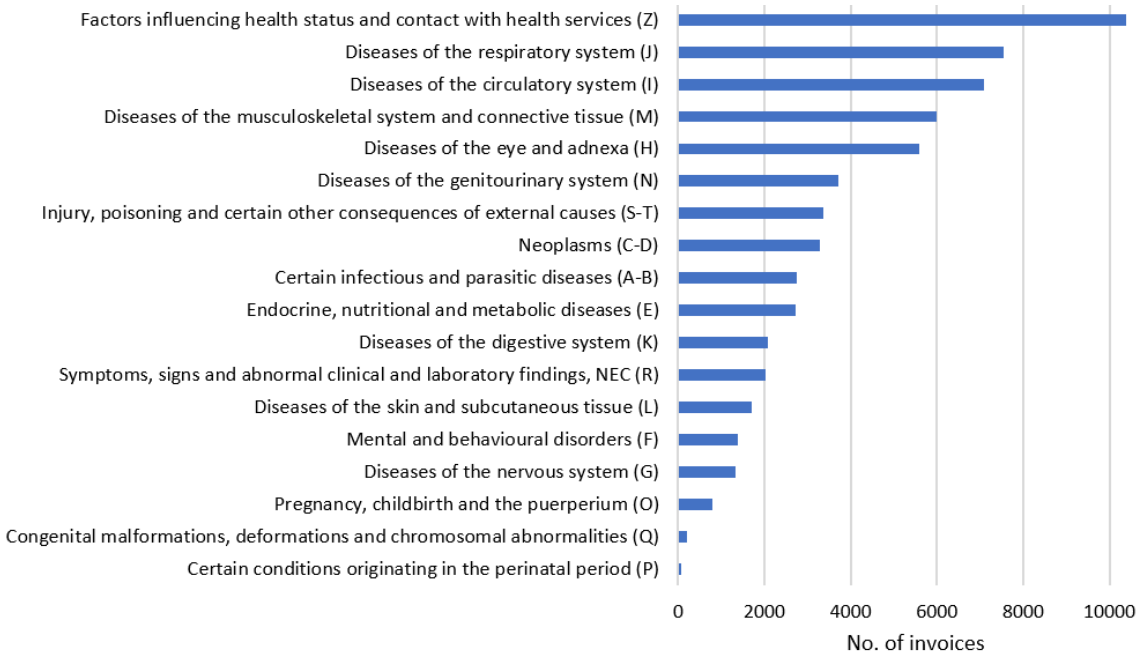


Figure 6. Distribution of diagnoses (18 categories)

Ten most frequent categories in `diag2` (diagnoses with 174 classes) are shown in Figure 7. Similar to `diag`, its distribution was highly uneven, with the largest class containing 9.4% and the smallest classes only 0.03% of the samples.

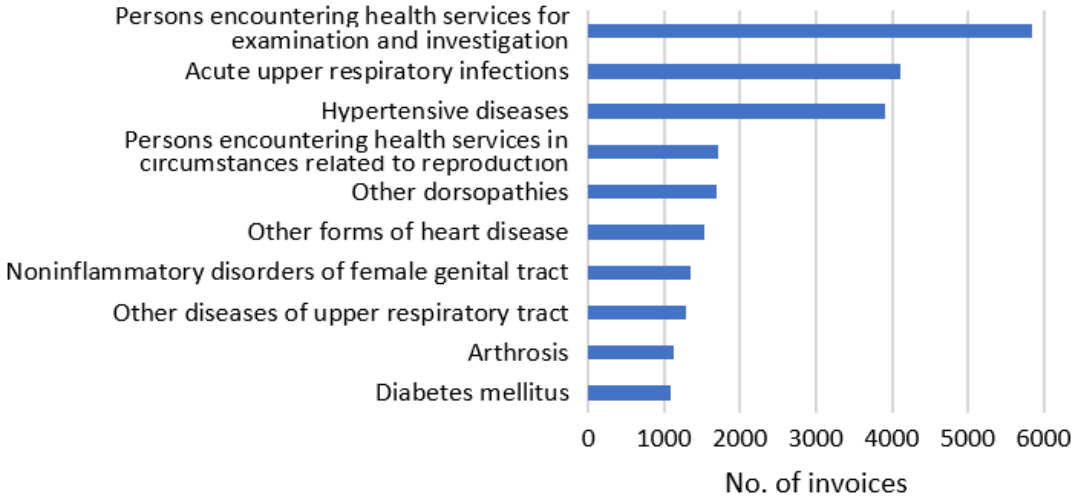


Figure 7. Distribution of diagnoses (10 most frequent out of 174 categories)

Distribution of the feature `discharge_status` (see Figure 8) showed that as a result of the services covered by the invoice, 41.4% of the patients recovered and 0.2% deceased. In case of most of the others, treatment was set to continue.

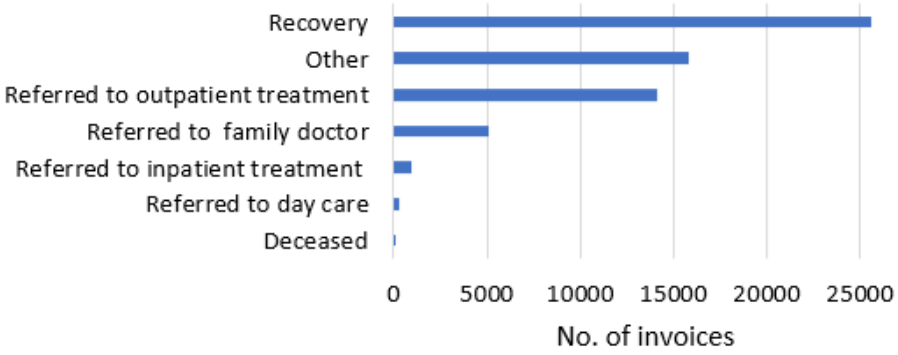


Figure 8. Distribution of discharge statuses

The two largest classes in `treatment_type` (see Figure 9), *'Outpatient treatment'* and *'Family doctor'*, contained respectively 45.6% and 46.5% of the samples, leaving in total only 7.9% of the samples to the three other classes combined.

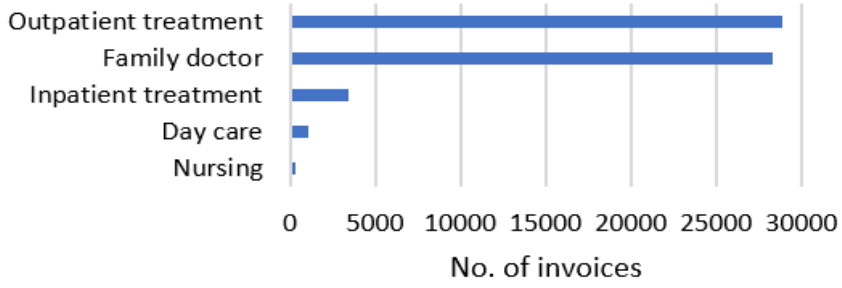


Figure 9. Distribution of treatment types

The largest and smallest classes in `tto_type` (types of service providers) were *'Chosen partner'* (54.7% of data), and *'Local hospital'* (0.3% of data), as shown in Figure 10.

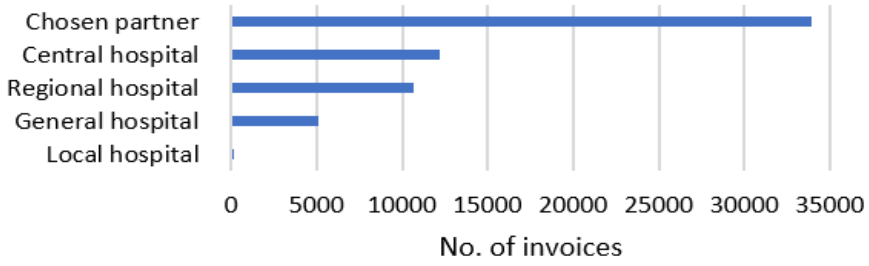


Figure 10. Distribution of types of service providers

Distribution of `largest4` is shown in Figure 11. Each of the four largest hospitals had submitted from 5.1% to 8.5% of all the invoices, while 72.1% had been submitted by other medical service providers.

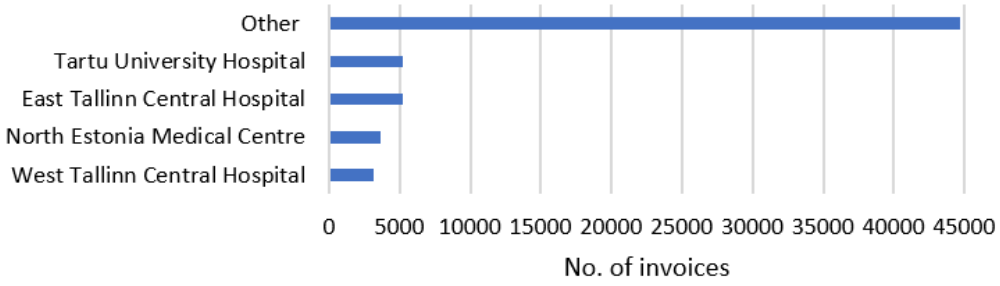


Figure 11. Distribution of four largest *vs.* other service providers

As seen from the figures in this section, even after somewhat improving the distributions by merging some related classes (as described in Section 3.1.2), the classes of all predicted features were still unbalanced.

4. Methodology

Models in this thesis were trained in two stages (see Figure 12) – training of the service code embeddings (stage 1) and classification (stage 2). In stage 1 (see Section 4.1), four BERT models of different dimensions were pre-trained on patient treatment trajectories. For comparison, word2vec models with different vector length and context window size were trained on the same inputs. In Figure 12, only the three best performing word2vec models are shown. In the names of word2vec models, `'w'` indicates window size and `'v'` vector length. For example, `W2V_w5_v400` was a word2vec model with window size 5 and vector length 400. The specifications of the pre-trained BERT models are described in Section 4.1.1 and the ones of word2vec in Section 4.1.2.

The performance of both types of models was tested on mask prediction task. In case of BERT this meant predicting the token that was masked out, given the rest of the trajectory. In case of word2vec, it meant predicting the token in the middle of a pre-defined context window, given the rest of the service codes in that window. Although 'mask' is a term specific to BERT, for terminological consistency purposes the term 'mask prediction' is also used for predicting the target word in word2vec in this thesis. The task itself in both cases was the same – predicting the unknown token. Measuring mask prediction performance of both models allowed to see if contextual embeddings of BERT are more effective in this task than the non-contextual ones of word2vec.

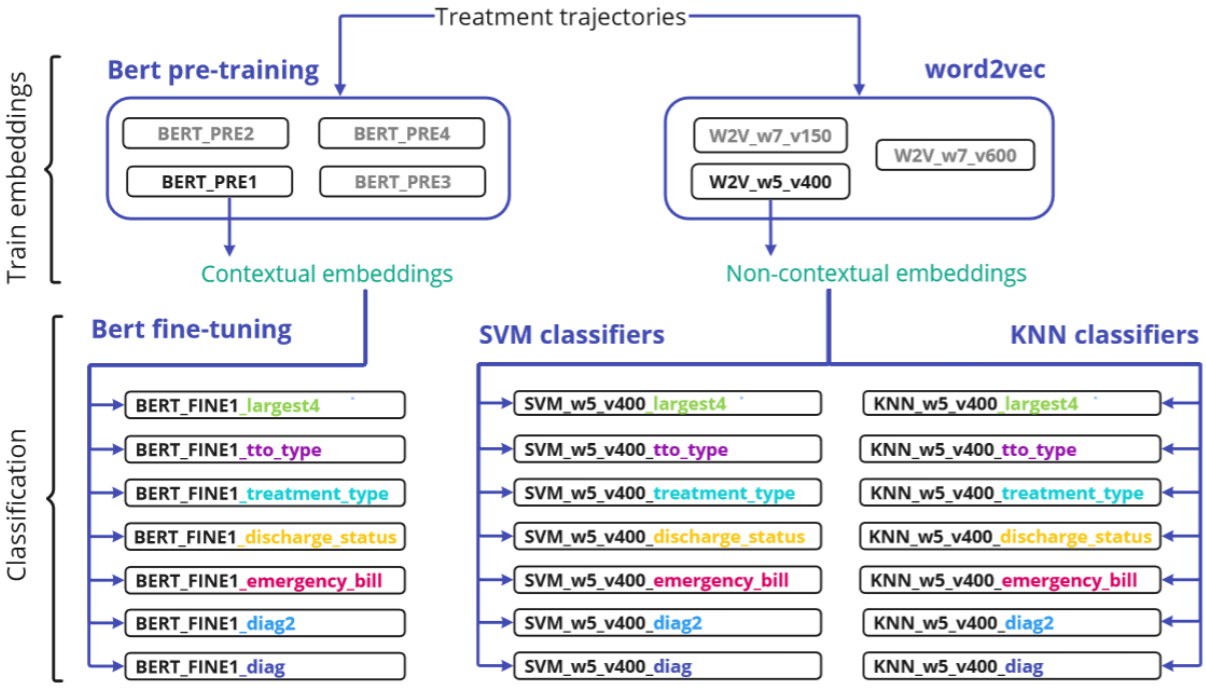


Figure 12. General set-up of building the models

In stage 2 (see Section 4.2), classification models were built for predicting the labels of each of the features listed in Table 3. This allowed to compare the usefulness of contextual and non-contextual embeddings trained in stage 1 in different downstream classification tasks. More specifically, in stage 2 all four pre-trained BERT models were fine-tuned for classification. For comparison, two common classifiers, KNN and SVM, were trained on top of the embeddings of three best performing word2vec models. Since a separate model was trained for each of the seven features (see Table 3), this meant fine-tuning in total 28 BERT models and training 21 KNN and 21 SVM classifiers. In Figure 12, classifiers built by fine-tuning only one of the pre-

trained BERT models and training KNN and SVM models on top of only one word2vec model are shown as an example.

In the names of the models built in stage 2, the last part (e.g. `largest4`) indicates the name of the predicted feature, as listed in Table 3. In the names of fine-tuned BERT models, the number indicates which pre-trained model was fine-tuned. For constructing the names of KNN and SVM models, `W2V` was replaced with `KNN` or `SVM` in the name of the word2vec model which embeddings they used. So, `BERT_FINE1_diag` denotes BERT model obtained by fine-tuning `BERT_PRE1`, and `KNN_w5_v400_diag` denotes KNN model built on top of `W2V_w5_v400`, where the task of both was classifying diagnosis. In the sections below, for referring to all models of certain type, the predicted feature is omitted. E.g. `BERT_FINE1` refers to all BERT models obtained by fine-tuning `BERT_PRE1`, and `KNN_w5_v400` to all KNN models built on top of `W2V_w5_v400`.

The performance metrics used in both stages are described in Section 4.3. Since the performance of several classification models in stage 2 was rather similar, McNemar statistical test[7] was used for checking the performance of which models did not statistically differ. McNemar test is a non-parametric test, which means that it does not make any assumptions about the underlying data distribution. More specifically, it is a two-sided approximated test which tells if the proportion of classification errors made by two models tested on the same test set are statistically significantly different or not, based on comparison of the predicted labels. Here, null hypothesis is that the performance of the models does not differ, and alternative hypothesis is that it differs. Since this test allows comparing only two models at a time, it was done for each pair of models for each classification task. If the test showed that accuracy of the model that performed best in a given classification task was not statistically different from some other models, all such models were regarded as the set of best models for that task. Although in general it is useful to also fix minimal effect size before doing a statistical test, in this case it was not done. Effect size is especially important in case of larger datasets (since in a large dataset, even very small differences tend to be statistically significant). In this thesis, however, the statistical test was only used in classification stage where test set was not very large (10,000 samples).

All models were trained in the computing cluster of University of Tartu's HPC Centre[8].

4.1. Embeddings for service codes

4.1.1 Pre-trained BERT models

In this thesis, four BERT models of different sizes were pre-trained: `BERT_PRE1`, `BERT_PRE2`, `BERT_PRE3` and `BERT_PRE4`, from largest to smallest. The models only differed in the number of layers and attention heads (see Appendix IX). The parameters whose values differed (see Appendix IX) were the same ones that were fine-tuned in the BEHRT model [4], where also a model with smaller dimensions than proposed in the original BERT paper [1] was found to be the best. The reason for pre-training models with different sizes was that it was thought that perhaps treatment trajectories are less complex than natural language, and therefore easier for the models to learn. Initially, just the two larger models (`BERT_PRE1` and `BERT_PRE2`) were pre-trained, but since the performance of the smaller one was somewhat better, two even smaller models were pre-trained as well.

[7] https://www.statsmodels.org/dev/generated/statsmodels.stats.contingency_tables.mcnemar.html
[8] https://hpc.ut.ee/

As regards training parameters, mostly the default ones were applied, since due to pre-training a BERT model being a lengthy process, no parameter tuning was planned. The parameters that differed from the default ones (except the ones related to saving checkpoints, logging and evaluation) are set out in Appendix IX. As for the learning rate, the one also applied by the authors of BERT model (1e-4) [1] was used instead of the default one (5e-5).

As regards implementation, the models were pre-trained using Hugging Face Transformers library[1]. The type of pre-trained models was `BERTForMaskedLM`[9]. A custom Pytorch dataset was used for loading the inputs (39.3 million tokenized treatment trajectories), which were then collated into batches with `DataCollatorForLanguageModeling`[10]. The input parameters were specified in `TrainingArguments`[11] and `BERTConfig`[12] objects. Finally, a `Trainer`[13] object was used for training the models.

All models were trained for 4 epochs using 7 GPUs. Pre-training the largest BERT model took 4 days and 12 hours. The evolution of losses in pre-training (see Figure 13) showed that in all models, evaluation loss remained smaller than train loss, which was unexpected. At the same time, the decrease in both train and evaluation loss showed that the models were still learning something. It appeared that most of the training happened during the first 0.5 epochs. Since losses were still very slowly decreasing at the end of epoch 4, the two larger models (`BERT_PRE1` and `BERT_PRE2`) were further pre-trained for 2 more epochs from the last checkpoint to see if their losses would go further down. However, the differences were negligible, and thus those versions were disregarded.

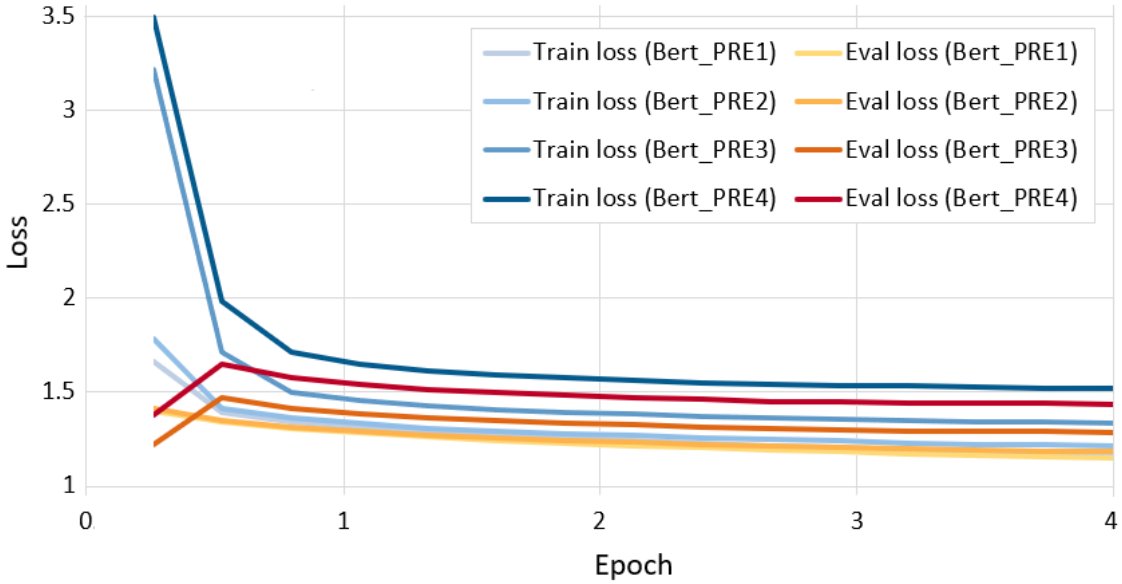


Figure 13. Training and evaluation loss of pre-training BERT models

The models were tested on 1 million treatment trajectories in the test set (and for comparison, on the reduced version of training set containing also 1 million samples), with one randomly chosen token in each trajectory being masked out. The mask prediction task then consisted in predicting the masked token. It must be noted that in case of BERT, it is rather the performance on downstream tasks (classification, in this thesis) than on mask prediction what is usually

[9] https://huggingface.co/transformers/v4.7.0/model_doc/bert.html#bertformaskedlm
[10] https://huggingface.co/transformers/v4.7.0/main_classes/data_collator.html#datacollatorforlanguagemodeling
[11] https://huggingface.co/transformers/v4.7.0/main_classes/trainer.html#trainingarguments
[12] https://huggingface.co/transformers/v4.7.0/model_doc/bert.html#bertconfig
[13] https://huggingface.co/transformers/v4.7.0/main_classes/trainer.html

considered to matter. For example, neither in the article where BERT [1] nor word2vec [6] was proposed, the performance on mask (target word) prediction was not even reported (the authors only reported performance on downstream tasks). However, measuring it in this thesis was essential, since it allowed to compare the pre-trained contextual embeddings of BERT with the non-contextual ones of word2vec prior to moving on to the next stage of building classification models.

As regards mask prediction, it must also be noted that in language modelling, if the aim is predicting labels for single tokens, e.g. for part-of-speech tagging, normally BERT is fine-tuned for token prediction task (e.g. using `BERTForTokenClassification`[14] in Transformers library) instead of using pre-trained BERT directly for predicting tokens. At the same time, the task of predicting labels for known tokens differs from mask prediction, since the masked token itself is unknown.

The results of pre-trained BERT models are provided in Section 5.1.1 and are, together with results of word2vec models, further analysed in Section 5.1.3. The further analysis includes a small case study performed on one of the BERT models (`BERT_PRE2`), where Principal Component Analysis (PCA) was applied on the pre-trained embeddings of 10 selected medical services. PCA is a dimensionality reduction technique where a chosen number of new features (principal components) are constructed as a linear combination of existing features in such way that each principal component maximizes the variance in the direction orthogonal to other principal components [16].

The aim of applying PCA was to better understand whether the pre-trained models, given their poor performance, had at least learnt something useful. By running PCA with two principal components, the data was transformed into 2D space, which allowed it to be plotted. The plot was expected to show, on the one hand, if the service embeddings were indeed contextual, i.e. if the model gave the same service a different embedding depending on its context. On the other hand, it was expected to show if embeddings of same services cluster together and what the clusters look like. The PCA was performed with `PCA` library[15] of scikit-learn.

4.1.2. Word2vec models

Initial experiments with smaller datasets showed that parameters that influenced word2vec performance the most were window size (`window`), vector length (`vector_size`) and whether the model was a skipgram or CBOW model (`sg`). Since skipgram models in general performed worse in the initial experiments, these were not used. Also, `min_count` 5 was used to avoid training embeddings for service codes that appeared in training set less than 5 times. For determining the best input parameters, models were then trained using the whole 39 million training set with all possible combinations of window sizes 3, 5, 7, 10 and 20 and vector lengths 150, 250, 400 and 600 (i.e. 20 models). All models were trained for 20 epochs with batch size 5,000 using 3 CPUs. In order to choose the best model, the models and training progress were saved after each epoch during training by using a callback. Training time of each model was between 1.5 to 3 hours, but they were trained in parallel in order to save time.

Performance of the 20 word2vec models was first tested on evaluation set. Based on results, three models were selected to go forward with in the classification stage. The performance of those three models was also reported on test set, and for comparison, on the reduced version of training set containing 1 million samples.

[14] https://huggingface.co/transformers/v4.7.0/model_doc/bert.html#bertformaskedlm
[15] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

When comparing word2vec to pre-trained BERT, we can on the one hand say that the vocabulary size of both types of models was the same (3,440). Also, although the embeddings learnt by word2vec were static (a fixed embedding for each medical service) and the ones learnt by BERT were contextual (embedding for each medical service depended also on other medical services appearing in the same trajectory), the lengths of the embeddings were actually in a similar range. Namely, the lengths of different word2vec embeddings were 150, 250, 400 and 600, and the lengths of embeddings that could be extracted from the four BERT models were 96, 192, 384 and 768. For BERT, embeddings can be extracted from the last hidden layer of the prediction output, and their length corresponds to the hidden size (see Appendix IX).

4.2. Classification models

4.2.1 Fine-tuned BERT models

Each of the four pre-trained BERT models was fine-tuned for each of the seven classification tasks. Since fine-tuning essentially builds upon the pre-trained model, the training arguments used in fine-tuning were the same as in pre-training (see Appendix IX), except the number of epochs. For all models, `BERTForSequenceClassification`[16] was used as model type. Similar to pre-training, a custom Pytorch dataset object was used for loading the tokenized input trajectories as well as labels, which were then collated into batches with `DataCollatorWithPadding`[17]. The data collator was different than in pre-training, because masking did not need to be applied. Also similar to pre-training, a `Trainer`[13] object was used for training the models.

All BERT models were fine-tuned for 5 epochs, using early stopping. The number of epochs was chosen based on initial experiments run with the largest pre-trained model (`BERT_PRE1`) on the classification task with the largest number of classes (`diag2`). The way early stopping was applied was that when training each of the 28 models, the current state of the model and its performance on training and evaluation set were saved as checkpoints four times per epoch. Then the version of the model that performed best on evaluation set was selected as the final version of the model from among the checkpoints. Finally, performance of the best model was measured on test set (and for comparison, on training set).

The number of epochs used for training the best-performing final version of each of fine-tuned model is given in Appendix X. As shown in Appendix X, the number of epochs ranged between 2 and 5, with training time being the shortest for the classification task with the smallest number of classes (`emergency_bill`, 2 classes) and in general longest for the one with the largest number of classes (`diag2`, 174 classes). Training time of the largest model (`BERT_FINE1_diag2`) was 8 hours on the main node of HPC.

4.2.2. KNN and SVM models

On top of the three best performing word2vec models, two types of classification models, KNN and SVM, were built for each of the seven classification tasks. KNN is a classifier that predicts the class of a test instance by taking a vote between its k≥1 neighbours (i.e. training instances that are closest to it) and predicting the majority class [16]. SVM is a classifier that finds a decision boundary between the classes in a way that maximizes its distance ('margin') from the two nearest training data points ('support vectors') of each class [*Ibid.*].

[16] https://huggingface.co/transformers/v4.7.0/model_doc/bert.html#bertforsequenceclassification
[17] https://huggingface.co/transformers/v4.7.0/main_classes/data_collator.html#datacollatorwithpadding

Those two models were chosen because this allowed to also compare how a simple non-linear model (KNN) behaves compared to a linear model (SVM) on classifying patient treatment trajectories. For comparison, the classification layer that is added to BERT for fine-tuning is also linear, and is followed by softmax function. For implementation, `KneighborsClassifier`[18] and `SGDCkassifier`[19] from scikit-learn library were used. According to the API, `SGDCkassifier` by default gives a linear SVM. Another classifier from the same library that was considered was `SVC` (Support Vector Classifier)[20], but it could not be used because it can handle only small datasets.

In order to build classifiers on top of word2vec, treatment trajectories were converted into feature vectors by averaging the word2vec vectors corresponding to all tokens in the trajectory. For example, for classifiers built on top of `W2V_w5_v400`, the trajectory `'9001 9002 .'` was converted into a feature vector of length 400 by averaging the vectors of the tokens `'9001'`, `'9002'` and `'.'` (each also of length 400) extracted from that model. Initially, using summing instead of averaging was also considered, but was disregarded since it tended to give slightly worse results. The vectors of treatment trajectories were normalized using `StandardScaler`[21] from scikit-learn library, which consisted in standardizing the features by subtracting the mean and dividing by standard deviation. It must be noted, though, that at least for KNN, standardizing gave only a minimal gain in performance (evaluation accuracy improved on average by 0.13%).

Since the training time of the classifiers was much shorter than fine-tuning BERT, some parameter tuning was applied on evaluation set. The fine-tuned parameter values of KNN and SVM models are given in Appendix XI for replicability. For KNN, the number of neighbours (`n_neighbours`) was tuned. For SVM, the parameters tuned related to learning rate scheduling[22] and regularization[23]. In addition, in all SVM models the type of learning rate scheduling was inverse scaling (`invscaling`), which reduces the learning rate over time, and the number of CPUs (`n_jobs`) was 10. Also, to avoid overfitting, early stopping was used in all SVM models by setting `early_stopping` to `True`, limiting the maximum number of epochs (`max_iter`) to 200, and forcing the training to stop after 10 epochs with no improvement on validation set (`n_iter_no_change`). Other parameters in KNN and SVM models were left to default values.

Then, the best KNN and SVM model in each classification task were chosen by comparing performance of models that were best in parameter-tuning on test set. Their results were then also compared with the results of fine-tuned BERT models.

4.3. Performance metrics

Performance of the classification models (incl. fine-tuned BERT models) was measured with accuracy, precision, recall and F-score. Accuracy meant the proportion of correctly predicted service codes. The other metrics were calculated with the following formulas [17]:

$$precision = \frac{\text{TP}}{(\text{TP}+\text{FP})}, \tag{2}$$

[18] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[19] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
[20] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
[21] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
[22] Initial learning rate (`eta0`) and exponent in the learning rate schedule (`power_t`).
[23] Type of regularization, i.e. either L1 or L2 (`penalty`) and regularization strength (`alpha`).

$$recall = \frac{\text{TP}}{(\text{TP+FN})} \quad \text{and} \tag{3}$$

$$F\text{-}score = 2 \times \frac{precision \times recall}{precision + recall}, \tag{4}$$

where TP, FP and FN stand for true positives, false positives and false negatives.

Since predicting the mask was essentially multi-class classification, those metrics were first calculated for each class (service code) separately, with precision showing the proportion of predictions of this code that were correct, and recall the proportion of the actual codes that were classified correctly. F-score combined precision and recall by taking their harmonic mean. The metrics for individual service codes were then aggregated by taking their weighted average, using the number of true instances (actual masked service codes) for each class as weights[24].

For measuring performance of word2vec models and pre-trained BERT models, a wider set of metrics, given in Table 5, was used.

Table 5. Performance metrics used for word2vec and pre-trained BERT models

Performance metric	Meaning
Top1 accuracy	Proportion of correctly predicted service codes
Top3 and top5 accuracy	Proportion of predictions where actual service code was among respectively 3 or 5 service codes with highest probability (for BERT) or similarity (for word2vec) attributed by the model
Precision, recall and F-score	See equations (2), (3) and (4)
Top1 subtype accuracy	Proportion of predictions where the subtype of the predicted service code was correct (i.e. same as subtype of the actual service code)
Top3 and top5 subtype accuracy	Proportion of predictions where subtype of the actual service code was among the subtypes of 3 or 5 service codes with highest probability (for BERT) or similarity (for word2vec) attributed by the model
Subtype precision, recall and F-score	Precision, recall and F-score of the predicted service code belonging to the subtype of the actual service code
Average rank of actual service code	Average rank of actual service code in the probabilities (for BERT) or similarities (for word2vec) output of predictions of the model
Avg. probability/ similarity of actual service code	Average probability (for BERT) or similarity (for word2vec) of the actual service code in the probabilities or similarities output of predictions of the model

The reason for using a wider range of metrics (incl. top3 and top5 accuracies, average rank and probability/similarity and metrics related to subtypes) was that the mask prediction task was a rather difficult classification task with 3,434 highly unbalanced classes, and was therefore not expected to give excellent results. Measuring also if the actual service code, even if it was not correctly predicted, was at least among the 3 or 5 service codes that the model considered most likely, or if it at least belonged to the correct subtype (or 3 or 5 most likely subtypes), and what was the rank and probability/similarity of the correct token was expected to help to better understand how the models behaved.

[24] In practical implementation, the `precision_score()`, `recall_score()` and `f1_score()` methods in `sklearn.metrics` library (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics) were applied for this with the parameter `average='weighted'`.

For calculating average rank and average probability/similarity of actual service code, as well as top3 and top5 metrics, a vector of probabilities or similarities was obtained from the prediction output. In case of BERT, the vector of probabilities was obtained by applying softmax on the prediction logits (i.e. unnormalized scores assigned by the model to each token in the vocabulary). Softmax is a function that maps the values of a numeric vector **z** into a probability distribution:

$$softmax(\boldsymbol{z}_i) = \frac{\exp(\boldsymbol{z}_i)}{\exp(\boldsymbol{z}_1) + \ldots + \exp(\boldsymbol{z}_K)}, 1 \leq i \leq K, \quad (5)$$

where *K* is the length of the vector [7]. In case of word2vec, a vector of similarity values was obtained by applying word2vec's `most_similar()` method to the service codes in the context window. This gave cosine similarities (defined by equation (1)) between the embedding of the context window, calculated as arithmetic mean of all service codes within it, and the embedding of each service code in the vocabulary of the model. The possible values of rank ranged from 1 to 3,435 (i.e. size of vocabulary without special tokens), with smaller values of rank being better.

Since the performance of pre-trained BERT models was far from encouraging, the above metrics were for comparison also calculated for a random guess approach where predictions were made by randomly choosing a service code.

5. Results

In this Chapter, the results of pre-trained BERT models and word2vec models are presented in Section 5.1 and the results of fine-tuned BERT models and classifiers built on top of word2vec embeddings in Section 5.2.

5.1. Embeddings for service codes

5.1.1 Pre-trained BERT models

The performance of pre-trained BERT models in mask prediction task is given in Table 6, with the best test and train result marked in bold for each row. We can see that the performance was really poor, with the best model (BERT_PRE2) achieving top1 accuracy of 0.0055, top3 accuracy of 0.0119 and top5 accuracy of 0.0165, and the other metrics being equally poor. However, when looking at the results, it must be borne in mind that the mask prediction task was essentially a difficult classification task with 3,434 unbalanced classes, and was therefore not expected to give excellent results. It should be stressed that top1, top3 and top5 accuracies were still around 10 times higher than random guess for all models, except for **BERT_PRE4** where the difference was smaller. Also, both average rank and probability of the actual service code were better for all BERT models than for random guess. Despite precision being higher for random guess, F-score, which combines precision and recall, was clearly better for all BERT models. Overall, it can be concluded that despite the results looking discouraging, the pre-trained models were still better than random guess, at least at token level.

Table 6. Mask prediction performance of pre-trained BERT models and random guess*

		TRAIN (1M)					TEST (1M)				
		BERT_PRE1	**BERT_PRE2**	**BERT_PRE3**	**BERT_PRE4**	**Random**	**BERT_PRE1**	**BERT_PRE2**	**BERT_PRE3**	**BERT_PRE4**	**Random**
TOKEN LEVEL METRICS	Top1 acc.	0.0052	**0.0055**	0.0049	0.0018	0.0003	0.0051	**0.0055**	0.0049	0.0018	0.0003
	Top3 acc.	0.0103	**0.0120**	0.0090	0.0036	0.0009	0.0103	**0.0119**	0.0090	0.0037	0.0009
	Top5 acc.	0.0147	**0.0165**	0.0133	0.0057	0.0015	0.0148	**0.0165**	0.0135	0.0059	0.0014
	Precision	0.0138	0.0106	0.0162	0.0098	**0.0605**	0.0126	0.0087	0.0165	0.0109	**0.0622**
	Recall	0.0017	0.0018	**0.0028**	0.0010	0.0003	0.0016	0.0018	**0.0028**	0.0010	0.0003
	F-score	0.0022	0.0023	**0.0039**	0.0016	0.0005	0.0021	0.0023	**0.0039**	0.0017	0.0005
SUBTYPE LVL. METRICS	Top1 acc.	**0.0849**	0.0653	0.0623	0.0371	0.0837	**0.0847**	0.0654	0.0625	0.0372	0.0838
	Top3 acc.	0.2196	0.2004	0.1924	0.1525	**0.2772**	0.2187	0.2001	0.1926	0.1524	**0.2773**
	Top5 acc.	0.2868	0.2715	0.2506	0.2195	**0.3834**	0.2862	0.2716	0.2508	0.2195	**0.3836**
	Precision	0.1284	0.1384	0.1429	0.1135	**0.1989**	0.1265	0.1384	0.1431	0.1125	**0.1982**
	Recall	**0.0849**	0.0653	0.0623	0.0371	0.0601	**0.0847**	0.0654	0.0625	0.0372	0.0602
	F-score	**0.0795**	0.0685	0.0661	0.0432	0.0720	**0.0790**	0.0686	0.0663	0.0432	0.0722
OTHER	Avg. rank	1688	1612	**1535**	**1535**	1719	1688	1614	1536	**1535**	1720
	Avg. probability	0.0032	**0.0038**	0.0031	0.0010	0.0003	0.0032	**0.0038**	0.0032	0.0010	0.0003

* In Table 6, colour scaling has been applied separately for token and subtype level metrics, and each of the *'Other'* metrics, with reference to *'Random'* column. Best result in each row for train and test set is marked in bold.

At the same time, subtype performance of pre-trained BERT models was surprisingly bad, with some metrics (top3 and top5 subtype accuracy and subtype precision) being worse than random guess even for the best pre-trained BERT model (BERT_PRE1). We can indeed say that at

subtype level, pre-trained BERT models failed. A possible explanation for this is that subtypes might have contained medical services that differed a lot from each other, in terms of appearing in very different contexts.

Also, we can see that both at subtype and token level, precision was considerably worse in BERT models compared to random guess, while recall was clearly better than random at token level and for most BERT models also somewhat better than random at subtype level. This seems to suggest that the models predicted too many service codes as belonging to some very frequent classes. This could have been the reason because precision, recall and F-score were first calculated for each individual service code, followed by taking weighted average of the results using counts of actual service codes (or subtypes) as weights (see Section 4.3). Therefore, if the model had poor precision for some frequent service code(s) or subtype(s), this would have torn down also its overall precision (and consequently also F-score).

When comparing the four pre-trained BERT models (see Table 6), the performance of the two larger ones (`BERT_PRE1` and `BERT_PRE2`) was quite similar, with top1, top3 and top5 accuracies, F-score and average rank and probability of the actual service code being somewhat better for `BERT_PRE2`, and subtype accuracies and subtype F-score somewhat better for `BERT_PRE1`. As mentioned in Section 4.1.1, the two larger models having similar performance was also the reason for additionally training the two smaller ones, since it was thought that perhaps smaller models might perform better. Indeed, as shown in Table 6, average rank was best with the two smaller models, and the second smallest (`BERT_PRE3`) also had better precision, recall, F-score and subtype precision. Otherwise, though, the larger models had better performance metrics.

When comparing results on training and test set, (see Table 6), the models seemed to underfit. Underfitting means that a model is too simplistic to adequately represent the underlying relationships between the input features and the target feature [17]. Often this is due to either the model not being complex enough or not having enough training data available [18]. The first reason was not likely to apply, given that 12,170,096 parameters in the best fine-tuned model (`BERT_PRE2`) surely made it complex enough. As regards the size of training set, although 39 million instances seems a lot, the great majority of the trajectories were very short (see Figure 4), and the trajectories also repeated a lot (see Table 4). Indeed, it could have been possible to augment training data by reordering services offered on the same date in different ways. This might have helped to cancel out the possible effect of such services being logged to the system not exactly in the same order as they were offered on that date, making it easier for the models to learn that the order of services on the same date is not that important. However, such data augmentation was considered out of scope for this thesis.

It is also possible that using the modified position embeddings similar to BEHRT [4] (where diagnosis codes within the same visit were assigned the same position embedding) would have improved results. The MLM precision of 0.6597 reported for BEHRT [4] was indeed clearly better than precision of the best model in this thesis (0.0165). However, it must be kept in mind that the results of BEHRT are not directly comparable with results in this thesis for at least two reasons. Firstly, the vocabulary size of BEHRT – and therefore the number of classes in mask prediction task – was much smaller (301 diagnosis codes, compared to 3,435 service codes in this thesis). Secondly, the trajectories in BEHRT were longer (trajectories containing less than five visits were not used, while each visit could contain multiple diagnosis codes), while the shortest trajectories in this thesis contained only two service codes. As the analysis of performance per trajectory length (see section 5.1.3.1) shows, very short trajectories are in general much harder to predict.

Still, the poor performance of pre-trained BERT models raised the question what MLM accuracy (i.e. accuracy of predicting the mask) could expected to be in natural language models. As already mentioned in Section 4.1.1, the focus in training BERT models is often only on downstream tasks, and MLM performance is usually not reported. However, in the extensive BERT literature, some indications can still be found. For example, in [19], where a BERT model for analysing COVID-19 content on Twitter was proposed, MLM accuracy was close to 0.7. In [20], where LV-BERT was proposed as a further development of BERT, it was around 0.57. Therefore, MLM accuracy of the best pre-trained BERT model in this thesis (0.0055) was clearly far behind of what could be expected with natural language data.

Although one might think that perhaps it is simply not possible for any model to learn to predict the masked token from this type of input data, the results of word2vec (see Section 5.1.2) show that some models can perform remarkably better. Since BERT is inherently more powerful, with many more parameters, the most likely reason for its bad performance was that the parameters in the models were not efficiently learnt. Therefore, the problem was most likely in the choice of input parameters used in pre-training (see Section 4.1.1 and Appendix IX). Although the input parameters were mostly the default ones and thus likely to work reasonably well with natural language data, apparently they were not appropriate with patient treatment trajectories. However, since pre-training BERT is a lengthy process, the only parameter tuning applied in this thesis was pre-training four models with different dimensions.

The results of the models are further analysed in Section 5.1.3.

5.1.2. Word2vec models

Results of all 20 word2vec models on evaluation set are shown in Figure 14.

Top1 accuracy

Window size \	150	250	400	600
3	0.065	0.066	0.068	0.061
5	0.070	0.064	**0.087**	0.075
7	0.074	0.068	0.069	**0.087**
10	0.080	0.066	0.067	0.082
20	0.083	0.079	0.081	0.068

Top3 accuracy

Window size \	150	250	400	600
3	0.186	0.200	**0.202**	0.199
5	0.191	0.189	0.192	0.195
7	0.196	0.189	0.187	0.192
10	0.180	0.191	0.182	0.190
20	0.178	0.171	0.176	0.183

Top5 accuracy

Window size \	150	250	400	600
3	0.266	0.268	0.270	**0.271**
5	0.259	0.257	0.260	0.258
7	0.259	0.251	0.253	0.256
10	0.248	0.251	0.249	0.251
20	0.244	0.236	0.240	0.247

Top1 subtype acc.

Window size \	150	250	400	600
3	0.344	0.345	0.328	0.276
5	0.349	0.364	0.354	0.380
7	**0.389**	0.375	0.379	0.386
10	0.360	0.379	0.377	0.378
20	0.372	0.329	0.374	0.377

Top3 subtype acc.

Window size \	150	250	400	600
3	0.442	0.444	0.443	0.442
5	0.445	0.449	0.448	0.451
7	0.450	0.449	0.450	**0.455**
10	0.445	0.448	0.449	0.448
20	0.448	0.446	0.452	0.454

Top5 subtype acc.

Window size \	150	250	400	600
3	0.490	0.503	0.499	**0.505**
5	0.490	0.489	0.497	0.495
7	0.485	0.493	0.494	0.494
10	0.479	0.488	0.490	0.493
20	0.485	0.485	0.486	0.493

F-score

Window size \	150	250	400	600
3	0.048	0.050	0.052	0.049
5	0.049	0.049	**0.054**	0.049
7	0.051	0.047	0.047	0.052
10	0.050	0.046	0.046	0.051
20	0.048	0.048	0.049	0.045

Subtype F-score

Window size \	150	250	400	600
3	0.312	0.314	0.303	0.267
5	0.313	0.321	0.317	0.333
7	**0.336**	0.332	0.332	0.335
10	0.319	0.329	0.328	0.328
20	0.320	0.296	0.323	0.328

Avg. rank

Window size \	150	250	400	600
3	277	266	274	267
5	**237**	265	264	257
7	303	290	294	274
10	362	391	342	343
20	426	479	459	441

Figure 14. Performance of word2vec models on evaluation set

For easier comparison, results of each performance metric in Figure 14 are reported by window size and vector length that were used as input parameters in the model. Best result for each metric is marked in bold. Based on performance on evaluation set (see Figure 14) three models were selected to go forward with in the classification stage:

- `W2V_w5_v400`
- `W2V_w7_v600`
- `W2V_w7_v150`

The first two (`W2V_w5_v400` and `W2V_w7_v600`) had the highest top1 accuracies, and the third one (`W2V_w7_v150`) had the highest top1 subtype accuracy. The first one also had the highest F-score and third best average rank. The second one also had second highest top1 subtype accuracy, F-score and subtype F-score. The third one also had the highest subtype F-score.

Performance of the three best word2vec models on test set (and for comparison, on the reduced version of training set) is given in Table 7. Performance on training and test set was very similar, so the models did not overfit.

Table 7. Mask prediction performance of best word2vec models*

		TRAIN (1 million)			TEST (1 million)			Ran-dom
		W2V_w5_v400	W2V_w7_v600	W2V_w7_v150	W2V_w5_v400	W2V_w7_v600	W2V_w7_v150	
TOKEN LVL. METRICS	Top1 acc.	**0.0873**	0.0870	0.0741	**0.0871**	0.0869	0.0740	0.0003
	Top3 acc.	0.1926	0.1917	**0.1965**	0.1924	0.1914	**0.1964**	0.0009
	Top5 acc.	**0.2596**	0.2561	0.2586	**0.2593**	0.2556	0.2581	0.0014
	Precision	**0.0542**	0.0516	0.0525	**0.0544**	0.0525	0.0517	0.0622
	Recall	**0.0873**	0.0870	0.0741	**0.0871**	0.0869	0.0740	0.0003
	F-score	**0.0544**	0.0522	0.0513	**0.0544**	0.0544	0.0512	0.0005
SUBTYPE LEVEL METRICS	Top1 acc.	0.3536	0.3855	**0.3885**	0.3532	0.3857	**0.3885**	0.0838
	Top3 acc.	0.4483	**0.4545**	0.4499	0.4484	**0.4549**	0.4502	0.2773
	Top5 acc.	**0.4968**	0.4937	0.4850	**0.4970**	0.4938	0.4852	0.3836
	Precision	0.3018	**0.3080**	0.3068	0.3017	**0.3079**	0.3070	0.1982
	Recall	0.3536	0.3855	**0.3885**	0.3532	0.3857	**0.3885**	0.0602
	F-score	0.3171	0.3348	**0.3353**	0.3168	0.3351	**0.3354**	0.0722
OTHER	Avg. rank	**264**	275	303	**264**	275	304	1720
	Avg. simi-larity	**1.5814**	1.5212	1.5667	**1.5805**	1.5203	1.5656	0.0003

* In Table 7, colour scaling has been applied separately for each row, except for the *'Random'* column, where the colour depicts how much values in that column differed from the minimum values in other columns. Best result in each row for train and test set is marked in bold.

Performance of word2vec models (see Table 7) on test set was remarkably better compared to pre-trained BERT models (see Table 6), with best top1 accuracy and F-score being respectively 0.0871 and 0.0544 (compared to 0.0055 and 0.0039 in BERT) and top1 subtype accuracy and subtype F-score respectively 0.3885 and 0.3354 (compared to 0.0847 and 0.0790 in BERT). Also, the best average rank of actual service code in word2vec models was 264 (compared to 1535 in BERT).

Therefore, it can be concluded that the non-contextual embeddings of word2vec performed better in the mask prediction task than the contextual embeddings of pre-trained BERT. Apart from the possibility that the input parameters of BERT models were simply not appropriate, it

is also possible that word2vec performed better because 'word order' in the 'language' of treatment trajectories is more flexible than in natural language, and thus non-contextual embeddings (which do not try to learn which service codes should follow one another) might indeed work better. The 'word order' being more flexible could have been caused by services provided on the same date not being recorded in the system exactly in the same order as they were provided, or by cases where the precise order of services provided around the same time is not strictly fixed.

5.1.3 Further analysis of results

In this section, the performance of both word2vec and pre-trained BERT models is further analysed by looking at how predictions depended on trajectory length (see Section 5.1.3.1). In addition, results of PCA performed on the embeddings of 10 selected services from one of the pre-trained BERT models (`BERT_PRE2`) is presented as a small case study in Section 5.1.3.2, as a way to explain why the predictions of this type of models were not so good. Finally, Section 5.1.3.3 includes some remarks on how the chosen masking strategy might have influenced performance.

5.1.3.1 Accuracies per trajectory length

Since treatment trajectories varied a lot in length (see Figure 4), it seemed relevant to see whether and to what extent performance of the models depended on trajectory length. For this purpose, top1 accuracies were calculated separately for different ranges of trajectory lengths. The results are shown in Table 8. For comparison, accuracies on all test data without breakdown into trajectory lengths is shown in column *'All'*. Since the proportion of trajectories of different lengths differed considerably, with the majority of trajectories being very short, also the distribution of trajectories by length is given (on row *'% of data'*). The shortest trajectory length 3 corresponds to a trajectory of two service codes and full stop. In Table 8, the largest value in each column is marked in bold.

Table 8. Top1 accuracies by trajectory length on test set

		Trajectory length							
		3-5	**6-10**	**11-15**	**16-25**	**26-50**	**51-80**	**81-126**	**All**
% of data		**59.3%**	22.8%	9.3%	6.3%	1.8%	0.3%	0.2%	100.0%
Models	**BERT_PRE1**	0.004	0.005	0.009	0.010	0.012	0.015	0.015	0.005
	BERT_PRE2	0.004	0.005	0.009	0.010	0.013	0.019	0.022	0.006
	BERT_PRE3	0.003	0.006	0.010	0.011	0.016	0.029	0.031	0.005
	BERT_PRE4	0.001	0.002	0.003	0.004	0.006	0.009	0.010	0.002
	W2V_w5_v400	0.061	0.101	**0.132**	**0.160**	**0.210**	**0.252**	**0.333**	**0.087**
	W2V_w7_v600	**0.064**	**0.103**	0.123	0.143	0.190	0.220	0.308	**0.087**
	W2V_w7_v150	0.043	0.101	0.122	0.150	0.196	0.233	0.307	0.074

Results in Table 8 show that all models made better predictions when longer trajectories were used as input. Top1 accuracies increased with trajectory length in case of all models, achieving accuracy of 0.333 for the best word2vec model (`W2V_w5_v400`) and 0.031 for the best pre-trained BERT model (`BERT_PRE3`) with the longest trajectories. For comparison, top1 accuracies of the same models on all test data were only 0.087 and 0.005, i.e. respectively 3.8 and 6.2 times lower. However, since very short trajectories made up the majority of data, the

poor results on those trajectories outweighed the much better results on longer ones. Still, even with the shortest trajectory length, top1 accuracies of even the worst models were still much better than accuracy random guess (0.0003).

It can be concluded that one of the weaknesses of all models was that they were unable to make good predictions if the input trajectory was very short, which however made up the majority of the data. An explanation to the poor results on short trajectories is that the models simply did not have enough input data to make an informed decision. For example, in case of the shortest trajectories of length 3, which actually made up as much as 30% of the data (see Figure 4)[25], the models had to make a prediction based on either only one code and the full stop token, or two codes (if full stop was masked).

5.1.3.2 Case study: PCA of selected BERT service embeddings

The PCA performed on `BERT_PRE2` embeddings of 10 selected services (see Figure 15) showed that the pre-trained embeddings of the model were indeed contextual, with the same service having different embedding depending on the trajectory where it appeared. If this had not been the case, Figure 15 would have featured exactly one point in each colour. We can also see that the service embeddings formed clusters. However, we can see that the clusters in 2D were very wide and diffuse and heavily overlapped with each other, which very likely was also the reason why mask prediction performance of the pre-trained model was so poor.

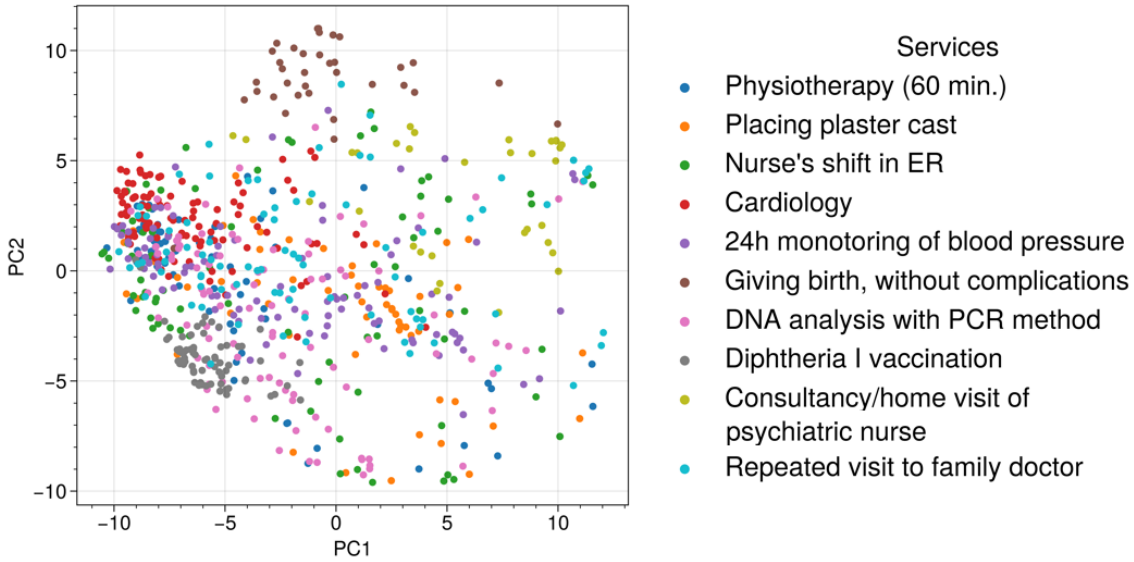


Figure 15. PCA on BERT word embeddings of selected services

In Figure 15, *'Repeated visit to family doctor'* being among the most diffuse clusters seems logical, since this service can surely appear in a wide range of treatment trajectories. The diffuseness of '*Nurse's shift in ER'* could perhaps be attributed to the same reason. *'24h monitoring of blood pressure'* is also very diffuse, but more concentrated in the area also occupied by '*Cardiology',* which also seems a logical result. On the other hand, the reasons for *'DNA analysis with PCR method'* and '*Placing plaster cast'* being so diffuse are not clear. The services with most concentrated clusters were '*Giving birth, without complications'* (which also overlapped the least with other clusters) and '*Diphtheria I vaccination'*. However, the overlappings of the clusters, which were very extensive, often did not seem easily explainable.

[25] In Figure 4, the lengths do not include full stop and are therefore smaller by 1.

In order to quantify the clustering, the mean in-cluster distance (average distance between cluster centre and each vector in the cluster, averaged over all clusters) was compared to the mean inter-cluster distance (average distance from cluster centre to all other cluster centres, averaged over all clusters). In general, clustering can be considered good if clusters are compact (in-cluster distance is small) and cluster centres are far from each other (inter-cluster distance is large). In this case, however, mean in-cluster distance was 12.33 ± 2.36 in the original space (and 5.17 ± 2.10 in 2D space), while mean inter-cluster distance was 12.18 ± 2.15 in the original space (and 7.26 ± 2.36 in 2D space). In-cluster distance being similar to inter-cluster distance in the original space confirms that the clusters were wide and heavily overlapping.

Nevertheless, from the qualitative assessment above it can be concluded that service embeddings learnt in pre-training still somewhat made sense, which showed that even if the model was not good at predicting the masked service code, it still had at least learnt something.

5.1.3.3 Impact of masking strategy on performance

The masking strategy applied in this thesis when testing performance was to mask one service code in each trajectory, which meant one prediction per trajectory. As explained in Section 3.1.1, this strategy was chosen instead of the built-in one used in pre-training (where 15% of tokens were masked), because the latter resulted in longer trajectories having more than one mask and some short trajectories not having any masks at all. However, it would not have made sense to have trajectories with no masks in the test set, and predicting several masks in the same trajectory at a time was not possible (although actually such trajectories could have been duplicated). This masking strategy was chosen, because it seemed clear and straightforward. However, later analysis of what was actually masked showed that this simple strategy might not have been best.

Namely, this masking strategy had two important implications. First, service codes appearing in long trajectories had smaller chance of being masked, and secondly, more frequent service codes were also masked more frequently. A closer look at what was actually masked revealed that out of 3,435 service codes (and full stop) in the vocabulary of the models, 1,307 were never masked. Therefore, the ability of the models to predict them was not measured on test set. The analysis also showed that the most frequently masked token was full stop, which made up as much as 21% of all masked tokens. This can be explained by full stop appearing at the end of all trajectories, which made it the most frequent token. Its probability of getting masked was especially high in short trajectories, which made up large proportion of the data (see Figure 4).

The effect that so many full stops being masked had on performance was evident when looking at predictions. For example, `W2V_w5_v400` predicted it in 499 different ways (i.e. in more ways than any other token), but never correctly. The most frequent prediction for it (25% of predictions) was *'Repeated visit to family doctor'*. Such result could be explained by the context window of full stop containing a wide range of different tokens during training, which likely resulted in the model learning that it was 'similar to everything', and being very much unconfident when it had to predict it.

For comparison, `BERT_PRE2` indeed did better in this task, by predicting full stop correctly in 0.34% of the cases, which was still lower than the overall accuracy of this model. Interestingly, the most frequent token it predicted instead of full stop was *'Chromosome analysis from the skin'* (42% of predictions), followed by some other specific medical procedures. The reasons for such outcome do not seem entirely clear.

Considering the implications above, it must be concluded that the chosen masking strategy might not have been the best. A more elaborate strategy, for example duplicating longer

sentences to predict more tokens in them, or simply never masking full stop, or not adding it to the end of the trajectories in the first place, might have given better picture of performance of the models. In order to avoid not having any predictions for quite large number of less frequent tokens, a possible approach could have been to mask each token in each trajectory by duplicating the trajectories respective number of times, so that each test instance would have contained just one mask. However, the results of masking were only analysed after all models had already been trained and tested, and it was no longer possible to repeat this process.

5.2. Classification models

5.2.1 Results of parameter-tuning of KNN and SVM models

The performance of all KNN and SVM models built on top of each of the three word2vec models on test set (and for comparison, on training set) is shown in Appendix XII and Appendix XIII, respectively. The models reported were the best in parameter-tuning on evaluation set. (For details on parameter-tuning, see Section 4.2.2.) In the appendices, the KNN and SVM models that were best in each classification task are marked in bold. If several models are marked in bold for the same task, this means that their performance was found not to be statistically significantly different based on McNemar test with significance level 0.05. From among the best models for each task, only one SVM and one KNN model (underlined in the appendices), i.e. the ones with highest accuracy, were chosen for comparison with BERT models. The results of those best models are discussed together with the results of BERT models in Section 5.2.2.

As regards the results in Appendix XII and Appendix XIII, we can see that KNN models built on top of different word2vec models for the same classification task seemed to perform remarkably similarly, with their accuracies differing by less than 0.01. The performance among SVM models built for the same task differed slightly more. Also, both for KNN and SVM in most tasks there were several best models whose performance was not found to be statistically significantly different.

The results in Appendix XII and Appendix XIII also show that all KNN models tended to overfit, while SVM models did not. It must be noted that in parameter-tuning, just the best models on evaluation set were picked, without considering overfitting. The fact that SVM models did not have this problem shows that the parameters specifically meant to counter overfitting in SVM (see Section 4.2.2) were indeed effective.

5.2.2 Results of fine-tuned BERT and best KNN and SVM models

The performance of all fine-tuned BERT models and the best KNN and SVM models in all classification tasks on test set is given in Table 9. For comparison, performance of fine-tuned BERT models on training set is given in Appendix XIV. Both in Table 9 and in Appendix XIV, the number of classes in each predicted feature is given in brackets in the *'Task'* column. Models that were the best in each classification task are marked in bold. If several models are marked in bold for the same classification task in Table 9, this means that their performance was found not to be statistically significantly different based on McNemar test with significance level 0.05. Column *'Best BERT vs best KNN/SVM'* in Table 9 shows difference between the best BERT and best KNN or SVM model.

Table 9. Performance of fine-tuned BERT and best KNN and SVM models on test set*

Task	Metric	BERT_FINE1	BERT_FINE2	BERT_FINE3	BERT_FINE4	Best KNN	Best SVM	Best BERT vs best KNN/SVM
diag (18)	Accuracy	**0.5853**	**0.5820**	**0.5854**	0.5476	0.5478	0.5475	0.0376
	Precision	**0.5829**	**0.5816**	**0.5907**	0.5736	0.5461	0.5417	0.0446
	Recall	**0.5853**	**0.5820**	**0.5854**	0.5476	0.5478	0.5475	0.0376
	F-score	**0.5736**	**0.5693**	**0.5700**	0.5159	0.5352	0.5311	0.0384
diag2 (174)	Accuracy	**0.4121**	**0.4067**	0.3644	0.2452	0.3670	0.3445	0.0451
	Precision	**0.4329**	**0.4441**	0.5456	0.6574	0.3717	0.3720	0.2854
	Recall	**0.4121**	**0.4067**	0.3644	0.2452	0.3670	0.3445	0.0451
	F-score	**0.3755**	**0.3629**	0.2936	0.1482	0.3308	0.3241	0.0447
emergency_bill (2)	Accuracy	**0.9602**	**0.9601**	**0.9603**	**0.9589**	0.9557	0.9529	0.0046
	Precision	**0.9589**	**0.9590**	**0.9590**	**0.9575**	0.9542	0.9510	0.0048
	Recall	**0.9602**	**0.9601**	**0.9603**	**0.9589**	0.9557	0.9529	0.0046
	F-score	**0.9591**	**0.9593**	**0.9592**	**0.9573**	0.9545	0.9508	0.0048
discharge_status (7)	Accuracy	0.6052	**0.6177**	0.6099	0.5927	0.5944	0.5652	0.0233
	Precision	0.5936	**0.6049**	0.5950	0.5933	0.5824	0.5373	0.0225
	Recall	0.6052	**0.6177**	0.6099	0.5927	0.5944	0.5652	0.0233
	F-score	0.5900	**0.6051**	0.5956	0.5773	0.5790	0.5389	0.0261
treatment_type (5)	Accuracy	**0.9913**	**0.9918**	**0.9914**	0.9885	0.9851	0.9876	0.0042
	Precision	**0.9913**	**0.9918**	**0.9914**	0.9885	0.9851	0.9876	0.0042
	Recall	**0.9913**	**0.9918**	**0.9914**	0.9885	0.9851	0.9876	0.0042
	F-score	**0.9913**	**0.9918**	**0.9914**	0.9885	0.9850	0.9876	0.0042
tto_type (5)	Accuracy	0.7817	**0.7900**	0.7834	0.7454	0.7468	0.7183	0.0432
	Precision	0.7841	**0.7913**	0.7773	0.7584	0.7399	0.7120	0.0514
	Recall	0.7817	**0.7900**	0.7834	0.7454	0.7468	0.7183	0.0432
	F-score	0.7794	**0.7873**	0.7782	0.7459	0.7404	0.7027	0.0469
largest4 (5)	Accuracy	**0.8240**	**0.8288**	0.8182	0.7741	0.7683	0.7369	0.0605
	Precision	**0.8083**	**0.8150**	0.7996	0.7339	0.7382	0.6650	0.0768
	Recall	**0.8240**	**0.8288**	0.8182	0.7741	0.7683	0.7369	0.0605
	F-score	**0.8113**	**0.8188**	0.8045	0.7412	0.7479	0.6813	0.0709

* In Table 9, colour scaling has been applied separately for each classification task, and separately for the column *'Best BERT vs best KNN/SVM'*.

In this section, performance of the models is compared based on accuracies. In Table 9, also F-score, and its calculation components precision and recall, are reported for each model for reference. They were calculated in the first place because F-score is a metric often used when the classes are imbalanced [21]. However, in this case, accuracies and F-scores did not differ that much, and the best models were the same based on both metrics.

The results in Table 9 show that fine-tuned BERT models performed surprisingly well, especially considering the overall horrible performance of the pre-trained models (see Table 6). Indeed, despite word2vec models being markedly better in predicting the mask (see Table 7), the best fine-tuned BERT models outperformed the best classifiers built upon word2vec

vectors in all classification tasks. Therefore, it can be concluded that the contextual embeddings of BERT were better than the non-contextual ones of word2vec.

Among fine-tuned BERT models, the second largest (`BERT_FINE2`) can be considered as the overall best, since it was among the set of best-performing models (marked in bold in Table 9) in all 7 classification tasks. The next best was `BERT_FINE1`, then `BERT_FINE3`, and finally the worst was `BERT_FINE4` (among the best respectively in 5, 3 and 1 task). The fact that the largest model (`BERT_FINE1`) was not the best suggests that when the 'language' of treatment trajectories is less complicated than natural language, then using smaller BERT models than the original one proposed by the authors of BERT [1] might be beneficial. When comparing the results of BERT models on test set (see Table 9) with their results on training set (see Appendix XIV), we can notice that the models somewhat overfitted.

Among classifiers built on top of word2vec vectors, KNN outperformed SVM – it was better in four tasks (`diag2`, `discharge_status`, `tto_type` and `largest4`), while SVM was better only in one (`treatment_type`). In the remaining two tasks, their performance was not statistically different. KNN outperforming SVM may indicate that the tasks were better solvable by a non-linear than a linear model.

For better overview, the accuracies of the models are shown in Figure 16. In the figure, the number of classes is given in brackets for the predicted features. The best accuracy (0.9918) was achieved in classifying `treatment_type` (with `BERT_FINE2`), followed by `emergency_bill` (0.9603 with `BERT_FINE3`). This shows that treatment trajectories of services provided in inpatient, outpatient and day care and by family doctors and nurses on the one hand, and whether or not the invoice was for emergency services on the other hand, were easily distinguishable. At the same time, it was a bit surprising that classifying `emergency_bill` was harder, since it only had 2 classes compared to 5 in `treatment_type`, while in general the models performed worse with higher number of classes.

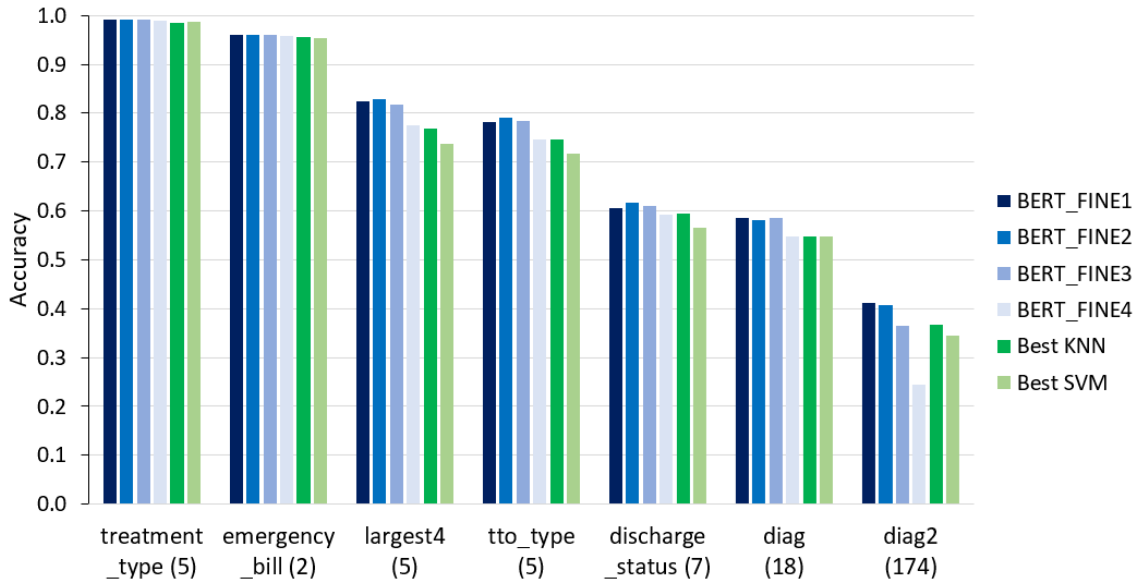


Figure 16. Accuracies of fine-tuned BERT and best KNN and SVM models

Performance in classifying `largest4` and `tto_type` was rather good, with best accuracies of 0.8288 and 0.7900, respectively (both with `BERT_FINE2`). The fact that the performance of the latter was slightly weaker shows that it was somewhat harder for the models to decide the

specific type of service provider (local, regional, central or general hospital or chosen partner) than which one of the four largest or other service providers it was. Interestingly, the rather good performance in classifying `largest4` indicated that treatment trajectories of the four largest hospitals must have been distinguishable. This seemed interesting, because from the trajectories being different it can be concluded that also the medical practices of those large hospitals at least to some extent differed.

Lastly, Figure 16 shows that classification tasks with the highest number of classes were the most difficult for the models to solve, with the best accuracies of `diag2` (174 classes), `diag` (18 classes) and `discharge_status` (7 classes) being the lowest – respectively 0.4121 (with `BERT_FINE1`), 0.5854 (with `BERT_FINE3`) and 0.6177 (with `BERT_FINE2`). The moderate performance of `discharge_status` also shows that what happened to the patient next after s/he was discharged (i.e. what type of treatment s/he was referred to, or whether s/he recovered or deceased) was harder to predict than what type of treatment s/he was currently undergoing (i.e. `treatment_type`).

In general, it can be concluded that performance of fine-tuned BERT models in classifying patient treatment trajectories was quite good, and better than performance of classifiers built on top of word2vec models. Therefore, it can be concluded that the contextual embeddings were indeed more efficient with this type of input data than non-contextual ones.

5.2.3. Further analysis of results

5.2.3.1. Closer look at prediction errors of best fine-tuned BERT model

In this section, a closer look is taken at prediction errors of `BERT_FINE2` models, which were the overall best across all classification tasks (see Section 5.2.2). The errors are analysed by looking at confusion matrices of each of the classification tasks. The values in the matrices show which percentage of instances in each class were predicted to belong to each of the classes. The total number of instances in each class is given under each matrix for reference. For example, in the first confusion matrix (see Table 10), the value 1.0% in the first column shows that 1% out of 4,598 instances in class *'Family doctor'* were predicted as belonging to class *'Outpatient treatment'*. The values on the main diagonal are class accuracies.

Table 10. Confusion matrix of `BERT_FINE2_treatment_type`

		Actual				
		Family doctor	**Outpatient treatment**	**Inpatient treatment**	**Day care**	**Nursing**
Predicted	**Family doctor**	99.0%	0.6%	0.0%	0.0%	0.0%
	Outpatient treatment	1.0%	99.4%	1.0%	1.1%	0.0%
	Inpatient treatment	0.0%	0.0%	99.0%	0.0%	0.0%
	Day care	0.0%	0.0%	0.0%	98.9%	0.0%
	Nursing	0.0%	0.0%	0.0%	0.0%	100.0%
Total		4,598	4,645	524	177	56

The model predicting treatment types rarely made any errors at all (see Table 10).

The model predicting discharge status (see Table 11) was best at predicting recovery and worst at predicting if the patient deceased, with the respective class accuracies being 83.5% and 0.0%. The likely reason for this is that *'Recovery'* was the largest and *'Deceased'* the smallest class.

Also, it often predicted patients referred to outpatient or inpatient treatment or in class *'Other'* as recovered, and patients referred to day care or inpatient treatment as referred to outpatient treatment.

Table 11. Confusion matrix of `BERT_FINE2_discharge_status`*

		Actual						
		Reco-very	**>> family doctor**	**>> out-patient treatm.**	**>> in-patient treatm.**	**>> day care**	**Deceased**	**Other**
Predicted	**Recovery**	83.5%	10.4%	26.0%	21.7%	6.8%	15.0%	25.6%
	>> family doctor	1.6%	47.2%	5.8%	8.7%	13.6%	30.0%	3.5%
	>> outpatient treatment	6.1%	17.4%	41.7%	26.1%	31.8%	10.0%	16.8%
	>> inpatient treatment	0.1%	0.6%	0.6%	16.1%	15.9%	25.0%	0.4%
	>> day care	0.0%	0.0%	0.0%	0.0%	4.5%	0.0%	0.0%
	Deceased	0.0%	0.4%	0.0%	0.6%	0.0%	0.0%	0.1%
	Other	8.7%	24.0%	25.9%	26.7%	27.3%	20.0%	53.6%
Total		4,139	820	2,307	161	44	20	2,509

* '>>' indicates which type of treatment the patient was referred to.

The model predicting if the invoice was for emergency services or not (see Table 12) had much higher accuracy in detecting non-emergency bills (98.4%) than emergency bills (78.9%). Its main error was that it failed to identify 21.1% of the emergency bills.

Table 12. Confusion matrix of `BERT_FINE2_emergency_bill`

		Actual	
		Emergency bill	**Other bill**
Predicted	**Emergency bill**	78.9%	1.6%
	Other bill	21.1%	98.4%
Total		1,211	7,224

Table 13 shows that the weakness of the model classifying `largest4` was that it often classified treatment trajectories of the four largest hospitals as belonging to the class *'Other'*. At the same time, errors in classifying trajectories of one of the four largest hospitals as belonging to another one were quite infrequent. This confirms the observation made at the end of Section 5.2.2 that treatment trajectories of the four largest hospitals were distinguishable among themselves, which suggests that also their medical practices must have somehow differed.

Table 13. Confusion matrix of `BERT_FINE2_largest4`*

		Actual				
		ITKH	**LTKH**	**PERH**	**TÜK**	**Other**
Predicted	**ITKH**	51.8%	8.5%	7.8%	4.6%	1.7%
	LTKH	5.3%	40.5%	4.1%	1.7%	1.0%
	PERH	5.0%	2.6%	58.0%	6.3%	0.8%
	TÜK	5.5%	2.4%	7.1%	53.9%	1.7%
	Other	32.4%	46.0%	23.1%	33.5%	94.9%

The model classifying types of service providers (see Table 14) was not able to identify treatment trajectories of local hospitals (out of 26 invoices issued by them, it classified only one correctly). Again, this was the smallest class. Also, it attributed a large proportion of trajectories of general hospitals to central hospitals (38.1%) or regional hospitals (23.5%). It

was good at classifying chosen partners, though, with class accuracy of 91.3%, which again shows that the largest class was the easiest to classify.

Table 14. Confusion matrix of `BERT_FINE2_tto_type`

		Actual				
		General hospital	**Central hospital**	**Regional hospital**	**Local hospital**	**Chosen partner**
Predicted	**General hospital**	31.1%	6.9%	3.4%	15.4%	0.7%
	Central hospital	38.1%	70.8%	18.1%	34.6%	5.5%
	Regional hospital	23.5%	15.4%	72.7%	7.7%	2.4%
	Local hospital	0.1%	0.1%	0.0%	3.8%	0.1%
	Chosen partner	7.1%	6.9%	5.8%	38.5%	91.3%
Total		816	1,985	1,671	26	5,502

As regards diagnoses with 18 categories (see Table 15), the easiest trajectories to classify were those pertaining to categories '*H', 'I'* and 'J' (eye diseases and diseases of the circulatory and respiratory system, respectively), with class accuracies of 81.8%, 71.0% and 77.6%. However, in terms of class frequency, those categories were from second to fourth largest. The largest category *'Z'* (factors influencing health status and contact with health services) in this case ranked only as 5th in class accuracy (which was 64.7%), suggesting that in addition to the size of the class, also how specific its treatment trajectories were compared to other classes played an important role for the model. This is confirmed by the quite specific diagnoses of *'O'* and *'P'* (conditions related to pregnancy and childbirth, and in perinatal period) achieving relatively good accuracies (65.4% and 40.0%, respectively), despite being among the smallest classes.

Table 15. Confusion matrix of `BERT_FINE2_diag`*

		Actual																	
		A-B	**C-D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S-T**	**Z**
Predicted	**A-B**	19.0%	3.8%	0.7%	0.0%	0.4%	0.1%	0.0%	1.1%	3.0%	11.1%	0.3%	0.7%	0.0%	20.0%	0.0%	1.2%	0.4%	0.7%
	C-D	2.7%	47.1%	1.6%	1.0%	4.5%	0.3%	1.6%	1.3%	5.7%	8.3%	1.6%	4.9%	0.8%	20.0%	6.1%	3.4%	0.6%	2.8%
	E	1.4%	3.1%	49.1%	2.5%	4.5%	0.7%	5.8%	1.4%	2.7%	3.1%	1.8%	2.3%	0.8%	0.0%	12.1%	3.7%	0.4%	2.9%
	F	0.0%	0.2%	0.0%	53.5%	1.3%	0.0%	0.1%	0.0%	0.0%	0.3%	0.1%	0.0%	0.0%	0.0%	0.0%	1.8%	0.0%	0.1%
	G	0.0%	0.6%	0.5%	4.5%	26.3%	0.7%	0.4%	0.2%	0.3%	0.0%	1.2%	0.0%	0.8%	0.0%	6.1%	2.5%	1.0%	0.4%
	H	0.5%	1.5%	4.2%	0.0%	1.8%	81.8%	0.1%	3.3%	0.6%	0.0%	0.1%	0.3%	0.0%	0.0%	3.0%	2.8%	2.1%	5.9%
	I	4.8%	5.4%	26.7%	16.3%	20.5%	2.1%	71.0%	5.1%	13.6%	7.6%	12.5%	4.1%	0.0%	0.0%	15.2%	16.9%	2.1%	7.8%
	J	45.8%	5.4%	3.3%	8.4%	17.0%	8.5%	8.0%	77.6%	14.2%	20.1%	11.2%	3.5%	1.6%	0.0%	6.1%	18.4%	10.1%	4.6%
	K	1.8%	2.7%	1.4%	1.0%	0.9%	0.1%	1.4%	0.8%	46.4%	0.7%	0.9%	1.8%	0.0%	10.0%	0.0%	6.7%	0.6%	0.7%
	L	2.5%	2.5%	0.9%	0.0%	0.4%	0.5%	1.3%	0.7%	0.6%	21.5%	1.3%	0.7%	0.8%	0.0%	3.0%	0.6%	1.2%	0.6%
	M	4.1%	4.4%	2.6%	3.0%	11.2%	1.5%	3.7%	2.1%	3.0%	9.7%	56.1%	2.6%	0.0%	0.0%	15.2%	9.2%	15.3%	3.2%
	N	4.1%	7.1%	2.3%	1.5%	0.9%	0.1%	1.7%	1.5%	2.1%	1.7%	2.5%	48.1%	3.9%	0.0%	15.2%	7.7%	1.2%	3.9%
	O	0.5%	0.2%	0.0%	0.0%	0.0%	0.0%	0.0%	0.1%	0.0%	0.0%	0.0%	0.8%	65.4%	0.0%	0.0%	0.6%	0.2%	0.3%
	P	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	40.0%	0.0%	0.0%	0.0%	0.0%
	Q	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
	R	2.3%	1.1%	0.7%	3.5%	4.5%	0.4%	1.6%	1.1%	3.6%	1.7%	1.1%	1.8%	1.6%	0.0%	12.1%	13.8%	0.6%	0.5%
	S-T	0.9%	1.7%	0.5%	2.0%	3.6%	1.8%	0.5%	1.2%	1.2%	2.4%	7.0%	0.5%	0.0%	0.0%	6.1%	2.1%	61.9%	0.9%
	Z	9.8%	13.2%	5.6%	3.0%	2.2%	1.3%	2.9%	2.5%	3.0%	11.5%	2.4%	27.8%	24.4%	10.0%	0.0%	8.6%	2.5%	64.7%
Total		441	522	430	202	224	944	1,119	1,219	332	288	935	607	127	10	33	326	517	1,724

* Meanings of the categories can be found on Figure 6.

From above we could see that an issue with several models was that they wrongly predicted many test instances as belonging to the largest classes, which was likely caused by the classes being imbalanced. A method that could be used to alleviate this issue is oversampling, which increases the number of data points in the minority class by duplicating them [21]. It is indeed

possible that oversampling trajectories belonging to small classes could have improved performance. The question of whether more data could have been beneficial in fine-tuning BERT models classifying `diag2` has been analysed from another perspective also in Section 5.2.3.2.

5.2.3.2. Data-bound check for fine-tuned BERT models

Some of the fine-tuned models overfitted (see Table 9). To check if the overfitting was data-bound, i.e. caused by having too few data, a small case study was conducted with two models fine-tuned for classifying diagnosis (`BERT_FINE1_diag2` and `BERT_FINE4_diag2`), which were among the ones that overfitted respectively the most and the least. This involved training those models with different amount of training data (25%, 50%, 70%, 80%, 90% and 100% of the original training set). Similar to models trained on the entire training set (containing 45,000 samples), early stopping was applied here as well. The results are shown in Figure17.

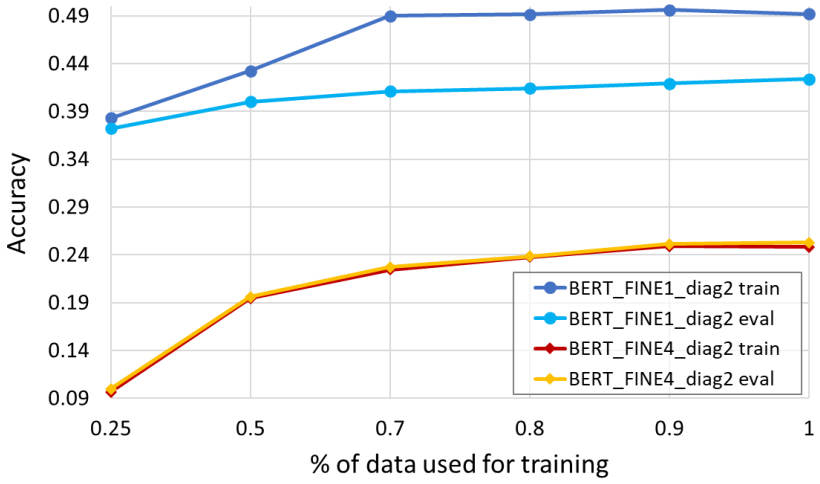


Figure 17. Training and evaluation accuracies of `BERT_FINE1_diag2` and `BERT_FINE4_diag2` with different amounts of training data

As we can see from Figure 17, while adding more data, evaluation accuracy had already stabilized for `BERT_FINE4_diag2` when 90% of the data was used for training, and was close to stabilizing for `BERT_FINE1_diag2` at the end of the experiment. Also training accuracies remained constant (starting from 70% of data for `BERT_FINE1_diag2` and from 90% for `BERT_FINE4_diag2`), which also shows that the model was not able to learn with more data. We can conclude that adding more data would not have helped increase performance of the models.

6. Conclusions

The aim of this thesis was to evaluate the suitability of applying BERT to patient treatment trajectories and to compare its performance with word2vec and classifiers built on top of it. After extracting treatment trajectories from medical invoices, the models in this thesis were built in two stages. In the first stage, four BERT models of different dimensions were pre-trained on 39 million treatment trajectories, and their performance in mask prediction task was compared with word2vec models trained on the same inputs. In the second stage, the pre-trained BERT models were fine-tuned on 45,000 trajectories for classifying seven features (diagnosis, treatment types, discharge status, type of service provider and which one of the largest four or other service providers it was, and if the invoice was for emergency services). Their performance in those tasks was compared with KNN and SVM models built on word2vec vectors.

Results showed that the performance of BERT in the pre-training stage was remarkably poor compared to word2vec, with its best MLM accuracy being 0.0055, while for word2vec it was 0.0869. Still, the accuracies of pre-trained BERT models were still around 10 times higher than accuracy of random guess (except for the smallest model for which the difference was smaller). Also, PCA performed on embeddings of 10 selected services showed that the services formed somewhat logical clusters, indicating that the model had at least learnt something. However, the clusters were very wide and diffuse and heavily overlapped with each other, which might explain the poor performance. Results also showed that BERT models underfitted, the most likely reason for it being that the default input parameters used were not appropriate for treatment trajectories. Another likely reason was that the order in which medical services appear on an invoice is not as strongly fixed as word order in natural language, which might explain why word2vec, which does not take word order into account, performed better. Breakdown of mask prediction accuracies per trajectory length showed that all models made better predictions for longer trajectories, with accuracy with the longest trajectories (81-126 services) being 0.031 and 0.333 for the best BERT and word2vec model, respectively. Performance of all models was clearly affected by weak results on short trajectories, since they made up the majority of the data.

Despite poor performance of BERT in the pre-training stage, fine-tuned BERT models outperformed classifiers built on top of word2vec in all classification tasks. Therefore, it was concluded that the contextual embeddings of BERT were better than the non-contextual ones of word2vec in classifying treatment trajectories. The highest accuracy (0.9918) was achieved in classifying treatment types (5 classes) and the lowest (0.4121) in classifying diagnosis (174 classes). It was concluded that BERT indeed proved useful with this type of non-natural language input data. From among the four BERT models built in this thesis, the second largest was the overall best, showing that if the 'language' used is simpler than natural language, then BERT models with reduced dimensions might work better. The results were further analysed by looking at confusion matrices of the best fine-tuned BERT model, which showed that it was better at predicting larger classes and therefore could have benefitted from oversampling. Also, data-bound check of two fine-tuned BERT models, where performance was measured after using different amounts of training data, showed that having more training data would not have helped increase performance of BERT classification models.

References

[1] Devlin, J.; Chang, M. W.; Lee, K.; Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, volume 1 *(Long and Short Papers)*. 2019. pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[2] Ji, Y.; Zhou, Z.; Liu, H.; Davuluri, R. V. DNABERT: pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in genome. *Bioinformatics*, Volume 37, Issue 15, August 2021, pp. 2112–2120. DOI: 10.1093/bioinformatics/btab083.

[3] Min, S.; Park, S.; Kim, S.; Choi, H. S.; Lee, B.; Yoon, S. Pre-Training of Deep Bidirectional Protein Sequence Representations With Structural Information. In: *IEEE Access*, vol. 9, pp. 123912-123926, 2021, DOI: 10.1109/ACCESS.2021.3110269.

[4] Li, Y.; Rao, S.; Solares, J. R. A.; Hassaine, A.; Ramakrishnan, R.; Canoy, D.; Zhu, Y.; Rahimi, K.; Salimi-Khorshidi, K. BEHRT: Transformer for Electronic Health Records. *Scientific Reports*, 10, 7155 (2020). DOI: 10.1038/s41598-020-62922-y.

[5] Shang, J.; Ma, T.; Xiao, C.; Sun, J. Pre-training of Graph Augmented Transformers for Medication Recommendation. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, pp. 5953–5959. arXiv:1906.00346v2 [cs.AI]. 2019. DOI: 10.48550/arXiv.1906.00346.

[6] Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781 [cs.CL], 7 Sept. 2013.

[7] Jurafsky, D.; Martin, J. H. Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Third edition draft. January 2022. https://web.stanford.edu/~jurafsky/slp3/ed3book_jan122022.pdf (04.05.2023).

[8] Huang, Z.; Liang, D.; Xu, P.; Xiang, B. Improve Transformer Models with Better Relative Position Embeddings. arXiv:2009.13658v1 [cs.CL]. 2020. DOI: 10.48550/arXiv.2009.13658.

[9] Liu, Z.; Lin, J.; Sun, M. Representation Learning for Natural Language Processing. 2020. Springer (e-book). DOI: 10.1007/978-981-15-5573-2.

[10] Goldberg, Y. Neural network models for language processing. In: *Synthesis Lectures on Human Language Technologies series*. Hirst, G. (Ed.). 2017. Morgan & Claypool Publishers. DOI: 10.2200/S00762ED1V01Y201703HLT037.

[11] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017. pp. 6000–6010.

[12] Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L. Deep Contextualized Word Representations. In: Proceedings of the 2018 Conference of the North American Chapter of *the Association for Computational Linguistics (NAACL): Human Language Technologies*, Volume 1 *(Long Papers)*, 2018, pp. 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics. DOI: 10.18653/v1/N18-1202.

[13] Radford, A.; Narasimhan, N.; Salimans, T.; Sutskever, I. Improving Language Understanding by Generative Pre-Training. Preprint, OpenAI, 2018. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (04.05.2023).

[14] Wu, Y.; Schuster, M.; Chen, Z.; Le, Q. V.; Norouzi, M.; Macherey, M.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; Klingner, J.; Shah, A.; Johnson, M.; Liu, Z.; Kaiser, Ł.; Gouws, S.; Kato, Y.; Kudo, T.; Kazawa, H.; Stevens, K.; Kurian, G.; Patil, N.; Wang, W.; Young, C.; Smith, J.; Riesa, J.; Rudnick, A.; Vinyals, O.; Corrado, G.; Hughes, M.; Dean, J. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv:1609.08144 [cs.CL] (2016). DOI: 10.48550/arXiv.1609.08144.

[15] World Health Organization, International Statistical Classification of Diseases and Related Health Problems, 10th Revision (ICD10). https://icd.who.int/browse10/2019/en (04.05.2023).

[16] Flach, P. Machine Learning: The Art and Science of Algorithms that Make Sense of Data. 2012. New York: Cambridge University Press.

[17] Kelleher, J. D.; Mac Namee, B.; D'Arcy, A. Fundamentals of Machine Learning for Predictive Data Analytics. Algorithms, Worked Examples, and Case Studies. 2015. Cambridge, Massachusetts, London, England: The MIT Press.

[18] Bengfort, B.; Bilbro, R.; Ojeda, T. Applied Text Analysis with Python. Enabling Language-Aware Data Products with Machine Learning. 2018. First edition. Beijing, Boston, Farnham, Sebastopol, Tokyo. O-Reilly.

[19] Müller, M.; Salathé, M.; Kummervold, P. E. COVID-Twitter-BERT: A Natural Language Processing Model to Analyse COVID-19 Content on Twitter. 2020. arXiv:2005.07503v1 [cs.CL]. DOI: 10.48550/arXiv.2005.07503.

[20] Yu, W.; Jiang, Z.; Chen, F.; Hou, Q.; Feng, J. LV-BERT: Exploiting Layer Variety for BERT. arXiv:2106.11740v2 [cs.CL] (2021). DOI: 10.48550/arXiv.2106.11740.

[21] Feng, Y.; Zhou, M.; Tong, X. Imbalanced classification: a paradigm-based review. arXiv:2002.04592v2 [stat.ME] (2021). DOI: 10.48550/arXiv.2002.04592.

Appendix

I. GitLab repository

https://gitlab.ut.ee/renata.siimon/patient-treatment-trajectories (private repository)

II. Discharge statuses before and after aggregation

Discharge status (before aggregation)	%	Discharge status (aggregated)	%
Recovery	41.36%	Recovery	41.36%
Other causes	25.32%	Other	25.48%
Leaving at own risk against recommendation by the doctor	0.12%		
Missing or erroneous invoice ending code	0.04%		
Called back to outpatient visit to the same specialist doctor in the same medical service provider	14.18%	Referral to outpatient treatment	22.82%
Referred to outpatient visit to a specialist doctor in another medical service provider	5.74%		
Referred to outpatient visit of another specialist doctor in the same medical service provider	2.90%		
Referred to visiting and monitoring by family doctor	8.16%	Referral to family doctor	8.16%
Referred to inpatient treatment in the same medical service provider	0.89%	Referral to inpatient treatment	1.51%
Referred to inpatient treatment in another medical service provider (regional hospital)	0.20%		
Referred to inpatient treatment in another medical service provider (general hospital)	0.19%		
Referred to inpatient treatment in another medical service provider (central hospital)	0.15%		
Referred to inpatient treatment in another medical service provider (except general, central or regional hospital)	0.08%		
Referred to day care in the same medical service provider	0.39%	Referral to day care	0.48%
Referred to day care in another medical service provider	0.09%		
Deceased	0.20%	Deceased	0.20%

III. Diagnosis categories (18)

A-B	Certain infectious and parasitic diseases
C-D	Neoplasms
E	Endocrine, nutritional and metabolic diseases
F	Mental and behavioural disorders
G	Diseases of the nervous system
H	Diseases of the eye and adnexa
I	Diseases of the circulatory system
J	Diseases of the respiratory system
K	Diseases of the digestive system
L	Diseases of the skin and subcutaneous tissue
M	Diseases of the musculoskeletal system and connective tissue
N	Diseases of the genitourinary system
O	Pregnancy, childbirth and the puerperium
P	Certain conditions originating in the perinatal period
Q	Congenital malformations, deformations and chromosomal abnormalities
R	Symptoms, signs and abnormal clinical and laboratory findings, NEC
S-T	Injury, poisoning and certain other consequences of external causes
Z	Factors influencing health status and contact with health services

IV. Diagnosis categories (174)*

A00-A09	C76-C80	G20-G26	I26-I28	L60-L75	O00-O08	S10-S19
A15-A19	C81-C96	G30-G32	I30-I52	L80-L99	O10-O16	S20-S29
A30-A49	D00-D09	G35-G37	I60-I69	M00-M03	O20-O29	S30-S39
A50-A64	D10-D36	G40-G47	I70-I79	M05-M14	O30-O48	S40-S49
A65-A69	D37-D48	G50-G59	I80-I89	M15-M19	O60-O75	S50-S59
B00-B09	D50-D53	G60-G64	J00-J06	M20-M25	O80-O84	S60-S69
B15-B19	D60-D64	G80-G83	J09-J18	M30-M36	O95-O99	S70-S79
B20-B24	D65-D69	G90-G99	J20-J22	M40-M43	P05-P08	S80-S89
B25-B34	D70-D77	H00-H06	J30-J39	M45-M49	P50-P61	S90-S99
B35-B49	D80-D89	H10-H13	J40-J47	M50-M54	Q10-Q18	T00-T07
B65-B83	E00-E07	H15-H22	J95-J99	M60-M63	Q20-Q28	T15-T19
B85-B89	E10-E14	H25-H28	K00-K14	M65-M68	Q60-Q64	T20-T25
B90-B94	E20-E35	H30-H36	K20-K31	M70-M79	Q65-Q79	T36-T50
B95-B97	E50-E64	H40-H42	K35-K38	M80-M85	Q80-Q89	T51-T65
C00-C14	E65-E68	H43-H45	K40-K46	M91-M94	Q90-Q99	T66-T78
C15-C26	E70-E90	H46-H48	K50-K52	N00-N08	R00-R09	T80-T88
C30-C39	F00-F09	H49-H52	K55-K63	N10-N16	R10-R19	T90-T98
C43-C44	F10-F19	H53-H54	K70-K77	N17-N19	R20-R23	Z00-Z13
C45-C49	F20-F29	H60-H62	K80-K87	N20-N23	R25-R29	Z20-Z29
C50-C50	F30-F39	H65-H75	K90-K93	N25-N29	R30-R39	Z30-Z39
C51-C58	F40-F49	H80-H83	L00-L08	N30-N39	R40-R46	Z40-Z54
C60-C63	F50-F59	H90-H95	L20-L30	N40-N51	R47-R49	Z70-Z76
C64-C68	F70-F79	I05-I09	L40-L45	N60-N64	R50-R69	Z80-Z99
C69-C72	F80-F89	I10-I15	L50-L54	N70-N77	R70-R79	Other
C73-C75	F90-F98	I20-I25	L55-L59	N80-N98	S00-S09	

* For the meanings of the diagnosis codes, see ICD10 [15].

V. Breakdown of types of service providers

1) Central hospitals:
East Tallinn Central Hospital
West Tallinn Central Hospital
Ida-Viru Central Hospital
Pärnu Hospital

2) General hospitals:
Hiiumaa Hospital
Järvamaa Hospital
Kuressaare Hospital
Läänemaa Hospital
South-Estonian Hospital
Narva Hospital
Põlva Hospital
Rakvere Hospital
Rapla County Hospital
Raplamaa Hospital
Valga Hospital
Viljandi Hospital

3) Regional hospitals:
Tartu University Hospital
North Estonia Medical Centre
Tallinn Children's Hospital

4) Local hospital:
Jõgeva Hospital

5) Chosen partner:
1241 smaller medical service providers

VI. Most frequent services

Service code	Service	% of services	Subtype
9002	Repeated visit to family doctor	6.57%	Treatment by family doctor
3002	Initial visit to specialist doctor	5.96%	Outpatient visits
9001	Initial visit to family doctor	5.12%	Treatment by family doctor
3004	Repeated visit to specialist doctor	4.61%	Outpatient visits
66102	Creatinine, urea, uric acid	3.66%	Lab tests
66106	Enzymes: ALP, ASAT, ALAT, LDH, CK, GGT, CK-Mba, alpha-amylase	3.43%	Lab tests
66707	Determination of anaemia, heart, tumour markers, studies of pathogens, determination of antibodies, vitamins and enzymes with immune method	3.21%	Lab tests
9018	Patient consultancy by family doctor conducted over the phone and documented in patient's health record	2.88%	Treatment by family doctor
66706	Screening, hormone and pathogen tests with immune method	2.64%	Lab tests
66112	C-reactive protein	2.58%	Lab tests

VII. Service subtypes

Id	Subtype	No of services	% in data
4000	Surgeries	1,011	0.479%
7000	Tests and procedures	716	15.941%
6000	Lab tests	275	42.467%
2200	Accessory equipment during surgery	177	0.125%
1700	Drugs	163	0.239%
9000	Treatment by family doctor	135	24.014%
8	Musculoskeletal and connective tissue diseases	96	0.099%
5	Circulatory diseases	71	0.095%
5000	Dental care	71	0.032%
6	Diseases of the digestive organs	65	0.070%
2000	Bed days	61	1.191%
3	Ear, nose, mouth and throat diseases	60	0.070%
1	Nervous diseases	56	0.063%
1000	Outpatient visits	54	13.740%
11	Diseases of the kidneys and urinary tract	51	0.030%
1500	Blood and blood products	46	0.157%
19	Mental diseases	41	0.037%
2	Eye diseases	34	0.059%
4	Respiratory diseases	34	0.044%
9	Diseases of the skin, subcutaneous tissue and mammary gland	34	0.035%
7	Diseases of the liver, biliary tract and pancreas	31	0.026%
1200	School health care	30	0.068%
13	Diseases of female reproductive organs	30	0.044%
12	Diseases of the male reproductive organs	28	0.011%
10	Endocrine, nutritional and metabolic diseases	27	0.015%
21	Injuries, poisonings and toxic effects of drugs	27	0.007%
14	Pregnancy, childbirth and the postpartum period	26	0.081%
17	Myeloproliferative diseases, poorly differentiated tumours	25	0.030%
3000	Complex prices	20	0.064%
5400	Orthodontics	17	0.000%
2400	Anaesthesia	16	0.516%
18	Infectious and parasitic diseases	16	0.013%
15	Neonatal DRGs	15	0.009%
23	Factors affecting health and other contacts with health services	13	0.051%
16	Diseases of blood and hematopoietic organs and diseases related to immune mechanisms	12	0.008%
30	Breast conditions	12	0.001%
22	Burns	11	0.001%
5300	Dentures	8	0.000%
1400	Transport	7	0.053%
24	Severe multiple trauma	7	0.001%
99	Non-specific or incorrect information	5	0.003%
20	Abuse or addiction to alcohol and psychoactive substances	3	0.001%
0	Pre-MDC DRGs	2	0.000%
8000	Rehabilitation	1	0.007%
25	HIV infection	1	0.000%

VIII. Most frequent treatment trajectories

Treatment trajectory	Meaning of service codes in trajectory	Times repeated
9001 9002	Initial visit to family doctor → Repeated visit to family doctor	1,342,397
3002 7263 7267	Initial visit to specialist doctor → Eye fundus examination with three-mirrored lens or Volke magnifying glass → Refraction examination of eyes with autorefractometer	389,725
3002 7267 7263	Initial visit to specialist doctor → Refraction examination of eyes with autorefractometer → Eye fundus examination with three-mirrored lens or Volke magnifying glass	287,014
3002 3004	Initial visit to specialist doctor → Repeated visit to specialist doctor	244,689
3002 7954	Initial visit to specialist doctor → Vaginal ultrasound examination	240,247
9001 9002 9002	Initial visit to family doctor → Repeated visit to family doctor → Repeated visit to family doctor	234,376
9061 9062	Consultancy visit to family nurse → Nursing procedure by family nurse (manual activity)	222,047
3002 7263	Initial visit to specialist doctor → Refraction examination of eyes with autorefractometer	219,273
9018 9064	Patient consultancy by family doctor conducted over the phone and documented in patient's health record → Patient consultancy by family nurse conducted over the phone and documented in patient's health record	209,565
9002 9061	Repeated visit to family doctor → Consultancy visit to family nurse	172,955

IX. Parameters used for pre-training BERT models*

Parameter	Meaning	BERT_PRE1	BERT_PRE2	BERT_PRE3	BERT_PRE4
vocab_size	Size of vocabulary mapping service codes to their token ids	3,440	3,440	3,440	3,440
type_vocab_size	Number of sentences in each input	1	1	1	1
max_position_embeddings	Max length of input trajectory	128	128	128	128
num_attention_heads	Number of attention heads	12	6	4	3
num_hidden_layers	Number of hidden layers	12	6	4	3
hidden_size	Size of hidden layers (encoder layers and pooler layer)	768	384	192	96
intermediate_size	Size of intermediate (i.e. feed-forward) layer in the encoder	3,072	1,536	768	384
per_device_train_batch_size	Batch size	32	32	32	32
max_steps	Number of training steps	700,320 (4 epochs)	700,320 (4 epochs)	700,320 (4 epochs)	700,320 (4 epochs)
learning_rate	Learning rate	1e-4	1e-4	1e-4	1e-4
warmup_ratio	Ratio of training steps used for a linear warmup from 0 to learning_rate	0.01	0.01	0.01	0.01

* Other parameter values used were the default ones specified in Transformers API (version 4.7.0) for BERTForMaskedLM[9].

X. Number of epochs for fine-tuning BERT models

Classification task	Model			
	BERT_FINE1	BERT_FINE2	BERT_FINE3	BERT_FINE4
diag	4	4.25	4.5	4.5
diag2	4.75	4.5	5	4.5
emergency_bill	2	2	3	3.75
discharge_status	3.5	5	4	5
treatment_type	4	4.25	4.75	2.75
tto_type	4	4	4.75	4.25
largest4	4.5	4	4.5	5

XI. Parameter values in KNN and SVM models*

Model	Classification task	Parameter values				
		KNN	SVM			
		n_neigh-bours	eta0	power_t	alpha	penalty
KNN_w5_v400 and SVM_w5_v400	diag	15	0.032	0.45	0.00014	l1
	diag2	17	0.033	0.50	0.00025	l1
	emergency_bill	3	0.035	0.40	0.00015	l1
	discharge_status	35	0.035	0.40	0.00020	l1
	treatment_type	3	0.035	0.20	0.00012	l1
	tto_type	7	0.033	0.30	0.00020	l2
	largest4	15	0.035	0.20	0.00015	l1
KNN_w7_v150 and SVM_w7_v150	diag	11	0.035	0.55	0.00020	l1
	diag2	21	0.033	0.50	0.00025	l1
	emergency_bill	7	0.035	0.35	0.00020	l1
	discharge_status	15	0.033	0.35	0.00015	l1
	treatment_type	5	0.035	0.20	0.00012	l1
	tto_type	5	0.035	0.30	0.00010	l2
	largest4	15	0.035	0.50	0.00015	l1
KNN_w7_v600 and SVM_w7_v600	diag	15	0.032	0.55	0.00015	l1
	diag2	21	0.033	0.40	0.00020	l1
	emergency_bill	3	0.031	0.50	0.00020	l1
	discharge_status	15	0.035	0.45	0.00022	l1
	treatment_type	3	0.031	0.25	0.00015	l1
	tto_type	5	0.031	0.50	0.00010	l2
	largest4	11	0.033	0.40	0.00015	l1

* Other parameter values used were the default ones specified in Scikit-learn API[2] (version 0.24.2).

XII. Performance of KNN models*

Task	Metric	TRAIN			TEST		
		KNN_w5_v400	KNN_w7_v600	KNN_w7_v150	KNN_w5_v400	KNN_w7_600	KNN_w7_v150
diag (18)	Accuracy	0.5914	0.5911	0.5990	**0.5471**	**0.5478**	**0.5436**
	Precision	0.5927	0.5957	0.5996	**0.5442**	**0.5461**	**0.5419**
	Recall	0.5914	0.5911	0.5990	**0.5471**	**0.5478**	**0.5436**
	F-score	0.5810	0.5802	0.5884	**0.5344**	**0.5352**	**0.5310**
diag2 (174)	Accuracy	0.4166	0.4069	0.4063	**0.3645**	0.3620	**0.3670**
	Precision	0.4260	0.4141	0.4137	**0.3712**	0.3644	**0.3717**
	Recall	0.4166	0.4069	0.4063	**0.3645**	0.3620	**0.3670**
	F-score	0.3877	0.3745	0.3739	**0.3330**	0.3259	**0.3308**
emergency_bill (2)	Accuracy	0.9670	0.9670	0.9590	0.9521	0.9519	**0.9557**
	Precision	0.9663	0.9663	0.9578	0.9508	0.9507	**0.9542**
	Recall	0.9670	0.9670	0.9590	0.9521	0.9519	**0.9557**
	F-score	0.9665	0.9664	0.9581	0.9513	0.9511	**0.9545**
discharge_status(7)	Accuracy	0.6136	0.6340	0.6277	**0.5944**	0.5901	**0.5893**
	Precision	0.6014	0.6206	0.6133	**0.5824**	0.5731	**0.5701**
	Recall	0.6136	0.6340	0.6277	**0.5944**	0.5901	**0.5893**
	F-score	0.5986	0.6195	0.6122	**0.5790**	0.5745	**0.5716**
treatment_type (5)	Accuracy	0.9898	0.9896	0.9863	**0.9842**	**0.9851**	0.9836
	Precision	0.9898	0.9896	0.9863	**0.9842**	**0.9851**	0.9836
	Recall	0.9898	0.9896	0.9863	**0.9842**	**0.9851**	0.9836
	F-score	0.9897	0.9896	0.9863	**0.9841**	**0.9850**	0.9835
tto_type (5)	Accuracy	0.8027	0.8150	0.8133	**0.7468**	**0.7458**	**0.7458**
	Precision	0.7962	0.8070	0.8067	**0.7399**	**0.7318**	**0.7358**
	Recall	0.8027	0.8150	0.8133	**0.7468**	**0.7458**	**0.7458**
	F-score	0.7972	0.8088	0.8082	**0.7404**	**0.7377**	**0.7400**
largest4 (5)	Accuracy	0.7947	0.8017	0.7931	0.7633	**0.7683**	**0.7661**
	Precision	0.7709	0.7802	0.7680	0.7310	**0.7382**	**0.7324**
	Recall	0.7947	0.8017	0.7931	0.7633	**0.7683**	**0.7661**
	F-score	0.7760	0.7854	0.7733	0.7410	**0.7479**	**0.7423**

* In the *'Task'* column, the number of classes of the predicted feature is shown in brackets. Best models in each classification task are marked in bold. If several models are marked in bold for the same task, it means their performance was not statistically significantly different based on McNemar test with significance level 0.05. The model chosen for comparison with BERT models is underlined.

XIII. Performance of SVM models*

Task	Metric	TRAIN			TEST		
		SVM_w5_v400	SVM_w7_v600	SVM_w7_v150	SVM_w5_v400	SVM_w7_v600	SVM_w7_v150
diag (18)	Accuracy	0.5498	0.5528	0.5258	0.5415	**0.5475**	0.5266
	Precision	0.5494	0.5488	0.5216	0.5386	**0.5417**	0.5175
	Recall	0.5498	0.5528	0.5258	0.5415	**0.5475**	0.5266
	F-score	0.5377	0.5382	0.5080	0.5272	**0.5311**	0.5046
diag2 (174)	Accuracy	0.3473	0.3511	0.3321	0.3358	**0.3445**	0.3255
	Precision	0.3433	0.3738	0.3274	0.3365	**0.3720**	0.3233
	Recall	0.3473	0.3511	0.3321	0.3358	**0.3445**	0.3255
	F-score	0.3217	0.3331	0.3074	0.3086	**0.3241**	0.2964
emergency_bill (2)	Accuracy	0.9477	0.9457	0.9458	**0.9529**	0.9504	**0.9526**
	Precision	0.9455	0.9432	0.9433	**0.9510**	0.9482	**0.9506**
	Recall	0.9477	0.9457	0.9458	**0.9529**	0.9504	**0.9526**
	F-score	0.9459	0.9436	0.9437	**0.9508**	0.9481	**0.9505**
discharge_status(7)	Accuracy	0.5541	0.5690	0.5548	0.5521	**0.5652**	0.5550
	Precision	0.5388	0.5488	0.5339	0.5385	**0.5373**	0.5338
	Recall	0.5541	0.5690	0.5548	0.5521	**0.5652**	0.5550
	F-score	0.5359	0.5435	0.5303	0.5326	**0.5389**	0.5297
treatment_type (5)	Accuracy	0.9892	0.9887	0.9863	**0.9876**	**0.9869**	0.9851
	Precision	0.9893	0.9888	0.9864	**0.9876**	**0.9869**	0.9851
	Recall	0.9892	0.9887	0.9863	**0.9876**	**0.9869**	0.9851
	F-score	0.9892	0.9887	0.9862	**0.9876**	**0.9869**	0.9851
tto_type (5)	Accuracy	0.7182	0.7172	0.7096	**0.7183**	**0.7160**	0.7077
	Precision	0.7105	0.7129	0.7083	**0.7120**	**0.7112**	0.7063
	Recall	0.7182	0.7172	0.7096	**0.7183**	**0.7160**	0.7077
	F-score	0.7019	0.7028	0.7018	**0.7027**	**0.7017**	0.6998
largest4 (5)	Accuracy	0.7382	0.7425	0.7306	**0.7338**	**0.7369**	**0.7320**
	Precision	0.6978	0.6804	0.6424	**0.6900**	**0.6650**	**0.6468**
	Recall	0.7382	0.7425	0.7306	**0.7338**	**0.7369**	**0.7320**
	F-score	0.7094	0.6894	0.6537	**0.7032**	**0.6813**	**0.6535**

* In the *'Task'* column, the number of classes of the predicted feature is shown in brackets. Best models in each classification task are marked in bold. If several models are marked in bold for the same task, it means their performance was not statistically significantly different based on McNemar test with significance level 0.05. The model chosen for comparison with BERT models is underlined.

XIV. Performance of fine-tuned BERT models on training set*

Task	Metric	BERT_FINE1	BERT_FINE2	BERT_FINE3	BERT_FINE4
diag (18)	Accuracy	0.6551	**0.6670**	0.6190	0.5628
	Precision	0.6661	**0.6783**	0.6300	0.5900
	Recall	0.6551	**0.6670**	0.6190	0.5628
	F-score	0.6479	**0.6584**	0.6068	0.5320
diag2 (174)	Accuracy	**0.4920**	0.4647	0.3851	0.2483
	Precision	0.5367	0.5201	0.5355	**0.6325**
	Recall	**0.4920**	0.4647	0.3851	0.2483
	F-score	**0.4597**	0.4216	0.3105	0.1475
emergency_bill (2)	Accuracy	0.9645	**0.9669**	0.9649	0.9596
	Precision	0.9636	**0.9661**	0.9640	0.9583
	Recall	0.9645	**0.9669**	0.9649	0.9596
	F-score	0.9636	**0.9663**	0.9641	0.9584
discharge_status (7)	Accuracy	0.6368	**0.7130**	0.6492	0.6060
	Precision	0.6296	**0.7102**	0.6381	0.6067
	Recall	0.6368	**0.7130**	0.6492	0.6060
	F-score	0.6239	**0.7056**	0.6370	0.5909
treatment_type (5)	Accuracy	0.9944	**0.9955**	0.9948	0.9908
	Precision	0.9944	**0.9955**	0.9948	0.9909
	Recall	0.9944	**0.9955**	0.9948	0.9908
	F-score	0.9944	**0.9955**	0.9948	0.9908
tto_type (5)	Accuracy	0.8478	**0.8574**	0.8220	0.7565
	Precision	0.8539	**0.8609**	0.8180	0.7702
	Recall	0.8478	**0.8574**	0.8220	0.7565
	F-score	0.8460	**0.8555**	0.8176	0.7569
largest4 (5)	Accuracy	0.8910	**0.8959**	0.8528	0.7874
	Precision	0.8869	**0.8919**	0.8405	0.7502
	Recall	0.8910	**0.8959**	0.8528	0.7874
	F-score	0.8846	**0.8908**	0.8415	0.7552

* In the *'Task'* column, the number of classes of the predicted feature is shown in brackets. Best result in each row is marked in bold. Colour scaling has been applied separately for each classification task.

XV. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, Õie Renata Siimon,

1. grant the University of Tartu a free permit (non-exclusive licence) to:

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis

“Patient treatment trajectories using vector embeddings”,

supervised by Sven Laur.

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work from ***09.05.2023*** until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in points 1 and 2.

4. I confirm that granting the non-exclusive licence does not infringe other persons’ intellectual property rights or rights arising from the personal data protection legislation.

Õie Renata Siimon
09.05.2023