

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS PHYSICS AND INFORMATICS



PREDICTION OF HEALTH STATUS DETERIORATION

Master thesis

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS PHYSICS AND INFORMATICS



PREDICTION OF HEALTH STATUS DETERIORATION

Master thesis

Study program: Applied informatics
Branch of study: Applied informatics
Department: Department of Applied Informatics
Supervisor: MSc. František Dráček
Consultant:



ZADANIE ZÁVEREČNEJ PRÁCE

Typ záverečnéj práce: diplomová
Jazyk záverečnéj práce: slovenský
Sekundárny jazyk: anglický

Názov: Predikcia zhoršenia zdravotného stavu
Prediction of Health Status Deterioration

Anotácia: V súčasnosti sa sektor zdravotníctva na Slovensku vyznačuje nízkou mierou využitia dostupných zdravotníckych dát. V rámci tejto práce je cieľom ukázať, že z existujúcich dát je možné predikovať vývoj ďalšieho zdravotného stavu pacienta, poprípade odhadnúť vývoj budúcich nákladov za účelom lepšieho plánovania prerozdelenia financií v rámci sektoru.

Cieľ: Práca bude rozdelená na dve časti, v prvej študent urobí teoretické zhnutie existujúcich metód spracovania dát a metód strojového učenia, ktoré sa budú dať potenciálne aplikovať na daný problém. V druhej časti navrhne a aplikuje predíchný model.

Literatúra: T. Sk, L. M. G, L. R. K and R. R. J, "Health Status Prediction using ML Techniques," 2022 6th International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2022, pp. 1191-1196, doi: 10.1109/ICCMC53470.2022.9753766.

Jödicke, A.M., Zellweger, U., Tomka, I.T. et al. Prediction of health care expenditure increase: how does pharmacotherapy contribute?. BMC Health Serv Res 19, 953 (2019). <https://doi.org/10.1186/s12913-019-4616-x>

Vedúci: MSc. František Dráček
Konzultant: Ing. Lukáš Palaj
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 05.10.2023

Dátum schválenia: prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

I hereby declare that I have written this thesis by myself, only with help of referenced literature, under the careful supervision of my thesis advisor.

Bratislava, 2025

.....
Bc. Marián Kravec

Acknowledgment

WRITE ACKNOWLEDGMENT

Abstract

ABSTRACT EN

Keywords: TODO

Abstrakt

ABSTRACT SK

Kľúčové slová: TODO

Contents

1	Introduction	2
2	Similar studies	3
3	Medical data	5
3.1	Patients records	5
3.1.1	Records of medical procedures from ambulatory health care . .	5
3.1.2	Records of prescribed medicines	6
3.2	Mappings	6
4	Proposed method	8
4.1	Embedding of patient	8
4.1.1	Timestamp	8
4.1.2	Diagnosis embedding	9
4.1.3	Drug embedding	11
4.1.4	Medical procedure embedding	12
4.2	Prediction of future number of records	16
4.3	Future record prediction	16
4.3.1	Recurrent neural network	17
4.3.2	Elman RNN	17
4.3.3	Gated recurrent unit	18
4.3.4	Long short-term memory	19
4.3.5	Transformer	20
4.4	Record cost prediction	22
4.4.1	Multilayer perceptron	22
5	Software design	24
6	Implementation	25
6.1	Embedding of patient	25
6.1.1	Timestamp	25
6.1.2	Diagnosis embedding	25

6.1.3	Drug embedding	26
6.1.4	Medical procedure embedding	27
7	Research	29
7.1	Embedding of patient	29
7.1.1	Diagnosis embedding	29
7.1.2	Drug embedding	30
7.1.3	Medical procedure embedding	32
8	Results	33

List of Figures

4.1	Structure of MKCH-10 code	9
4.2	Fourteen main anatomical or pharmacological groups and their corresponding first level ATC code [1]	11
4.3	Architecture of Elman RNN [6]	18
4.4	Architecture of GRU hidden layer	19
4.5	Architecture of LSTM hidden layer	20
4.6	Architecture of one encoder-decoder block in transformer model from original "Attention is all you need" paper [22]	21
4.7	Architecture of multilayer perceptron [17]	23
6.1	Showcase of resulting embedding of specific diagnosis (rounded to two decimal places)	26

List of Tables

6.1	Lengths of random vectors assigned to each information level of ATC code	27
7.1	Similarities of embedding of multiple chosen MKCH-10 codes	29
7.2	Similarities of embedding of multiple chosen ATC codes	31

Terminology

Terms

Abbreviations

- **CPT** - Current Procedural Terminology.
- **EHR** - Electronic Health Records.
- **ICD-10-CM** - International Classification of Diseases, Tenth Revision, Clinical Modification.
- **MKCH-10** - International Classification of Diseases, Tenth Revision (Medzinárodná klasifikácia chorôb).
- **ATC** - Anatomical Therapeutic Chemical.
- **LLM** - Large language model.
- **LaBSE** - Language-agnostic BERT sentence embedding model.
- **BERT** - Bidirectional encoder representations from transformers.
- **KNN** - K-nearest neighbors algorithm.
- **FFNN** - Feed forward neural network.
- **MLP** - Mutlilayer perceptron.
- **MLM** - Masked Language Model.
- **NSP** - Next Sentence Prediction.
- **TLM** - Translation Language Model.
- **PCA** - Principal component analysis.
- **RNN** - Recurrent neural network
- **SRN** - Simple recurrent network
- **GRU** - Gated recurrent unit RNN
- **LSTM** - Long short-term memory RNN

Motivation

Being able to accurately predict cost of patient in next period is important for both government and health insurance company.

Chapter 1

Introduction

Chapter 2

Similar studies

One of sub-task for prediction of patient future is to group medical procedures into clusters because there are many procedures that even though have different codes they are essentially same or similar enough that leaving them separate would only cause issue for predicting model.

For this task Lorenzi et al. from Duke University in Durham developed novel algorithm called Predictive Hierarchical Clustering [18]. This algorithm was developed for agglomerative clustering of surgical CPT codes. This algorithm uses one-pass bottom-up approach where they utilize EHR, more precisely using 317 predictors like lab values and patients history, excluding CPT information for 3,723,252 patients and 3,132 CPT codes where each patient have one main surgical CPT code. For each CPT code then they create tree containing patients with that code. Then at each iteration, the algorithm considers merging all pairs of existing trees. To compare two trees they utilize two hypothesis, first hypothesis say that data in both trees are generated from same model, while second say data in each tree is generated from models with different parameters. Final value is weighted average of probabilities of these two hypothesis considering data in trees, where weight is probability of first hypothesis 2.1.

$$p(D_k|T_k) = p(H_1^k)p(D_k|H_1^k) + (1 - p(H_1^k))p(D_i|T_i)p(D_j|T_j) \quad (2.1)$$

Where D_k is set of data in merged tree (merged T_i and T_j), T_k is merged tree, H_1^k is first hypothesis, D_i and D_j are data in trees T_i and T_j .

From perspective of prediction of patient future one of similar studies is study called Deep Patient by Riccardo Miotto et al. [19] where they were predicting which disease would patient have in the future based on his current state. Their input data were contained general demographic details such as age, gender and race, and common clinical descriptors such as diagnoses, medications, procedures, and lab tests. To predict future diseases they use random forest model with one-vs.-all learning. Study focus primarily on improving results of model by reducing noise in data by reducing their

dimensionality. They compared standard approaches like principle component analysis, Gaussian mixture model or K-means, but main focus was approach using stack of denoising autoencoders. Model using stack of denoising autoencoders to reduce dimension showed significantly better results compared to both model using original dataset and models using other dimensionality reduction techniques.

Another similar study, is study by Caballer-Tarazona et al. [8] in which they tried to predict future cost of the patient primarily using what they called "Aggregated Clinical Risk Group 3" computed from standardized Clinical Risk Group (CRG). This variable consist of the parts, first in one of nine grouped CRGs and second part is one of six levels of severity.

Chapter 3

Medical data

To train and verify model we used anonymized data obtained from Slovak National health information center also known under abbreviation NCZI.

Data we have can be split into two categories, those are patients records and mapping table.

3.1 Patients records

Patients records are split into two dataset:

- Records of medical procedures from ambulatory health care
- Records of prescribed medicines

In total we have data about (ADD NUMBER HERE) patients.

3.1.1 Records of medical procedures from ambulatory health care

Each row of this dataset is single patient record contains information about single medical procedure done to them, in total this dataset consist of (ADD NUMBER HERE) records. Each record consist of these variable:

- date of the procedure - date when procedure was performed
- code of the patient - identification code unique for the patient
- age of the patient - age of the patient at the time of procedure
- gender of the patient

- code of the diagnosis - identification code unique for the diagnosis for which procedure was prescribed
- code of the procedure - identification code unique for the medical procedure
- cost of the procedure - cost associated with performing of the procedure

For our prediction we use most of these information, date of the procedure combined with patient age is used to create timestamp information used to order all records for patient as well as one of the dimension of record embedding. Identification code of patient is used to be able to gather all records for single patient. Identification codes for diagnosis and procedure are matched with their corresponding numerical vector and embedded into vector corresponding to record (see 4.1). Cost is encoded into cost category and used as information of cost associated with embedding of record.

3.1.2 Records of prescribed medicines

Similarly to dataset containing procedures, each row is single record of single prescription of drug to specific patient, in total we have (ADD NUMBER HERE) records of prescribed drugs. Each record consist of these variable:

- date of the prescription - date when drug was prescribed
- code of the patient - identification code unique for the patient
- age of the patient - age of the patient at the time of procedure
- gender of the patient
- code of the diagnosis - identification code unique for the diagnosis for which drug was prescribed
- code of the drug - identification code unique for the medical procedure
- cost of the drug - cost associated with performing of the procedure

Use of these variable is also similar to procedures, with only difference that instead of using encoding procedure into record embedding we encode drug.

3.2 Mappings

Other than datasets containing patients data we use three mapping files to map identification codes used for attributes in patients records datasets to corresponding embedding vectors. Mapping datasets are:

- Diagnosis mapping
- Drug mapping
- Medical procedure mapping

Chapter 4

Proposed method

Task of predicting future cost of a patient can be split into multiple sub-tasks which follow each other. The sub-tasks are these:

1. Embed patient history into numerical vectors
2. Compute expected number of records patient would have in next year
3. Predict future records for patient
4. Predict cost of each future record
5. Complete total cost of patient for next year

4.1 Embedding of patient

First of sub-tasks for prediction of patient future costs is to embed each patient record into numerical vector that would be understandable for neural network. For each record of a patient we embed four information. First and easy to implement is a timestamp which is computed using numerical and date values, then there three more tricky information and those are diagnosis, medical procedure and prescribed drug.

4.1.1 Timestamp

Attribute what we call timestamp of patients record can more precisely be described as approximation of patients age at the moment of either medical procedure or drug prescription. This timestamp is computed using two of available information and those are age of patient in years and date of record. Computation for single patient is done like this: we found first record of patient, meaning record with earliest date, and we take age in years of that patient in that moment and set it as a timestamp, for each next record we take age from first record and add difference between first record and

currently processed one converted to fraction of years. This way timestamp contains approximation of age of patient while also containing information about order of records and comparative timeframe between each two records.

4.1.2 Diagnosis embedding

Base diagnose information we embed was ICD-10-CM code of disease. ICD-10-CM stands for "International Classification of Diseases, Tenth Revision, Clinical Modification" and is used to code and classify medical diagnoses [9] most precisely version of this code that is used in Slovakia and is better known by the acronym MKCH-10-SK (Medzinárodná klasifikácia chorôb) [4].

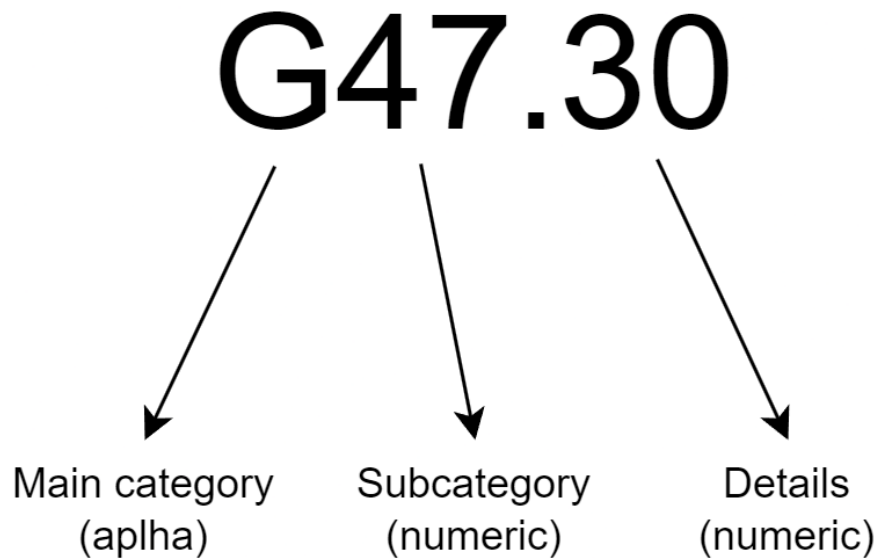


Figure 4.1: Structure of MKCH-10 code

This code consist of three parts as shown on image 4.1. First part is one letter that encodes main categories of diseases also known as chapter, for example codes starting with G are diseases of the nervous system. After that there is two numeric characters that further specify subcategory of disease such as codes from G40 to G47 which are episodic and paroxysmal disorders and specifically G47 are sleep disorders. We can see that episodic and paroxysmal disorders are only up to G47, meaning that theoretically there can exist subgroups G48 and G49 which have 4 in second position but does not belong to same G4 subcategory as G40 or G47 . Thankfully this is not the case and in cases like this when higher lower subcategory (like G4) does not have 10 lower level subcategories (like G47) these subcategories does not exist at all and in case it have more than 10 lower level subcategories it gets multiple consecutive high level subcategories, for example disorders of other endocrine glands spans from E20

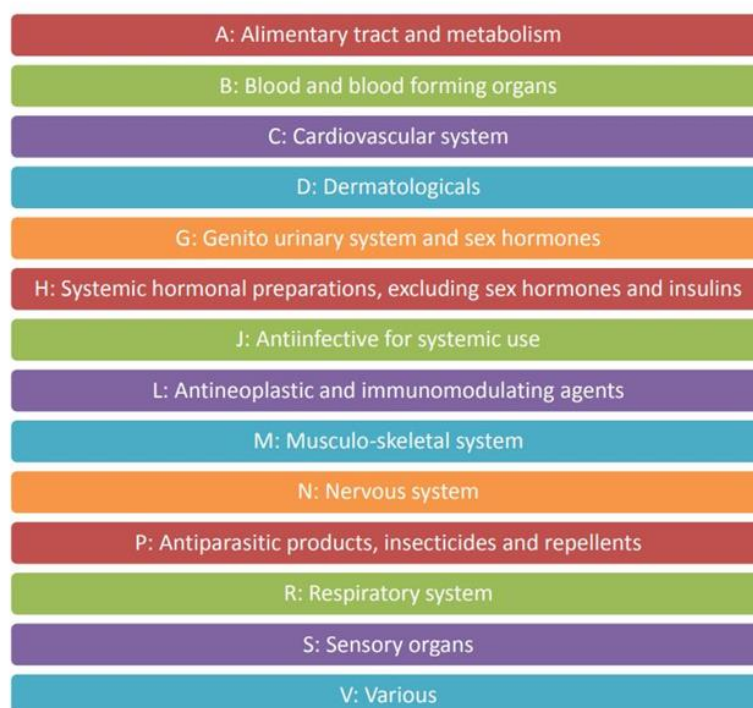
to E35. Then code contains dot after which there are characters that further describe details of disease such as etiology, anatomic site and severity. Official documentation of ICD-10-CM codes states that codes can be up to 7 characters long meaning that after first three characters specifying category there can be up to 4 alphanumeric to further specify the disease CITE, however Slovak version MKCH-10 codes contains at most two numeric characters to specify disease cite and these details are organized in a way where the first position conveys higher-level information than the second, for example G47.3 is sleep apnea and G47.30 is primary central sleep apnea.

To embed this code we firstly split it into its three parts and embed each part separately. After that we concatenated embedding of each part to get final embedding of disease. To embed main category we generated vector containing random numbers from uniform distribution for each letter of English alphabet (all main categories). We used random vectors in order to have relatively similar distance between any two main categories since there are no particular relationships between these categories, we were thinking also about using one-hot encoding which would make distance between each two categories perfectly same but we didn't do it since it would have restricted length of vector. In embedding second part which is subcategory we used different approach, where to each number between 00 and 99 (all possible values of this part) we assigned linearly number in chosen interval. This assigned number was then repeated multiple times to create vector. This approach has advantages and disadvantages. Advantage is that we can be sure that closely related disease subgroups like G46 and G47 would get close embedding since their subgroup codes are close on number line. However there are also two disadvantages, first is that G49 and G50 would be similarly close as G46 and G47, but thankfully in a most cases either X9 code doesn't exist at all, creating a gap, or if X9 code exist it belong to category that go past X on as higher level subgroup. Another disadvantage is that distance between two higher level subgroups can vary quite dramatically even though in reality there might not be reason for that difference in distance. For example, using this approach higher level subgroup G40-G47 is much closer to subgroup G50-G59 than to G80-G83. Finally to embed details we decided to use same approach as for subgroups since these codes work similarly, only difference was that not all codes had second level details, in such cases we add 5 as a proxy in order to minimize average distance from all potential codes with same first level details code that contain second level detail information while also maximizing average distance to different first level detail codes. Final after embedding each part final embedding is created as their concatenation, to encode importance of each part in final embedding we gave them different lengths, this works thanks to the fact that each value in each of vector have same mean and same variance. Importance of each level was encoded using different lengths of vector for each part where embedding of

main category was longest and details got shortest embedding.

4.1.3 Drug embedding

Similarly to diagnosis, to embed drug information we embed international code associated to these drug. In case of drugs it was Anatomical Therapeutic Chemical classification system also known under abbreviation ATC. In a same way as MKCH-10 code this code can be split into multiple parts where each next part contains finer information. It contains of 5 parts or levels. First level encodes main anatomical or pharmacological groups. There are fourteen such groups, encoded by single letter, which are shown in the figure 4.2. Then second level encodes pharmacological or therapeutic subgroup using two digit number, after that there two levels that further specify pharmacological, therapeutic or even chemical subgroup, these two levels are both encoded using single letter each. Final fifth encoded with two digit number contains information about specific chemical substance inside drug.



A: Alimentary tract and metabolism
B: Blood and blood forming organs
C: Cardiovascular system
D: Dermatologicals
G: Genito urinary system and sex hormones
H: Systemic hormonal preparations, excluding sex hormones and insulins
J: Antiinfective for systemic use
L: Antineoplastic and immunomodulating agents
M: Musculo-skeletal system
N: Nervous system
P: Antiparasitic products, insecticides and repellents
R: Respiratory system
S: Sensory organs
V: Various

Figure 4.2: Fourteen main anatomical or pharmacological groups and their corresponding first level ATC code [1]

Embedding was done in very similar way as in diagnosis embedding, meaning each level was embedded separately and final embedding was done as concatenations of them. In this case each level was embed using random vector from uniform distribution. We used random vectors for each level since none of levels contains any internal sub-groupings similar to subgroups of diagnosis (see 4.1.2 subgroup codes). To encode

importance of levels we again used lengths of random vectors. With this embedding we should get codes whose similarity is more dependent on whether lower more important levels match than higher ones.

4.1.4 Medical procedure embedding

Final part to embed was medical procedures. In this case there is no structured code that can be used and is implemented in Slovak healthcare systems. So what we have done is to embed description of the procedures. For that we used large language model (LLM) trained on multiple languages including Slovak. Dimensionality of resulting embedding was then reduced using PCA, to get rid of dimensions with small variance meaning they encode small amount of information.

For LLM we choose language-agnostic BERT sentence embedding also known under abbreviation LaBSE. It's a model developed by Google to encode text into high dimensional vectors. This model was trained 109 languages including Slovak. Main goal of this model is to generate similar representation to pairs of sentences which have same meaning and are only translations in two different language [3]. This approach should produce better results compared to standard text embedding models trained solely on Slovak language, since LaBSE model is during training comparing embedding not only to similar sentences in Slovak language but also their translation in other which could mitigate a relatively small amount of Slovak language data compared to other more commonly used languages. Additionally this model could know domain specific words in our case medical terms which are left in foreign language and would most likely not be found in Slovak only corpus. To confirm this expectation, we compare results to Word2Vec model trained on purely Slovak language using corpus containing 110 million words [2]. As a way of comparison we choose to firstly group embedding into categories using K-nearest neighbors algorithm (KNN), and visually asses whether description in resulting categories show any resemblance or whether they seem random.

Architecture of LaBSE model consist of 4 parts [5]:

1. Encoder-only transformer (BERT model)
2. Pooling layer
3. Dense layer
4. Normalization layer

Encoder-only transformer (BERT model)

First and most important part of LaBSE model is transformer, which is a deep learning architecture. More specifically LaBSE uses BERT so bidirectional encoder representations from transformers model which is encoder-only transformer architecture meaning this model does not contain decoder found in standard transformer which is usually used for prediction, because of this BERT model is focused in extracting contextual information from input text. Architecture of standard BERT model looks like this:

1. Tokenizer layer
2. Embedding layer
3. Encoder
4. Task layer

First layer is tokenizer, this layer takes input text split it into tokens which in case of BERT model is called PieceWise tokenizer which split text into subwords so something close to syllables. PieceWise tokenizer has advantage compared to different tokenizers that use either words or characters. Compared to character wise tokenizing, subwords contain more information than characters, and compared to word tokenizer is that there a lot less subwords than words and are more similar across multiple languages, creating much smaller vocabulary which is especially important for multilingual models. After split this layer assign integer number to each unique token, LaBSE model vocabulary distinguishes around 500 000 different tokens.

After that comes embedding layer which assign real number vector to each token, more specifically BERT model compute three different embeddings and add them together and normalize result to get final one. First is token type embedding, which is basic embedding where each token in vocabulary is assigned it's unique embedding. Second is positional embedding, as name suggest this embedding contain information about where in the sequence token is found giving additional information. Third and final embedding is segment type, which encodes information about to which segment, usually sentence token belong, important for embedding input text consisting of multiple sentences.

Third and most important layer is encoding. This is the layer in which contextual information are mined from the text. This consist of multiple attention block stacked one after the other. In case of BERT used in LaBSE there are 12 such blocks. Each

attention block consist of two parts, multi-headed self-attention layer and multi-layered feed-forward neural network.

Each head of self-attention layer takes a input set of embeddings and to compute new set of embeddings which should encode not only original information but also information about relationship between original ones, in other words it encodes effect of tokens onto each other. To do that it compute for each input embedding three vectors usually called key, query and value by multiplying input embedding with three matrices which values are learned during training. After that to get new embedding query vector is multiplied with matrix created from key vectors so resulting vector is vector of dot product of single query and all keys, this vector then goes through softmax function to normalize result. In some transformer models before softmax this vector get masked. Masking is done by setting values where key vector belongs to later embedding than query to minus infinity, this way after softmax this values become zeros. This is done so that later embedding does not affect previous ones. It's mostly useful in models trained to predict next token in order for model to learn to predict only based on tokens from past on not future. However in case of model like BERT where emphasis is on extracting as much information from input text masking is not done. Final step to get new embedding is to multiply vector we got after applying softmax with matrix composed of value vector to get linear combination of value vectors. This resulting vector is new embedding. This is done for all query vectors. List list resulting vectors is output embedding of attention head. Since this model use multi-headed attention more specifically in case of LaBSE 12-headed attention, this process is simultaneously independently done 12 times and results of each head are then concatenated into final result of attention. By using multiple head we expect that each head can extract different information from input and final concatenation give us result that contains all extracted information.

This concatenation of new embeddings goes than into multi-layered feed-forward neural network sometimes also called multilayer perceptron or MLP for short, architecture of this model is explained in subsection 4.4.1.

Result goes to next attention block and process is repeated again. Result of final attention block then goes to last part which is task layer, this layer can be viewed as simple decoder that map resulting embeddings back into token space. Based on this results is then model pre-trained. Training process of BERT model usually consist of two tasks on which are model trained at the same time. First is masked language modeling (MLM) where 15% of input tokens are masked meaning either replaced by mask placeholder or by random different token, after that masked input goes into

model, then from resulting tokens are taken those on position of masked tokens in input and are compared to correct token before masking which creates error that is back-propagated through model updating it's parameters [13]. Other task usually used is called next sentence prediction (NSP), in this task model gets input which start with special classify token and after that two spans of texts separated by special separator token. Task of model is to say whether these two spans of text can appear one after another or more precisely whether they appeared one after another in training corpus and put this information into first token of result encoded by two special tokens either "is next" or "not next" token and similarly difference between expected and resulting first token create error that back-propagates through model [21]. However in some BERT-based model like LaBSE second task is replaced with translation language modeling (TLM), this task is extension of MLM in which model gets two concatenated sentence instead of one and where the second sentence is translation of the first in another language, rest of the task is than same as in MLM, so whole input gets masked and model is tasked to predict masked tokens [12]. Task layer is used primarily in only during pre-training and is omitted when model is used for different task as many use-cases does not need tokens in results but use embeddings from encoding layer as form of text encoding which is that further used in task specific layers on which then model is fine-tuned.

Pooling layer

After BERT model returns embeddings of all input tokens, these then needs to be aggregated into single vector which corresponds to embedding of whole input. This aggregation is done using pooling layer. In case of LaBSE this pooling is done simply by taking embedding of first token which is an embedding of special classify token added to beginning of BERT input.

Dense layer

Next layer is standard feed forward dense layer with use hyperbolic tangent activation function. Number of input and output neurons is same, and additional bias neuron is used.

Normalization layer

Final layer is normalization which task is only to normalize resulting vector, so divide vector by number in order to have final vector with L_2 norm equal one.

After we embed each description of procedure using LaBSE model, we get 768 dimensional vectors assigned to them. As this dimensionality is relatively high, especially

when we consider that descriptions of procedures used limited primarily medical vocabulary, we decided to use dimension reduction technique in order to remove dimensions which are least informative and contribute mostly noise. We used principal component analysis (PCA).

Finally we create record embedding by concatenating all four parts. Since we have two separate datasets, one for prescribed drugs and one for medical procedures, we always have only three out of four information available for each record, since timestamp and diagnosis are always available, we substitute missing part with vector of zeros with appropriate length which is most neutral embedding since we centered both medical procedure and drug prescription embedding around zero.

4.2 Prediction of future number of records

Our task is to get information about how much will cost patient in the future, more specifically, how much it will cost in the next. Since our approach predicts future records, assign to them cost and sum the cost into resulting cost, we need to somehow be able to assess how many records need to be generated in order to simulate approximately next year. For this task we tried two different approaches.

First approach was to predict approximate number of records using linear regression which gets counts of records from previous years and predict next one. Other approach used stopping criterion, which uses timestamp information which is available in each input and generated records, and stop generating once difference between last timestamp from patient data and last generated timestamp surpass one year threshold.

Each of these method have it's own disadvantages. In case of linear regression, number of records per year can vary significantly, so even though we expect increase [7] regression might not be able to catch this trend, especially if patient data consist of only few years in the past. Potential issue with second approach is that it's dependent on how well model learned that timestamp should always increase.

4.3 Future record prediction

This task is most important while also hardest to train. Our goal is to create model which would be able to predict potential next record based on patient previous ones. This task in general is almost impossible with amount of information we have, since there are many other factors that determine whether, when and what new disease will

the patient become infected with in the future, whether his current state will improve or worsen, and most importantly what specific steps will doctor take.

Fortunately our ultimate goal is not to predict specific patient future but only estimate how much would he most likely cost. So we expect that even though we wouldn't predict what exactly happens we would be able to estimate total cost of those records.

We try multiple different model for this prediction. First three models we tried were recurrent neural networks (RNN). More specifically we tried multi-layer Elman RNN, multi-layered gated recurrent unit (GRU) RNN and multi-layer long short-term memory (LSTM) RNN. Last model we tried was Transformer.

4.3.1 Recurrent neural network

In general recurrent neural network is type of neural network that in some way uses results from previous step or steps in order to improve prediction. These models are used for ordered data where next step is dependent on more than single previous one.

4.3.2 Elman RNN

Elman RNN also known as simple recurrent network (SNR) is type of recurrent neural network that utilize results of hidden layer before activation in previous step as an additional input in next one. We can see this architecture on figure 4.3, where on left we can see how forward propagation look and on right how it looked unrolled over time, we see that middle layer which is a hidden layer of model gets two inputs, first is input vector x_t , where t denotes step (usually time step), which is multiplied with matrix of weights W_i and second one is vector h_{t-1} which is result of this layer also called context vector in previous step. which is multiplied by different matrix of weight denoted as W , these two results after they are then summed together and result goes to activation layer which create resulting new vector h_t [15], this whole process can be summarized by this equation (where b_i and b are bias vectors):

$$h_t = act(x_t W_i^T + b_i + h_{t-1} W^T + b) \quad (4.1)$$

Usually this result is directly used as output or go into single fully-connected feed-forward layer. In multi-layered version of this network, result of first hidden layer would go into next hidden layer together with result of that specific layer in previous step and again both would get multiplied by matrices of weight specific for that layer, summed and results goes through activation function.

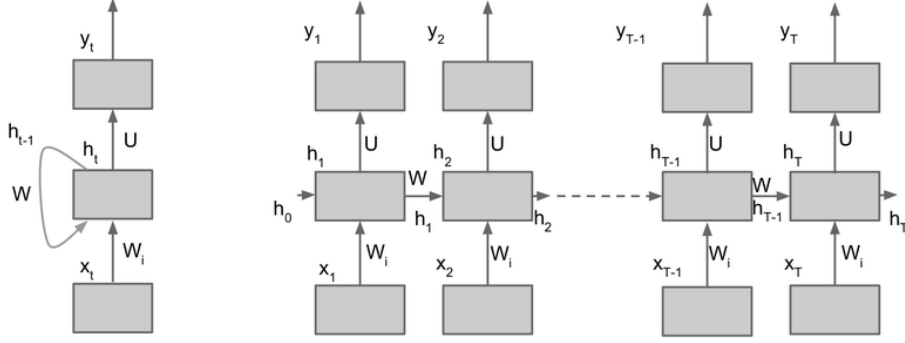


Figure 4.3: Architecture of Elman RNN [6]

4.3.3 Gated recurrent unit

Gated recurrent unit is more complex recurrent neural network compared to Elman RNN. This model was developed as simplification of even more complex LSTM model. We can say that it consist mainly of three parts that interact with each other and together create final prediction. Those parts are reset gate, update gate and candidate hidden state computation. We can see structure of hidden layer of GRU on figure 4.4. On this diagram sigmoid means fully-connected feed-forward layer with sigmoid activation function and tanh means fully-connected feed-forward layer with hyperbolic tangent activation function.

Reset gate is on left side of diagram, it's task is to using input and previous hidden state vectors modify previous hidden state vector which goes into candidate hidden state computation. This should helps capture short-term dependencies in time series by removing part of information from previous hidden state vector.

Candidate hidden vector computation is part that is similar to Elman RNN with two differences, firstly inputted hidden layer is modified by reset gate result and secondly resulting vector from this part is only candidate which goes into further calculation. This candidate contains mostly current and short-term past information thanks to specific vectors that input consist of.

Update gate, similarly to reset gate, uses concatenation of input and previous hidden state vector on input, but this time results are used to modify both previous hidden state and candidate hidden state in a way that resulting hidden state is weighted average of those two vectors where weights are results from this gate. This should help to

capture long-term dependencies in time series from previous hidden state vector and reintroducing it into final hidden state by combining it with candidate hidden state vector that should contain mostly current and short-term information.

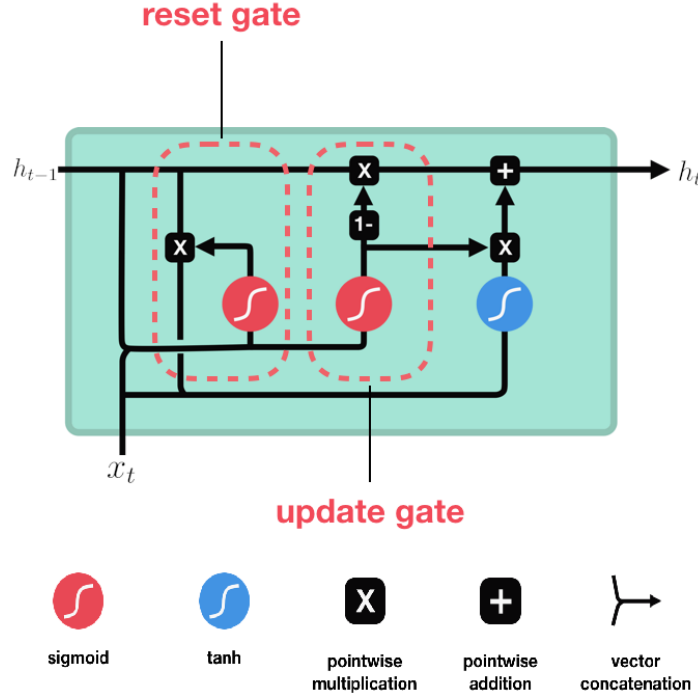


Figure 4.4: Architecture of GRU hidden layer

Multi-layered version is achieved by stacking hidden layers where hidden state vector of one layer is input vector of next one.

4.3.4 Long short-term memory

As mentioned earlier, long short-term memory is more complex predecessor of GRU. This model was developed with aim to mitigate vanishing gradient problem, that cause lack of long-term memory in models like Elman RNN, mainly by introducing memory vector, sometimes called memory cell, that should maintain information over longer period. As we can see in figure 4.5 this model consist of three gates, candidate memory computation. Legend in this diagram have same meaning as one in GRU architecture diagram.

All three gates and candidate memory computation has same input, which consist of input and previous hidden state vectors. In all of them this input goes into fully-connected feed-forward layer and then into activation function. In case of gates this activation function is sigmoid function, that bound all values into range (0,1), while

candidate memory calculation uses hyperbolic tangent as activation function.

Forget gate has the task of removing unnecessary information from memory vector by elementwise multiplication of it's result with previous memory vector.

Input gate together with candidate memory vector computation are meant to introduce new information into memory cell after adjustment from forget gate. Firstly model computes candidate memory then, then this vector is adjusted by input gate result and finally added to the modified previous memory vector. This results in new memory vector.

Finally output gate is used to determine how much each part of memory should contribute into new hidden state vector.

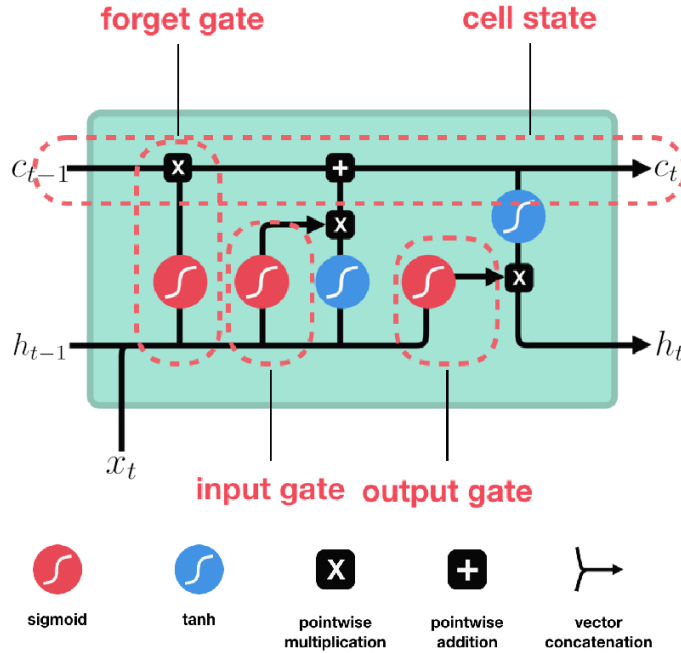


Figure 4.5: Architecture of LSTM hidden layer

4.3.5 Transformer

Last tested model was transformer, compared to transformer explained in subsection 4.1.4 here we tried more standard architecture that contains both encoder and decoder shown in figure 4.6. Encoder in this model is meant to contextualized representation of input while decoder takes this contextualized representation and mixes together with previous outputs to get prediction of next output. Other than these two main blocks this model usually also contains tokenizer and embedding layer to prepare in going data and fully-connected feed-forward layer with softmax activation function

to turn result of decoder into probability distribution of possible tokens.

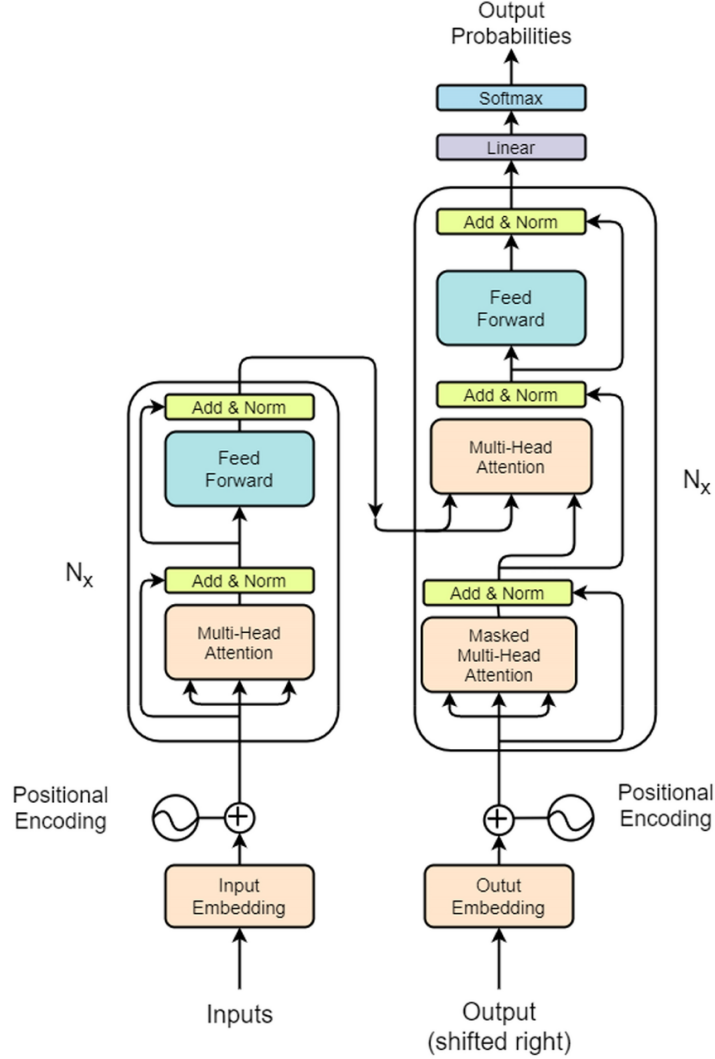


Figure 4.6: Architecture of one encoder-decoder block in transformer model from original "Attention is all you need" paper [22]

In our implementation we don't need tokenizer as our data are already in form of patient records which correspond to our tokens. In case of token embedding, we use embedding explained in section 4.1 and we add positional encoding to make sure model take into account also order of records.

Encoder architecture is very similar to one used in BERT and explained in 4.1.4, so it consist of multiple attention blocks where each block consist of multi-headed self-attention and multi-layered feed-forward neural network.

Architecture of decoder, similarly to encoder, consist of multiple attention blocks, however in this case attention block consist of three parts instead of two. First is

masked multi-headed self-attention which is similar to normal self-attention with only difference being application of masking to make sure that words on some specific positions does not affect words on other specific positions. Decoder use what's called casual or look-ahead masking which forbid future tokens to affect past ones, in our case this guarantees that record cannot be affected by record which happened after so in the future from it's perspective. Second part is multi-headed cross-attention, which takes two lists of token embeddings and compute effect of tokens in first list onto tokens in second list. In this case key and value vectors are computed from contextualized representation computed by encoder while query vectors are computed from results of self-attention in decoder. Final part of decoder attention block is multi-layered feed-forward neural network that further extract information from embeddings information from all attention heads.

After that results of decoder goes into final fully-connected feed-forward layer which change dimensionality of the decoder output from embedding size into vocabulary and apply softmax activation on result. This way we expect to get in last token probability distribution through all possible tokens. In our case we changed this last part. We run into problem where most of our records (our tokens) are unique creating extremely large vocabulary, this was caused partially by many possible combinations of disease, drug and medical procedure which are encoded in record and partially by timestamp which add additional uniqueness since it's improbable that two patients would get for example same drug, for same disease, at same age. Because of this we decided to remove softmax activation, changing fully-connected feed-forward layer in a way that result maintain dimension of embedding and added function that split result, find closest embedding of each part and concatenate result together leaving us with specific new embedding prediction instead of probability distribution.

4.4 Record cost prediction

Next step in total cost prediction is to predict how much each record would cost. For this we choose standard multilayer perceptron (MLP) or in another words multi-layered fully-connected feed-forward neural network.

4.4.1 Multilayer perceptron

Multilayer perceptron is a feed-forward neural network, so network where data flow single direction or in other words neurons don't form cycles. This network consist of fully-connected, sometimes called dense, layers with non-linear activation functions as

shown on figure 4.7, where we can see that this model can be split into three parts which are input layer which load the data, hidden layer which are trying to extract desired information using linear transformations and activation functions and finally output layer which contains final linear transformation followed by activation to output extracted information. Each layer can be described using this formula:

$$h_i = act(W_i h_{i-1} + b_i) \quad (4.2)$$

Where h_i is resulting vector of i-th layer, $act()$ is a non-linear activation function, W_i is weight matrix of i-th layer, h_{i-1} is resulting vector from previous layer and b_i is bias vector.

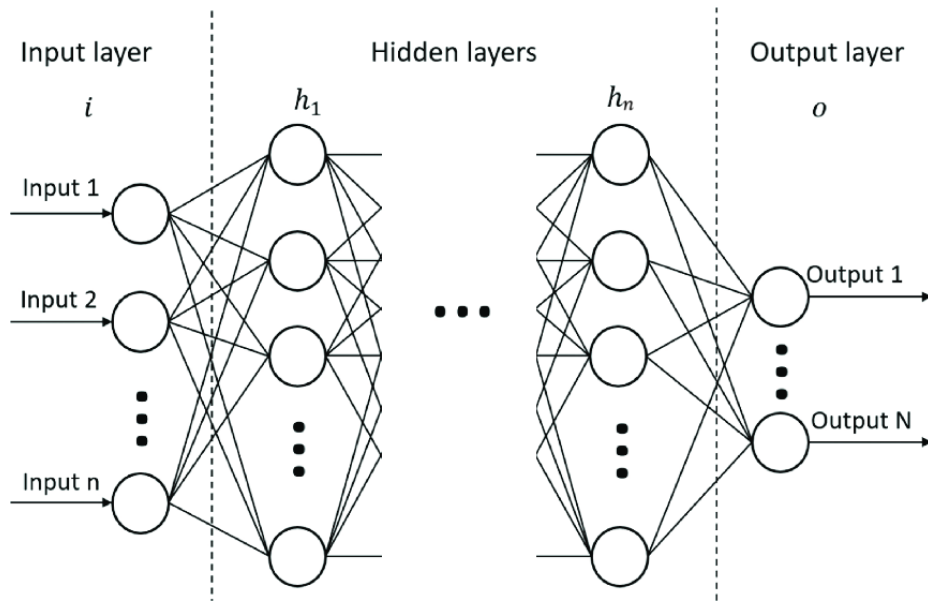


Figure 4.7: Architecture of multilayer perceptron [17]

Chapter 5

Software design

Whole code was written using Python language. We chose this language for its ease of use and extensive support for data processing and machine learning.

Whole code can be split into three parts:

1. Embedding - code to add embedded vectors into mapping files, has to be run once
2. Model training and validation - code to setup, train, validate and save prediction models, has to be run once
3. Predictor - code to load trained models and predict future cost of inputted patients

Chapter 6

Implementation

6.1 Embedding of patient

First of sub-tasks for prediction of patient future costs is to embed each patient record into numerical vector that would be understandable for neural network. For each record of a patient we embed four information. First and easy to implement is a timestamp which is computed using numerical and date values, then there three more tricky information and those are diagnosis, medical procedure and prescribed drug.

6.1.1 Timestamp

To compute timestamp we first took all records for single patient and found one with earliest date. This record is then used as pivot for computation of all timestamp for patient. To compute timestamp for this record we took age of patient multiplied by 365 to get approximate age of patient in days which served as a timestamp. For all subsequent records we compute difference between date of record and date of first record in days and add this difference to timestamp from first record to create one for this record.

6.1.2 Diagnosis embedding

As discussed in 4.1.2 embedding of diagnose is based on MKCH-10-SK (ICD-10-CM) code of disease. Where we split this code into three parts and embed each other independently and finally concatenate result.

To embed main category we generated vector containing random numbers using uniform distribution for each letter of English alphabet we choose to sample these random numbers from interval $[-0.5, 0.5]$. We choose this interval mostly arbitrarily since we were planing to pass recording embedding into normalization function once it complete.

For subcategory and details we linearly assigned value to each possible two digit code. We chose interval from which we take this value to be $[-0.5, 0.5]$, meaning subcategory 00 would get -0.5 category 50 would get 0 and category 99 would get 0.5, this interval was chosen in order to for each dimension of this embedding to have same mean and standard deviation as each dimension of main category. Thanks to that they also have on average same distance per dimension. This means that each position of each part of embedding should contribute to total distance with same weight.

Most important part was to assign lengths to vector of each part in a way that would encode their importance. Main category, the most important part, got vector of length 26, subcategory part got length 9 and finally details got length 3.

Showcase of resulting embedding can be seen on image 6.1 where each part is highlighted by different color and all values are rounded to two decimal.

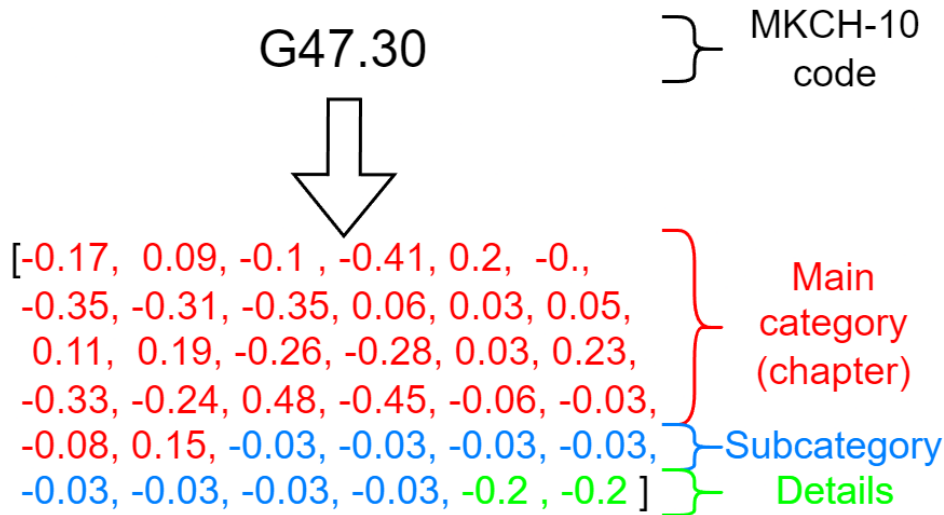


Figure 6.1: Showcase of resulting embedding of specific diagnosis (rounded to two decimal places)

6.1.3 Drug embedding

Embedding was done in very similar way as in diagnosis embedding, meaning each level was embedded separately and final embedding was done as concatenations of them. In this case each level was embed using random vector from uniform distribution with interval $[-0.5, 0.5]$. We used random vectors for each level since none of levels contains any internal sub-groupings similar to subgroups of diagnosis (see 4.1.2 subgroup codes). To encode importance of levels we again used lengths of random vectors, where with higher level vectors shortens. Lengths of vectors for each level can be seen in 6.1. So

total length of embedding is same as embedding for diagnosis. In case code is incomplete, meaning it's missing higher levels, missing part is substituted with zeros which we consider neutral elements.

Level	Length
1	21
2	9
3	5
4	2
5	1

Table 6.1: Lengths of random vectors assigned to each information level of ATC code

With this embedding we should get codes whose similarity is more dependent on whether lower more important levels match than higher ones.

6.1.4 Medical procedure embedding

Embedding of medical procedure was straightforward since we used already trained model. We inputted all medical procedure descriptions into LaBSE model resulting in 768 dimensional vector for each. Once all descriptions were embedded we took all embedding and performed principal component analysis (PCA) on them. Finally from results of PCA we took first 119 dimensions which account for over 90% of variance in original embedding, meaning we substantially decreased dimensionality of embedding while retaining most of the information.

For LLM we choose language-agnostic BERT sentence embedding also known under abbreviation LaBSE. It's a model developed by Google to encode text into high dimensional vectors. This model was trained 109 languages including Slovak. Main goal of this model is to generate similar representation to pairs of sentences which have same meaning and are only translations in two different language [3]. This approach should produce better results compared to standard text embedding models trained solely on Slovak language, since LaBSE model is during training comparing embedding not only to similar sentences in Slovak language but also their translation in other which could mitigate a relatively small amount of Slovak language data compared to other more commonly used languages. Additionally this model could know domain specific words in our case medical terms which are left in foreign language and would most likely not be found in Slovak only corpus. To confirm this expectation, we compare results to Word2Vec model trained on purely Slovak language using corpus containing 110 million words [2]. As a way of comparison we choose to firstly group embedding into categories using K-nearest neighbors algorithm (KNN), and visually asses whether

description in resulting categories show any resemblance or whether they seem random.

Finally we create record embedding by concatenating all four parts. Since we have two separate datasets, one for prescribed drugs and one for medical procedures, we always have only three out of four information available for each record, since timestamp and diagnosis are always available, we substitute missing part with vector of zeros with appropriate length which is most neutral embedding since we centered both medical procedure and drug prescription embedding around zero.

Chapter 7

Research

7.1 Embedding of patient

First of sub-tasks for prediction of patient future costs is to embed each patient record into numerical vector that would be understandable for neural network. For each record of a patient we embed four information. First and easy to implement is a timestamp which is computed using numerical and date values, then there three more tricky information and those are diagnosis, medical procedure and prescribed drug.

7.1.1 Diagnosis embedding

We need to confirm that closely related diagnosis would get embedding with smaller distance meaning higher similarity compared to less related diagnosis. In order to confirm that our embedding has desired properties we computed similarity of embedding of multiple codes. As similarity function we choose simple multiplicative inverse of Euclidean distance. In table 7.1 we can see results. Highest similarity was between codes G47.30 and G40.09 which is expected since they belong to same main category and very close subcategory, second highest was between H40.09 and H18.80 which are only other combination that belong to same main category, this confirms that main category has biggest impact since this similarity is significantly higher than that between G40.09 and H40.09 which differ only main category.

Code A	Code B	Similarity
G47.30	G40.09	2.77
G47.30	H40.09	0.53
G47.30	H18.80	0.46
G40.09	H40.09	0.54
G40.09	H18.80	0.45
H40.09	H18.80	0.84

Table 7.1: Similarities of embedding of multiple chosen MKCH-10 codes

Another confirmation that embeddings works as intended by clustering them. More specifically we would expect that if we group embeddings into as many clusters as are main categories, each cluster should contain mostly if not only diagnosis from one main category. On order to test this we used K-means clustering with 26 clusters which is number of different main categories of diagnosis and got these results:

Cluster ID	Cluster size	Frequency of first level values
0	654	C: 654,
1	2092	M: 2092,
2	766	Y: 766,
3	543	S: 543,
4	786	Z: 786,
5	1100	T: 1100,
6	1067	X: 1067,
7	620	H: 496, U: 124,
8	752	S: 752,
9	1770	M: 1770,
10	739	Q: 739,
11	584	D: 584,
12	507	F: 507,
13	958	W: 958,
14	556	E: 556,
15	491	A: 491,
16	553	O: 553,
17	627	K: 627,
18	872	V: 872,
19	521	G: 521,
20	463	L: 463,
21	420	R: 420,
22	465	B: 465,
23	717	J: 319, P: 398,
24	568	I: 568,
25	535	N: 535,

In table we can see that almost every cluster consist of diagnosis from with only single main category. Only immediately visible issues are clusters 7 and 23 where two main categories got assigned same cluster which is most likely caused by their small size compared to other and by the fact that largest categories M and S got split into two clusters.

7.1.2 Drug embedding

To confirm this we do similar check as we did with diagnosis code and compute similarity of four chosen ATC codes and see if our theory holds. To compute similarity we again use multiplicative inverse of Euclidean distance. We choose C01EB15, C01CA04,

C10AA07 and J01CA04, we would expect first two to be most similar since they match on first levels, than first two compared to third should have slightly lower similarity since they match on only first level and finally we expect that all three would be least similar to fourth one since first level is different, even though that second and forth match on all other levels. We can see results in table 7.2. We can see that results met our expectation with highest similarity between first two and lowest between any of first three and fourth, even in case of second and forth that match on all other levels except first.

Code A	Code B	Similarity
C01EB15	C01CA04	1.24
C01EB15	C10AA07	0.54
C01EB15	J01CA04	0.45
C01CA04	C10AA07	0.64
C01CA04	J01CA04	0.48
C10AA07	J01CA04	0.38

Table 7.2: Similarities of embedding of multiple chosen ATC codes

Also similarly to diagnosis we can test whether our embeddings would cluster mainly first level of code or not. Since there are 14 main anatomical or pharmacological groups 4.2 we used K-means algorithm with 14 clusters and got these results:

Based on results of these tests we can conclude that embeddings works generally as intended.

Cluster ID	Cluster size	Frequency of first level values
0	8645	C: 8645,
1	19132	N: 19132,
2	9322	L: 8657, S: 665,
3	11734	A: 11734,
4	7159	B: 7159,
5	9526	V: 9526,
6	4681	R: 4681,
7	14732	C: 14732,
8	7237	V: 7237,
9	4031	M: 3987, P: 44,
10	3599	G: 3599,
11	2367	N: 2367,
12	9032	D: 1266, G: 897, H: 1136, J: 5733,
13	6093	N: 6075, P: 18,

We can see that clusters mostly consist of embeddings with same first level values. There are some exceptions which are most likely caused by uneven number of drug in each category. Because of this categories with smallest number of drug like P, S, D,

H or G got assigned to cluster along with bigger category and categories with biggest number of drugs like N, C or V got split into multiple categories.

Based on results of these tests we can conclude that embeddings works generally as intended.

7.1.3 Medical procedure embedding

Embedding of medical procedures is two step operation, firstly embedding using LaBSE model and second dimension reduction using PCA.

Chapter 8

Results

Conclusion

REFERENCE SHOWCASE: 3

Bibliography

- [1] Anatomical Therapeutic Chemical (ATC) Classification — who.int. <https://www.who.int/tools/atc-ddd-toolkit/atc-classification>. [Accessed 25-09-2024].
- [2] GitHub - essential-data/word2vec-sk: Vector representations of Slovak words trained using word2vec — github.com. <https://github.com/essential-data/word2vec-sk>. [Accessed 19-10-2024].
- [3] Google | LaBSE | Kaggle — kaggle.com. <https://www.kaggle.com/models/google/labse/tensorFlow2/labse/1?tfhub-redirect=true>. [Accessed 18-10-2024].
- [4] Medzinárodná klasifikácia chorôb - MKCH-10 — nczisk.sk. <https://www.nczisk.sk/Standardy-v-zdravotnictve/Pages/Medzinarodna-klasifikacia-chorob-MKCH-10.aspx>. [Accessed 16-09-2024].
- [5] sentence-transformers/LaBSE · Hugging Face — huggingface.co. <https://huggingface.co/sentence-transformers/LaBSE>. [Accessed 19-10-2024].
- [6] Sivanand Achanta, Rambabu Banoth, Ayushi Pandey, Anandaswarup Vadapalli, and Suryakanth V Gangashetty. Contextual representation using recurrent neural network hidden state for statistical parametric speech synthesis. In *SSW*, pages 172–177, 2016.
- [7] Andreas Berzel, Gillian Z Heller, and Walter Zucchini. Estimating the number of visits to the doctor. *Australian & New Zealand Journal of Statistics*, 48(2):213–224, 2006.
- [8] Vicent Caballer-Tarazona, Natividad Guadalajara-Olmeda, and David Vivas-Consuelo. Predicting healthcare expenditure by multimorbidity groups. *Health Policy*, 123(4):427–434, 2019.
- [9] CDC. ICD-10-CM — cdc.gov. <https://www.cdc.gov/nchs/icd/icd-10-cm/index.html>. [Accessed 16-09-2024].

- [10] Yuriy Chechulin, Amir Nazerian, Saad Rais, and Kamil Malikov. Predicting patients with high risk of becoming high-cost healthcare users in ontario (canada). *Healthcare Policy*, 9(3):68, 2014.
- [11] Edward Choi, Cao Xiao, Walter Stewart, and Jimeng Sun. Mime: Multilevel medical embedding of electronic health records for predictive healthcare. *Advances in neural information processing systems*, 31, 2018.
- [12] Alexis Conneau and Guillaume Lample. Cross-lingual language model pretraining. *Advances in neural information processing systems*, 32, 2019.
- [13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.
- [14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.
- [15] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.
- [16] Fangxiaoyu Feng, Yinfei Yang, Daniel Cer, Naveen Arivazhagan, and Wei Wang. Language-agnostic bert sentence embedding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 878–891, 2022.
- [17] Miao Jin, Qinzhuo Liao, Shirish Patil, Abdulazeez Abdulraheem, Dhafer Al-Shehri, and Guenther Glatz. Hyperparameter tuning of artificial neural networks for well production estimation considering the uncertainty in initialized parameters. *ACS omega*, 7(28):24145–24156, 2022.
- [18] Elizabeth C Lorenzi, Stephanie L Brown, Zhifei Sun, and Katherine Heller. Predictive hierarchical clustering: Learning clusters of cpt codes for improving surgical outcomes. In *Machine Learning for Healthcare Conference*, pages 231–242. PMLR, 2017.

- [19] Riccardo Miotto, Li Li, and Joel T Dudley. Deep learning to predict patient future diseases from the electronic health records. In *Advances in Information Retrieval: 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20–23, 2016. Proceedings 38*, pages 768–774. Springer, 2016.
- [20] Mohammad Amin Morid, Olivia R Liu Sheng, Kensaku Kawamoto, Travis Ault, Josette Dorius, and Samir Abdelrahman. Healthcare cost prediction: Leveraging fine-grain temporal patterns. *Journal of biomedical informatics*, 91:103113, 2019.
- [21] Yi Sun, Yu Zheng, Chao Hao, and Hangping Qiu. Nsp-bert: A prompt-based few-shot learner through an original pre-training task–next sentence prediction. *arXiv preprint arXiv:2109.03564*, 2021.
- [22] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.