Berliner Hochschule für Technik

Department VI - Medieninformatik

Bachelor-Thesis

Transformer Based Medication Prediction from Clinical Notes

Philipp Haustein
Matriculation-Number: *879320*

1. Reviewer: Prof. Dr.-Ing. habil. Alexander Löser
2. Reviewer: Prof. Dr. Agathe Merceron

1. Supervisor: Paul Grundmann
2. Supervisor: Betty van Aken
3. Supervisor: Michalis Papaioannou

Berlin 07.10.2021

Abstract Deciding on a patient treatment strategy is complicated, as medical doctors only have limited time to make decisions, and the amount of known diseases, drugs, and treatments is vast and rapidly growing every day. The thesis aims to find an effective model for predicting prescribed drugs from clinical notes to support medical doctors with patient treatment strategy development. Therefore, we compare the Transformer based BERT Classifier by Devlin et al. ((2019)) and Bi-Encoder by Humeau et al. ((2020)). For the Bi-Encoder architecture, we added a general description of the prescribed drug from Wikipedia. Additionally, we propose a medication prediction task based on MIMIC-III by Johnson et al. ((2016)), the admission note dataset by van Aken et al. ((2021)) and Wikipedia and evaluate all proposed models on this task. Furthermore, we proposed a Wikipedia medication dataset that we linked to MIMIC-III. We found that including rarely used drugs in the training of our models does not hurt performance as much as expected. Additionally, we showed that adding additional information about the prescribed drugs from Wikipedia does not seem to increase prediction performance.

Zusammenfassung Sich für eine Behandlungsstrategie zu entscheiden ist kompliziert, da Mediziner meist nur begrenzt Zeit haben, um Entscheidungen zu treffen und die Menge an bekannten Krankheiten, Medikamenten und Behandlungen groß ist und täglich wächst. Das Ziel dieser Bachelorarbeit ist es, ein Model zu finden, welches in der Lage ist verschriebene Medikamente mithilfe der Aufnahmedokumenten in einer Klinik, vorherzusagen. Dafür haben wir das Transformer basierte BERT Model von Devlin et al. ((2019)) und das Bi-Encoder Model von Humeau et al. ((2020)) miteinander verglichen. Bei dem Bi-Encoder Model haben wir zusätzlich eine allgemeine Beschreibung des Medikaments von Wikipedia hinzugefügt. Außerdem schlagen wir einen Datensatz vor, anhand dessen sich die Qualität der Vorhersagen messen lässt. Dieser Datensatz würde mithilfe der MIMIC-III Datenbank von Johnson et al. ((2016)), dem Admission Note Datensatz von van Aken et al. ((2021)) und Wikipedia erstellt. Des Weiteren schlagen wir einen Wikipedia Medikamentendatensatz vor, welchen wir mit MIMIC-III verknüpft haben. Wir haben herausgefunden, dass das Einbeziehen von selten verwendeten Medikamenten beim Training unsere vorgeschlagenen Modelle die Vorhersagequalität nicht so stark verschlechtert wie erwartet. Zudem haben wir auch herausgefunden, dass das Hinzufügen von zusätzlicher Information über die verschriebene Medikamente von Wikipedia die Vorhersagequalität nicht zu erhöhen scheint.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Developing effective patient treatment strategies is a complex process, and medical doctors only have limited time to make decisions. Also, the amount of known diseases, drugs, and treatments is vast and rapidly growing every day. We want to propose multiple models which can support doctors with clinical decision-making by suggesting drugs they may not have considered. Like medication prediction, clinical outcome prediction is one part of clinical decision-making. Additionally, medication prediction and outcome prediction have similar inputs. Therefore, clinical outcome prediction is closely related to medication prediction. We were motivated to work on medication prediction because advances in outcome prediction have shown significant progress recently. Especially the usage of clinical notes is of particular interest for us. Natural Language Processing also showed substantial improvements over the years for different tasks. We are especially interested in transformer-based models as they have outperformed the previous state-of-the-art models based on Recurrent neural networks. Additionally, we are interested in adding information on prescribed drugs as textual drug descriptions and how this could refine prediction performance.

1.1 Problem Statement

Medical personal has to consider many different factors when deciding which patient treatment strategy to pursue. Additionally, they only have limited time to make decisions, and the amount of known diseases, drugs, and treatments is vast and rapidly growing. Automated systems can quickly search a massive amount of data and suggest drugs that medical doctors might not have considered. Therefore we want to present our approach to medication prediction.

For medication prediction, we try to predict the drugs prescribed during an admission from the admission notes. This model should provide tips for possible medication medical doctors might not have considered. If combined with outcome prediction, it can create a system that could support medical doctors with developing patient treatment and clinical resource planning.

But this is hard because medical professionals frequently use a lot of specialized vocabulary. Additionally, some drugs are infrequently used because they have a very specialized use. Therefore, examples of the usage of these drugs are limited.

1.2 Hypothesis

Devlin et al. ((2019)) introduced new transformer-based models like BERT, which they trained on a general text corpus, and which we can fine-tune to a variety of tasks. Additionally, a variant of BERT was proposed with the name BioBERT by Lee et al. ((2019)) which they trained on biomedical domain corpora. They have shown that they perform well on a variety of tasks. Therefore, we expect them to perform well on this task, that adding information on prescribed drugs should boost prediction performance and that it should be harder to predict rarely used drugs.

1.3 Methodology

1.3.1 Neural Network Architectures

We evaluated two neural network architectures. First, we chose a classification approach as our baseline, which is based on BERT by Devlin et al. ((2019)). Additionally, we decided pointwise learning to rank approach based on the Bi-Encoder by Humeau et al. ((2020)) with BERT-based encoders. This architecture allows us to add additional information about the drugs.

1.3.2 Evaluation Tasks

We propose a medication prediction task, which takes a textual patient representation as admission notes and predicts the drugs prescribed during the admission. We evaluated all proposed architectures on this task.

1.3.3 Training- and Evaluation Datasets

We created new datasets based on the MIMIC-III database by Johnson et al. ((2016)), the admission note dataset by van Aken et al. ((2021)) and Wikipedia for training and evaluation. Also, we created a Wikipedia medication dataset and linked it to MIMIC-III.

1.4 Outline

In chapter 2, we go through the foundations of the different neural network architectures used in this thesis. Furthermore, we present previous work on outcome prediction and medication prediction. In chapter 3, we define the medication prediction task we want to solve, present the data we use and how we preprocess it, and present the different approaches we employ to solve the defined task. In chapter 4, we go through our data processing pipeline and explain our experimental setup. Moreover, we show our hyperparameter tuning setup and which and how we tune our hyperparameters. In chapter 5, we present our hypothesis, evaluation metrics, results, and findings. Finally, in chapter 6, we summarize our results and findings and offer perspectives for future work.

1.5 Summary

In this chapter, we highlighted how medication prediction could help medical doctors with patient treatment strategy development. Additionally, we showed the problem currently associated with it: a specialized vocabulary in the medical domain and limited data of rarely used drugs and how we try to solve them. Moreover, we presented our methodology of how to develop an effective medication prediction model. Therefore, we introduced the different neural network architectures we employ, the evaluation task we use, and how we compose our training- and evaluation datasets. Finally, we gave an overview of the structure of the thesis.

Chapter 2

Background

2.1 Introduction

In this chapter, we will explain the underlying concepts of this work. These Concepts include the BERT model and the foundational concepts like the Attention mechanism and the transformer architecture. We will then continue with the Bi-Encoder and the background of the MISH activation function. After that, we present the strongly related outcome prediction task and discuss the previous progress in the medication prediction task.

2.2 Attention

Graves et al. ((2014)) first introduces the Attention mechanism. A basic Attention cell has three inputs(query, key, and value) and one output. All the inputs and outputs are vectors. The idea behind the Attention mechanism is as the names of the inputs imply a key-value store. There are two parts to the attention mechanism the compatibility function and a weighted sum. The compatibility function extracts how well each query vector matches each key vector. This matching is expressed as weights by which a weighted sum extracts the final output from the value vector. We could also think of the compatibility function as a measure of the amount of attention the model wants to pay to a specific value.

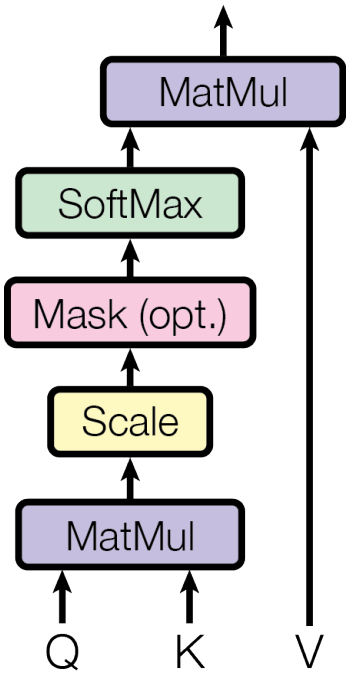


Figure 2.1: Scaled Dot Product Attention, Source: Vaswani et al. ((2017))

2.2.1 Scaled-Dot-Product-Attention

Vaswani et al. ((2017)) first introduces the scaled dot product attention. The scaled dot product attention is an implementation of the attention mechanism. Figure 2.1 shows how the cell works. They formulate the mechanism the following:

$$Attention(Q, K, V) = softmax(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}})V \tag{2.1}$$

Q, K, and V, respectively, refer to the query key and value input. Multiple keys, values, and queries can be processed at once as a matrix in parallel. d_k refers to the length of the key vectors. They divide QK^{T} by $\sqrt{d_k}$ to increase the numeric stability of the Softmax function. $softmax(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}})$ is the similarity function of the attention implementation while the the dot-product between $softmax(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}})$ and V represents the weighted sum.

2.2.2 Self-Attention

Vaswani et al. ((2017)) first introduces the Self Attention mechanism. Self Attention takes a basic attention cell and inputs the same vector as query, key, and value. This trick allows the parallel evaluation of sequences without using Reccurrent neural networks (RNNs), which are only able to process sequences sequentially. Then, the compatibility function outputs a matrix representing the relationships of the vectors in the sequence to each other.

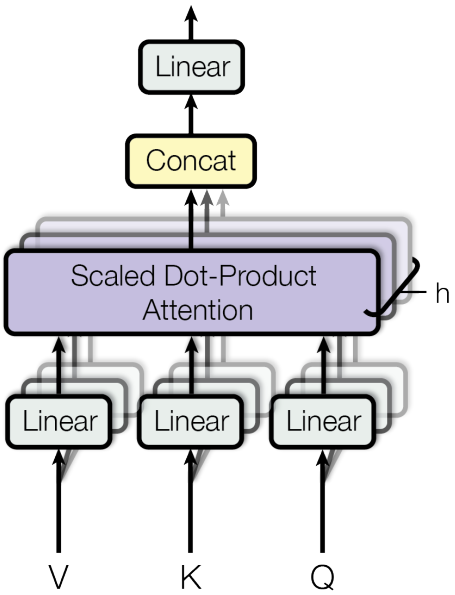


Figure 2.2: Multi Head Attention, Source: Vaswani et al. ((2017))

2.2.3 Multi-Head-Attention

Vaswani et al. ((2017)) first introduces the multi-head attention mechanism. Like the scaled dot product attention, it is an implementation of the attention concept. This approach uses multiple attention cells, which are called heads. As shown in Figure 2.2, the query, key, and value vector are passed through a linear layer, then split into h parts, where h represents the number of attention heads, and each attention head gets assigned a slice of the input. Then, each attention head processes its corresponding slice of the input vectors. Finally, the attention heads outputs are concatenated and passed through another linear layer.

2.3 Transformer

The Transformer is a sequence-to-sequence model from Vaswani et al. ((2017)). This model uses self-attention instead of RNNs layers. This approach avoids the sequential nature of RNNs and is, therefore, better suited for training on today's parallel Compute Accelerators. Transformers-based model often outperform RNN based approaches. Vaswani et al. ((2017)) use scaled dot product multi-head self-attention as attention implementation.

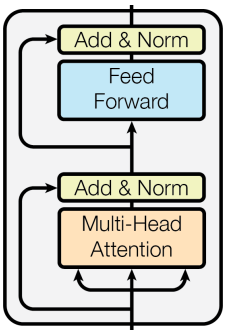


Figure 2.3: Transformer Encoder, Source: Vaswani et al. ((2017))

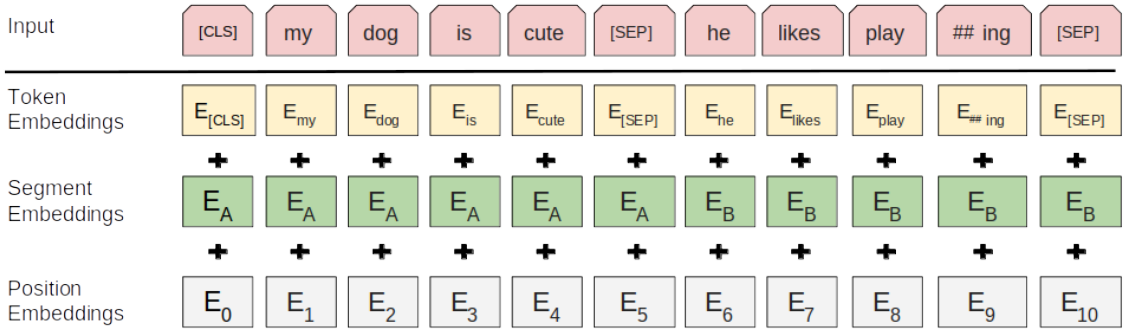


Figure 2.4: Bert Embedding, Source: Devlin et al. ((2019))

2.4 BERT

BERT, first introduced by Devlin et al. ((2019)), stands for Bidirectional Encoder Representation from Transformers. This model provides a language representation, which we can adapt for different tasks. At the core of the model, they stack multiple transformer encoder layers on top of each other.

Figure Figure 2.3 shows how the transformer encoder works. The encoder is composed of 2 layers, the scaled dot product multi-head self-attention mentioned in the Transformer section and a fully connected feed-forward layer. Both layers have a residual connection and layer normalization.

Before they pass the input sentence through the encoders, it is embedded. Figure 2.4 show how they do it. The final embedding comprises three embeddings, a token embedding, a segment embedding, and a position embedding. The token embedding encodes the token, the segment embedding encodes the sentence the token is in, and the position embedding encodes the token's position in the input sequence. The result is the sum of all three vectors.

They separate the training process into two parts the pretraining and fine-tuning. First, the model's pretraining is done on general text to generate a broad understanding of language. BERT uses the masked-language model and next-sentence-prediction tasks for pretraining, which we describe below. The advantage of both training methods is that we can generate the training data from a monolingual corpus. Second, for fine-tuning, the pre-trained model is adapted to a specific task. They showed that fine-tuning a pre-trained model only takes a few hours on a GPU.

They provide Pretrained models in 2 configurations, BERT base with 12 stacked encoder layers, embedding size of 768 and 12 attention heads per encoder layer, and BERT large with 24 encoder layers, embedding size of 1024 and 16 attention heads.

2.4.1 Masked Language Model

The mask language model task takes a sentence and masks some percentage of words at random. Then, the model tries to predict the masked words. In particular, they mask 15% of the words for BERT. For that, they replace 80% of the words with a unique mask token, replace 10% with a random token, and not replace 10%. They do not replace all the words with the mask token because most fine-tuning tasks will probably not use the mask token.

2.4.2 Next sentence prediction

In This task, they show the model two sentences, and the model tries to predict if the second sentence follows the first. For BERT, 50% of the pairs, the second sentence follows the first, while for the other 50%, the second is a random sentence from the corpus.

2.4.3 BioBERT

We will use a BERT model, which is pre-trained on biomedical domain corpora. This model is called BioBERT, was introduced by Lee et al. ((2019)) and uses the same hyperparameters as Bert base. They showed that they could deliver state-of-the-art performance in biomedical tasks.

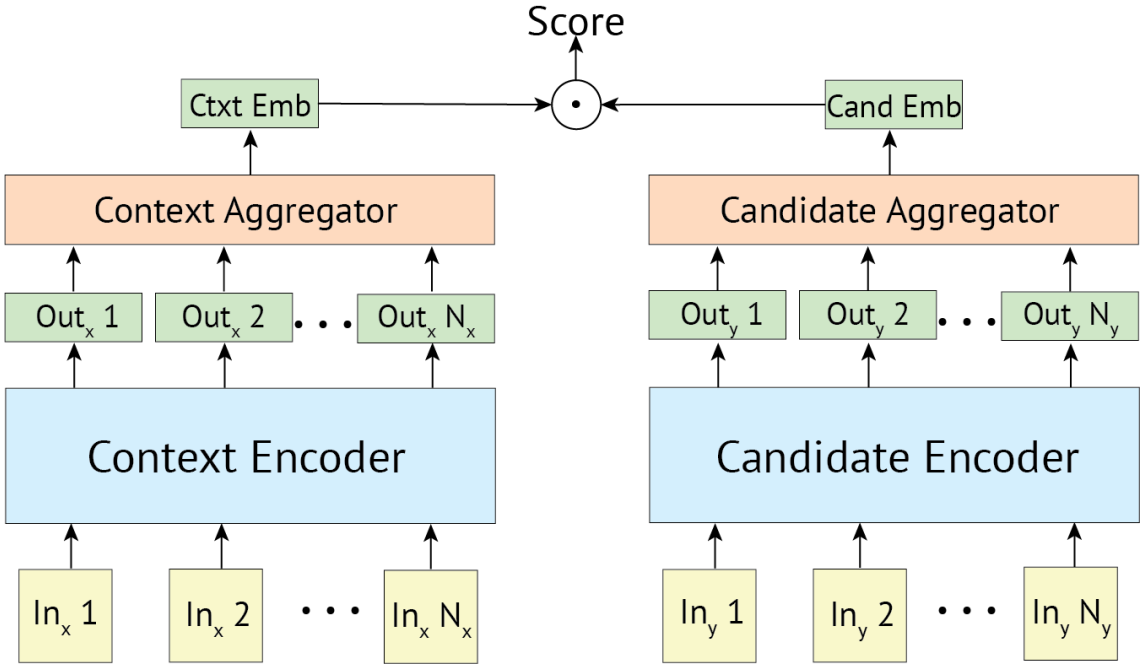


Figure 2.5: Bi-Encoder, Source: Humeau et al. ((2020))

2.5 Bi-Encoder

The Bi-Encoder architecture by Humeau et al. ((2020)) uses two encoders, a context encoder, and a candidate encoder. The idea is that the training transforms the embeddings of both encoders such that a context vector is near its relevant candidates in latent space. Figure 2.5 shows how the Bi-Encoder works. There is a context and a candidate encoder, which process the contexts and the candidates, respectively. If the encoder outputs multiple vectors, an aggregator should reduce the output of each encoder to one vector. Then they pass both vectors through a similarity measure to obtain the final score. The similarity measure at the top should extract the relevance of a given candidate for a given context. For example, we could use cosine or dot product similarity as the similarity measure.

2.6 MISH

MISH, first introduced by Misra ((2019)), is a self-regularized non-monotonic activation function. Furthermore, the SWISH activation function inspires them in the creation of this activation function. SWISH was first proposed by Ramachandran et al. ((2017)) it was found by Neural Architecture search on the CIFRA-10(Krizhevsky ((2009))) classification task using ResNet-20(He et al. ((2016))). They found that MISH outperforms SWISH on various tasks. Furthermore, Eger et al. ((2018)) showed that SWISH also performs well in NLP tasks. Therefore, we expect MISH to perform better on our task than other more widely used activation functions. We use MISH as an activation function for the Encoders of the Bi-Encoder, which we explain in the following chapters in greater detail. Misra ((2019)) define MISH as the following Function:

$$f(x) = x \cdot \tanh(\text{softplus}(x)) = x \cdot \tanh(\ln(1 + e^x)) \tag{2.2}$$

2.7 Clinical Outcome Prediction

Clinical Outcome Prediction describes a range of tasks where someone takes a representation in some form of a patient and predicts specific properties of clinical care for this patient. Most approaches filter out direct information about the predicted properties. Additionally, Medication prediction is strongly related to clinical outcome prediction as both use a patient representation as input.

This thesis is based on the paper by van Aken et al. ((2021)). The main goal of this paper is to help medical professionals consider possible risks and support hospitals with planning capacities. They propose a textual patient representation which is created by admission notes from the MIMIC-III database by Johnson et al. ((2016)). We explain MIMIC-III further in 3.4.1 and the generation of the admission note dataset in 3.4.2. These admission notes only contain information that medical professionals know at the time of admission. Therefore, we are not able to extract specific properties of clinical care from the patient representation.

We will go over common clinical outcome prediction tasks in the following sections. These tasks were manly taken from van Aken et al. ((2021)) and Zhang et al. ((2020)).

2.7.1 Diagnosis and Procedure prediction

The goal of diagnosis and procedure prediction is to predict possible diagnoses and procedures respectively for a patient. Both diagnosis and procedure are most times encoded as International Statistical Classification of Diseases and Related Health Problems (ICD) codes. Because of the hierarchical nature of ICD codes, most approaches group the codes to reduce the already huge amount of classes. We could view The diagnosis and procedure prediction tasks as separate tasks, but most approaches solve both tasks simultaneously because of the similarity of the output. Solutions to this task are proposed by Mullenbach et al. ((2018)), Hu and Teng ((2021)), van Aken et al. ((2021)) and Choi et al. ((2016)).

2.7.2 In-hospital mortality prediction

The goal of in-hospital mortality prediction is to predict the probability of a patient's death during admission. Most approaches treat this task as a binary classification task and filter out terms regarding mortality from the patient representation. Solutions to this task are proposed by Deznabi et al. ((2021)), Hashir and Sawhney ((2020)), van Aken et al. ((2021)) and Zhang et al. ((2020))

2.7.3 Length-of-stay prediction

The goal of in length-of-stay prediction is to predict the duration of a patient's admission. Depending on the approach, they represent the length of stay in another way. For example, van Aken et al. ((2021)) group length-of-stay into four categories: Under three days, 3-7 days, 1-2 weeks, more than two weeks. These categories were recommended to them by medical doctors. On the other hand, the approach of Zhang et al. ((2020)) treats the task as a binary classification problem with one class representing length-of-stay of up to seven days and the other of over seven days.

2.7.4 Hospital readmission prediction

The goal of hospital readmission prediction is to predict the probability of readmission. Most approaches classify admission as readmission if it happens no more than 30 days after a previous discharge. Additionally, most approaches treat this task as a binary classification task. Solutions to this task are proposed by Huang et al. ((2019)), Golmaei and Luo ((2021)) and Zhang et al. ((2020)).

2.8 Medication prediction

For medication prediction, we try to predict the medication during an admission from a textual patient representation as an admission note. As mentioned in 2.7 medication prediction is strongly related to clinical outcome prediction as both use a patient representation as input. Therefore, we will use the admission note dataset from van Aken et al. ((2021)). Although clinical outcome prediction has a solid foundation in literature, medication prediction in the form we practice it in this thesis has not. Therefore, we will present literature that is related but solve another task in the following section.

One approach is to predict effective drugs for cancer treatment using neural networks. They take properties of the cancer cell lines like metabolism or DNA to predict effective drugs. Solutions to this task are proposed by Baptista et al. ((2020)) and Kim et al. ((2020)). Although this approach presents a way of finding a solution to effective cancer treatment using neural networks, it is limited to cancer.

Another approach is to predict the effects and side effects of drugs using neural networks. The representation of the drug is dependent on the approach, for example, textual description or chemical structure of the drug. Solutions to this task are proposed by Jang et al. ((2018)) and Wu et al. ((2019)). Although this approach presents a way of effectively predicting drug effects, we have to extract patient information to search a database annotated with this data effectively.

A further approach is to predict prescriptions from previous clinical events using neural networks. They usually represent an admission of a patient as a sequence of events. These clinical events are most times encodes as ICD codes. Solutions to this task are proposed by Choi et al. ((2016)), Liu et al. ((2020)) and Song et al. ((2021)). The problem is that this approach disregards patient-specific information like gender or age and is only applicable if a Electronic health record (EHR) in the appropriate format exists.

2.9 Summary

This chapter first presented the background and related work about the models used in this thesis. Therefore, we show that key-value stores inspire the attention mechanism, a concept in neural networks. Furthermore, We presented the inner workings of the scaled dot attention, the multi-head attention, and self-attention, which are the foundational concepts of the Transformer architecture. Additionally, we showed how BERT by Devlin et al. ((2019)) leverages the underlying concepts of Transformer architecture to create the state-of-the-art model for solving multiple NLP tasks effectively and how Lee et al. ((2019)) created the BioBERT model variant. Moreover, we presented the Bi-Encoder concept by Humeau et al. ((2020)) and the MISH function and why we use it.

After that, we went through the previous related work of the strongly related outcome prediction. Then, we summarized the tasks that are usually associated with outcome prediction: diagnosis, procedure, in-hospital mortality, length-of-stay, and hospital readmission prediction, and reference works that proposed solutions to this task. Likewise, we briefly defined medication prediction and showed the relatedness of our task to clinical outcome prediction. In addition, we showed related approaches which solve related tasks, as previous work does not propose a solution to the task we defined.

Chapter 3

Methodology

3.1 Introduction

In this chapter, we focus on the methods we applied and developed for this thesis. First, we will define the medication prediction task. After that, we present the learning-to-rank concept. Afterward, we present the different data sources we used and how they are composed. Then we go over the different approaches we take to solve the problem. Finally, we present the concept behind the different loss functions we use.

3.2 Task Definition

In the medication prediction task, we try to predict the medication from a representation of a patient. This representation may involve natural language, images, or both. The images may contain the patient or parts of the patient. The text may describe the patient's symptoms, medical history, previous medication, allergies, family or social history. The medication's prediction may contain information on the medication's active ingredients, form, dose, and route. The active ingredient is the ingredient that is responsible for the primary health effect of a drug. This ingredient could also cause secondary effects. The form describes the delivery form of a drug like a tablet or injection. Also, The dose details the amount of the drug's active ingredient per intake. Further, The route depicts the delivery method of a drug like oral or infusion.

We wanted to use a task setup that is easy to create data for and is compatible with previous approaches. Therefore, we used clinical notes from doctor letters from hospital admissions. Furthermore, we focused on the active ingredient for the prediction of the medication. While other information like the dose could also affect the effect of a drug, we omitted these because it would make the prediction more complex in most cases.

3.3 Learning to rank

Learning to rank refers to a technic where neural network models rank search results, depending on the relevance. This goal could be archived independent or dependent on a user's query. This query could also contain information on the user's preferences. For example, a Programmer might search for the command line tool "cat" and not for the animal. Common approaches are pointwise, pairwise, and listwise learning to rank.

3.3.1 Pointwise learning to rank

Pointwise learning to rank calculates a score for an input document and optionally query. For example, someone could archive this goal by optimizing a function $f(d, q)$ where d refers to the document to rank, and q refers to the query. The user obtains the final result by sorting the list of documents by the calculated score.

3.3.2 Pairwise learning to rank

Pairwise learning to rank takes two documents plus optionally a query and returns which document ranks higher. This approach turns the problem into a binary classification problem. This could be archived by optimizing a function $f(d_a, d_b, q)$ where d_a and d_b refers to the documents and q refers to the query. This approach can yield contradictory results like A is better than B, B is better than C, and C is better than A. The user obtains the final result by using this function as a comparator function in a sorting algorithm.

3.3.3 Listwise learning to rank

Listwise learning to rank takes the list of documents and optionally a query to and returns the final order of the documents. This could be archived by optimizing a function $f_n(D, q)$ where $D = \{d_1, d_2, ..., d_n\}$ refers to the list of documents to rank and q to the query. The user obtains the final result by transforming the documents list in the model suggested order.

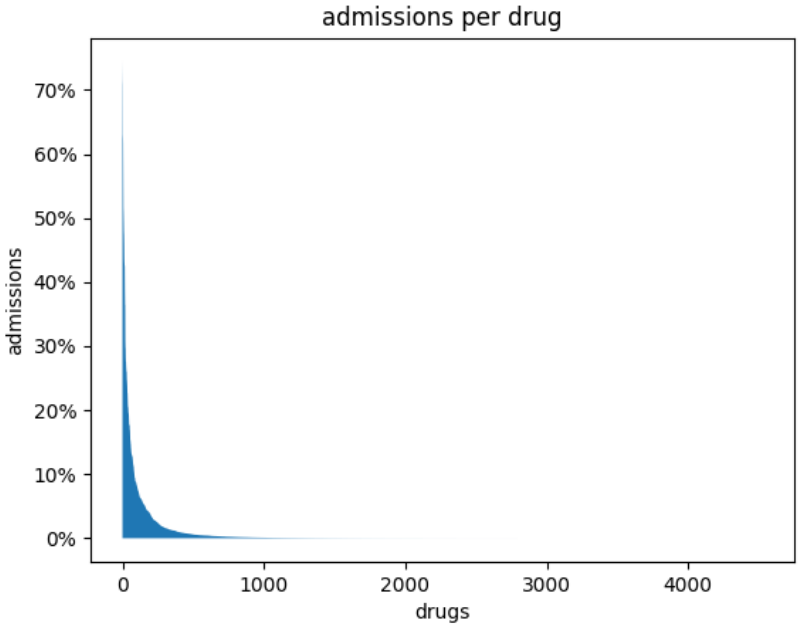


Figure 3.1: Admissions per durg

3.4 Data

3.4.1 MIMIC-III

Johnson et al. ((2016)) proposed the MIMIC-III v1.4 dataset. They say, *"MIMIC-III is a large, freely-available database comprising deidentified health-related data associated with over forty thousand patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012"* (Johnson et al. ((2016))).

We used the prescription table to obtain the prescribed medication. The prescription table contains information like the drug, form, dose, amount, and route. There is also a representation of the drug in various coding systems like Generic Sequence Number (GSN) or National Drug Code (NDC). Both are identifiers represented by a number that uniquely identifies a combination of ingredient, strength, form, and route. The NDC is also composed of a labeler code (4-6 digits), a product code (3-4 digits), and a package code (1-2 digits). There is a notation where dashes separate the different parts, but this separation does not exist in MIMIC. Therefore, splitting the different parts is not trivial for MIMIC. There are 4156450 prescriptions in total in MIMIC-III.

We use the drug column from the prescription table in MIMIC-III as the target of our prediction. The advantage of using this column is that the creators of MIMIC-III marked it as not null and consequently filled every row with data. Figure 3.1 shows the number of admissions per drug. The drugs are on the x-axis, and the y-axis how many admissions use a specific drug. We sorted the drugs on the x-axis by the amount admissions using a specific drug. As we can see, the data is long-tail.

3.4.2 Admission Notes

We used the Admission Note dataset proposed by van Aken et al. ((2021)). They created the data from discharge notes. Then, they split the discharge notes into sections with simple pattern matching. After that, they reviewed the section titles with medical doctors to only select sections which contain information known at admission. These sections include *Chief complaint, (History of) Present illness, Medical history, Admission Medications, Allergies, Physical exam, Family history and Social history.* Afterward, they filtered the discharge notes to only include these sections. They also make sure to filter out notes from newborns and removed duplicates. There are 48745 admission notes in total in the dataset. We split the dataset randomly into 3 parts train(70%), test(20%) and validation(10%). The following box shows an example admission note from a fictional person:

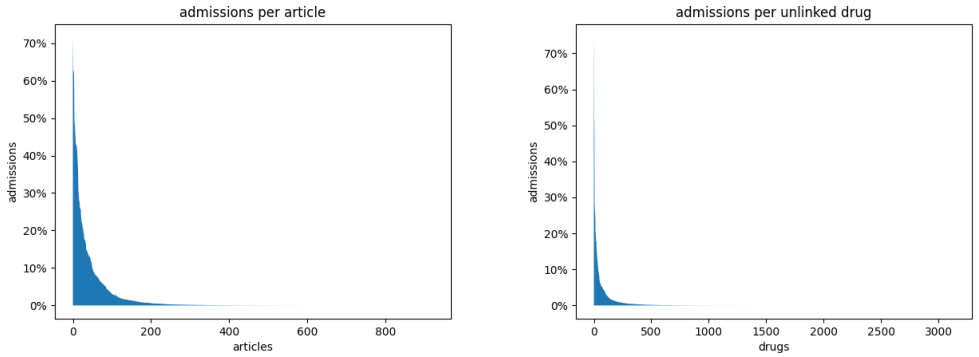
```
CHIEF COMPLAINT: Headaches

PRESENT ILLNESS: 58yo man w/ hx of hypertension, AFib on coumadin presented to ED
    with the worst headache of his life. Brother reports states that patient has
    been complaining of headache for 2 days and that the patient has lost
    consciousness. He had a syncopal episode and was intubated by EMS.

MEDICATION ON ADMISSION: 1mg IV ativan x 1, metformin

PHYSICAL EXAM: Vitals: P: 92 R: 14 BP: 151/78 SaO2: 99\% intubated. Cardiac: RRR.
    GCS  E: 3   V:2  M:5 HEENT: atraumatic, normocephalic Pupils: 4-3mm. Abd: Soft,
     BS+ Extrem: Warm and well-perfused.

FAMILY HISTORY: Mother had stroke at age 82. Father unknown.

SOCIAL HISTORY: Lives with wife. 25py. No EtOH
```

3.4.3 Wikipedia

We used Wikipedia as a source for descriptions of drugs. Therefore, we use a Wikipedia dump. There are 5676817 articles and 27818908 sections in Wikipedia. Furthermore, we use the introduction section as it is present in every Wikipedia document and summarizes its content well. Using the summarized version of the document is favorable because the quadratic memory requirement of the attention mechanism limits the amount of text that we can process at a time with BERT.

3.4.4 Trigram Fuzzy Matching

To use Wikipedia as our source of descriptions, we have to match Wikipedia documents to prescriptions. We tried creating a mapping from one of the coding system representations provided by MIMIC to Wikipedia using Wikidata, but we have not found a mapping that yielded satisfactory results. Therefore, we used fuzzy Trigram matching to match Wikipedia titles to drug names.



(a) Prescriptions per article (b) Prescriptions per unliked drug

Figure 3.2: Statistics of the linked data

The matching works with a similarity function. This similarity function takes two inputs and returns how well the input values match by outputting a value between 0 (not matching) and 1 (identical). Therefore, the algorithm splits the input values into a set of Trigrams. A Trigram is a string containing three characters. The algorithm expresses the split input as a vector. Each dimension of the vector is assigned one of the possible Trigrams. For example, if our alphabet has 26 letters, the vector has 26^3 dimensions. For each Trigram in the input, we set the corresponding dimension in the output vector to 1 while the remaining stay 0. Finally, We obtain the similarity by taking the cosine similarity between the two vectors.

We found that similarity of 0.8 provides a good enough matching while linking as many drugs as possible. Therefore, we were able to match 1390 from 4430 or approximately 31%. Figure 3.2a shows the number of admissions per Wikipedia article of the drugs used. On the x-axis are the articles, and on the y-axis, the number of admissions. We sort the articles on the x-axis by the number of admissions. As we can see, this data is also long-tail. Figure 3.2b shows the number of admissions per drug, which we could not link to a Wikipedia article. On the x-axis are the drugs, and on the y-axis, the number of admissions. We sort the drugs by the number of admissions. As we can see, we could not resolve some of the most used drugs.

We could not link some drugs because they are not widely known, and there is no Wikipedia article. However, this does not explain why some of the most used drugs are not linked. Looking into the set of unlinked drugs, we discover that drugs named by acronyms, like Intravenous sugar solution (D5W), could not be disambiguated by our method. The Wikipedia search could disambiguate these acronyms because they use redirection pages for synonyms and acronyms of a topic. Unfortunately, our dataset does not contain the redirects. Furthermore, some drugs point to disambiguation pages. Wikipedia uses these disambiguation pages if multiple topics share the same name or acronym. In this case, the description of the drug contains the introduction of the disambiguation page, which is a short sentence like, "X may refer to:" or "X as an abbreviation can mean:".

3.5 Approaches

3.5.1 Classification

In the classification approach, we take an admission note as input. The output is a vector where every dimension corresponds to a prescribable drug. This approach limits the number of drugs we can predict, and we have to retrain the model every time we want to add new drugs. Nevertheless, this model should provide a baseline for the following models.

We use a pre-trained BioBERT model(Lee et al. ((2019))), which we fine-tune in classification configuration like described by Devlin et al. ((2019)). First, we used the word piece tokenizer to tokenize the admission note text. Then, We prepend the tokenized sequence with the [CLS] token and separate the sentences with the [SEP] token. For example, an input could look like this: *[CLS] my dog is cute [SEP] he likes play ##ing [SEP]* We then take the output from the [CLS] token from BERT and pass it through a classification head to obtain the prediction.

The classification head consists of a linear layer with sigmoid activation. Then, we use binary cross-entropy loss to obtain the loss for the model. Additionally, we combine the sigmoid activation and binary cross-entropy for better numeric stability while training. The range of the output values depends on the type. While the sigmoid activated output has values between 0.0 (not prescribed) and 1.0 (prescribed), the raw model output(logits) is not restricted in any way.

We train two versions of the model, one where we infer the classes from the drug column of the prescription table in MIMIC-III and one where we infer the classes from the linked Wikipedia articles of the prescribed drugs using trigram matching, which should result in a model which could be compared more easily with the Bi-Encoder approach. For simplicity, we will continue calling the first drug classifier and the second article classifier.

To combat the long tail mentioned in Figure 3.1 and Figure 3.2a, we limit the classes of the drug classifier top 1000 most prescribed drugs and the article classifier to the top 200 most used Wikipedia articles in regards to the linked admission. We also trained versions that do not limit the classes to measure the performance impact of including the long-tail.

3.5.2 Bi-Encoder

In the Bi-Encoder Approach, we think of the problem as a ranking problem. Just like the classification, we take an admission note as input. Additionally, we have a collection of descriptions of drugs. With the Bi-Encoder, we then rank the admission note with this collection of descriptions. The output is a vector where each dimension corresponds to the drug description, with the value representing the relevance of the description to the admission note. The advantage of this approach is that we do not have to retrain the model every time we want to add new drugs. Therefore, the Bi-Encoder could be described as a pointwise ranking approach.

We process the encoder input like in the classification approach. Furthermore, We use BERT with a feedforward on the output of the [CLS] token as encoder and dot product similarity as similarity metric.

For training, we assembled batches by taking an admission note and randomly selecting the description of the prescribed drug. Then, We encode the admission notes and the drug descriptions and calculate the similarity of each encoded admission note and prescribed drug using the Bi-Encoders similarity measure. After that, we use two methods to calculate the loss of our model. First, we use single-label loss (cross-entropy with softmax activation) where for one admission note, the model should only predict the corresponding selected drug. We expressed this with a diagonal matrix. However, a selected drug might also be relevant for another admission note, which introduces noise with this setup. Therefore, the second method used multi-label loss (binary cross-entropy with sigmoid activation) where for one admission note, the model should predict all relevant drugs in the batch. We expressed this by a matrix where every cell is set to 1 if the corresponding drug is relevant for the corresponding admission note and 0 if the previous does not apply. We found that the second approach performs much better than the first.

As for the Bi-Encoder, we limit the number of descriptions to the 200 descriptions, which are used in most admissions to combat the log tail mentioned in 3.4.1 and 3.4.4. We also trained a version where we do not limit the amount classes to show how the approaches perform with and without the long-tail.

We filtered out admission notes that only use drugs that we can not link to Wikipedia articles because the training does not represent them. As a result, we filter out 3878 of 48745 or approximately 8% of the admission notes.

3.6 Loss functions

We mentioned two loss functions in the previous sections cross-entropy and binary cross-entropy. We describe the signature of the cross-entropy function as $ce(pred, target)$ and the signature of the binary cross-entropy function as $bce(pred, target)$. The $pred$ parameter describes the model output, while the $target$ parameter describes what the model should predict. Both parameters are vectors containing a dimension for each class. The $pred$ vector contains values between 0.0 and 1.0, where higher values correspond to a higher probability of a class being present. The $target$ vector contains either 0.0(negative) or 1.0(positive).

3.6.1 Cross entropy

The following function describes cross-entropy:

$$ce(pred, target) = -\sum_{c \in C} target[c] \cdot \log(pred[c]) \tag{3.1}$$

The C set is a set containing all classes. As mentioned above the *target* vector contains either 0.0(negative) or 1.0(positive). Therefore, only predictions of positive classes can affect the loss. As we use cross-entropy as loss for single-label classification, this equation could be simplified as follows:

$$ce(pred, class) = -\log(pred[class]) \tag{3.2}$$

The class parameter describes the id of the positive class. We combine the cross-entropy loss with the softmax activation to input prediction values with a range between 0.0 and 1.0. If we combine the formulas, we get the following result:

$$\begin{aligned} ce(pred, class) &= -\log\left(\frac{\exp(pred[class])}{\sum_{c \in C} \exp(pred[c])}\right) \\ &= -pred[class] + \log\left(\sum_{c \in C} \exp(pred[c])\right) \end{aligned} \tag{3.3}$$

Then, we take the mean to obtain the loss of multiple training samples of a batch.

3.6.2 Binary cross entropy

As mentioned in 3.5.2, the single label loss approach using cross-entropy could introduce noise. Therefore, we use multi-label loss using binary cross-entropy. The following function describes binary cross-entropy:

$$bce(p, t) = -\sum_{c \in C} t[c] \cdot \log(p[c]) + (1 - t[c]) \cdot \log(1 - p[c]) \tag{3.4}$$

As for the cross-entropy, the C set is a set containing all classes. Thus, unlike the cross-entropy in the binary cross-entropy positive, and negative classes can affect the loss. Furthermore, we take the mean to obtain the loss of multiple training samples of a batch.

3.7 Summary

In this chapter, we first defined the task of medication prediction for this thesis, where we try to predict the active ingredient of a prescription using a textual patient representation. After that, we explained learning to rank and divided it into the pointwise, pairwise, and listwise learning to rank approaches.

Then, we presented the data we use, how it is composed and how we preprocess it. We used the prescription table's drug column from MIMIC-III proposed by Johnson et al. ((2016)) as our source of prescriptions. We showed that the data is long-tail, meaning a few drugs are used in almost every admission while the majority only a few times. Additionally, we used the admission note dataset proposed by van Aken et al. ((2021)) as a source for textual patient representations in the form of admission notes. Furthermore, we used Wikipedia as a source for drug descriptions and used Trigram fuzzy matching to match Wikipedia titles to drug names and discussed the implications of this approach.

Afterward, we presented the different approaches we apply to solve the task of medication prediction. First, we presented a classification approach where we assigned each drug a class and fine-tuned a BioBERT model proposed by Lee et al. ((2019)) to predict the used drugs from an admission note using multi-label classification proposed by Devlin et al. ((2019)). Moreover, we used the Bi-Encoder proposed by Humeau et al. ((2020)) architecture as pointwise learning to rank approach with two BERT-based encoders to rank drugs using their probability of being used in an admission using an admission note and a drug description from Wikipedia. Additionally, we proposed a multi-label approach, which promises to evade noise introduced by the single-label approach. Following that, we explained the used loss functions, which are cross-entropy and binary cross-entropy.

Chapter 4

Implementation

4.1 Introduction

In this chapter, we describe the experimental setups for the training of the proposed architectures. First, we will give insight into the data loading and preprocessing pipeline. After that, we will describe the experimental setups for the different approaches. Finally, we will explain the design of the hyperparameter optimization we run.

4.2 Data Setup

We use Postgres as the database for all the data used. The advantage of this approach over using the raw Comma Separated Values (CSV) or JSON files is that the processing and transformation of these files are slow if we implement it in python. The usual approach is to implement parts of the data processing pipeline in native code. However, our Experience has shown that it is impossible to shift everything into native code most time, and the parts that we could not move could severely impact performance. Therefore, we use Postgres, which the developers completely implemented in C. Postgres uses SQL as its query language, which is declarative, meaning that we specify the representation of the data you want to retrieve, saving on costly transformations on the python side. Furthermore, it is most certainly not possible to write faster code for the retrieval of some data in the time you write an SQL query. Nearly every language supports libraries to access Postgres via SQL, making the SQL-Queries independent of the language used. This approach also has workflow benefits as the developers build Postgres with Multiuser use in mind. Data imported could be used by other users, which is beneficial in a research group setting.

The MIMIC-III data(Johnson et al. ((2016))) was available to us in CSV format. We use the buildmimic[1] scripts to import the data provided by MIMIC-III into the database. The Admission note dataset(van Aken et al. ((2021))) was also available to us in CSV format. Therefore, we added our scripts to import the admission note data. The following line describes the SQL commands we use. Note that we will use the Postgres SQL dialect throughout the thesis you, might need to adapt the commands if you use another database:

[1] `https://github.com/MIT-LCP/mimic-code/tree/main/mimic-iii/buildmimic`

```
-- Create admission not table
CREATE TABLE admission_notes
(
  ID INT NOT NULL,
  TEXT TEXT,
  HOSPITAL_EXPIRE_FLAG INT,
  CONSTRAINT admission_notes_id_pk PRIMARY KEY (ID)
);
```

The command above creates the table containing the admission notes. In addition, it includes an id as primary key, the text of the admission notes, and a hospital expire flag, which we do not use.

```
-- Import admission note data
\copy ADMISSION_NOTES from 'MP_IN_adm_test.csv'
  delimiter ',' csv header NULL ''
\copy ADMISSION_NOTES from 'MP_IN_adm_val.csv'
  delimiter ',' csv header NULL ''
\copy ADMISSION_NOTES from 'MP_IN_adm_train.csv'
  delimiter ',' csv header NULL ''
```

The commands above import the data into the admission note table. We use the Postgres specific copy function to import the CSV data.

We use SQLAlchemy by Bayer ((2012)) as ORM to request the data from Postgres using SQL. To create our database schema, we use the reflection feature from SQL alchemy. The following code shows how we do it:

```
engine = create_engine(connection)
```

The code above shows how to create an SQLAlchemy engine, which connects to the database. The connection variable is a string containing the information about the database driver, username, password, host, and name.

```
metadata = MetaData()
metadata.reflect(bind=engine)
```

The code above shows how to use the reflection feature from SQLAlchemy to infer the database's schema.

```
prescriptions = self.metadata.tables["prescriptions"]
admission_notes = self.metadata.tables["admission_notes"]
articles = self.metadata.tables["articles"]
sections = self.metadata.tables["sections"]
```

The code above shows how we access the tables from the inferred schema.

To import the Wikipedia data, we use an XML Wikipedia database dump, which we processed and converted to JSON, using Gensim by Řehůřek and Sojka ((2010)). Gensim extracts the article title, section titles, and section texts for every Wikipedia Article. Moreover, we used SqlAlchemy to insert the data into Postgres. Furthermore, we store Wikipedia articles and sections in separate tables instead of joining the text and storing it inside a row of the articles table. This approach makes it easier to query certain sections like the introduction section, which becomes useful later in the thesis. The following code shows how we do it:

```
meta = MetaData()

articles = Table(
  "articles", meta,
  Column("id", Integer, primary_key=True),
  Column("title", String(300), nullable=False),
)

sections = Table(
  "sections", meta,
  Column("id", Integer, primary_key=True),
  Column("article_id", Integer, ForeignKey("articles.id"), nullable=False),
  Column("title", String(3000), nullable=False),
  Column("text", Text, nullable=False),
)

meta.create_all(engine)
```

First, we create the new tables using the code shown above.

```
connection = engine.connect()
```

Then we connect to the database using the command shown above. We use a SqlAlchemy connection rather than a session because it allows us to send raw SQL commands to the database, which we need for later operations such as creating indexes, creating tables from select statements, and altering tables.

```
segment_and_write_all_articles(
  wikipedia_path,
  wikipedia_segment_path
)
```

Afterward, we use `segment_and_write_all_articles` function from Gensim to extract article titles, section titles and texts. It saves the result of this function as JavaScript Object Notation (JSON) in a file. The function takes two parameters, the path to the XML Wikipedia dump and the output file.

```
with utils.open(str(wikipedia_segment_path), "rb") as file:
  section_id = 0

  for article_id, article in enumerate(map(ujson.loads, file)):
    connection.execute(insert(articles), {
      "id": article_id,
      "title": article["title"]
    })

    for section_title, section_text in zip(article["section_titles"], article["
    section_texts"]):
      connection.execute(insert(sections), {
        "id": section_id,
        "article_id": article_id,
        "title": section_title,
        "text": section_text
      })
      section_id += 1
```

The code shows shown above how we read the Gensim output file and insert the database using SQLAlchemy.

We also create indexes for the primary key of the article and sections table and the `articles_id` column of the sections table using the following SQL commands:

```
-- Primary key indexes
CREATE INDEX IF NOT EXISTS articles_idx
  ON articles(id);
CREATE INDEX IF NOT EXISTS sections_idx
  ON sections(id);

-- Foreing key index
CREATE INDEX IF NOT EXISTS sections_article_idx
  ON sections(article_id);
```

To create the Trigram matching between prescribed drugs and Wikipedia titles, we used the Postgres Trigram extension. Therefore, we used the following SQL commands to accomplish this.

```
CREATE INDEX IF NOT EXISTS title_idx
  ON articles
  USING GIN (title gin_trgm_ops);
```

First, we create a Trigram index to search the Wikipedia titles faster. This index acts just as a speed-up for the following command, and we do not use this index further.

```
CREATE TABLE IF NOT EXISTS drug_wikipedia_title_trgm_mapping
  AS (
    SELECT
      drugs.drug as drug,
      articles.id as article_id,
      similarity(articles.title, drugs.drug) as similarity
    FROM
      articles,
      (
        SELECT
          DISTINCT prescriptions.drug
        FROM
          prescriptions
      ) AS drugs
    WHERE
      articles.id = (
        SELECT
          articles.id
        FROM
          articles
        WHERE
          articles.title %% drugs.drug
        ORDER BY
          similarity(articles.title, drugs.drug) desc
        LIMIT
          1
      )
);
```

Then we create the matching table using an SQL Select with the SQL statement mentioned above. The Select query takes the distinct set of drugs and finds the article with the best matching title for every drug using the before-mentioned Trigram index. We also store the trigram similarity of the best match in this table for later use.

```
ALTER TABLE drug_wikipedia_title_trgm_mapping
  ADD PRIMARY KEY(drug);
ALTER TABLE drug_wikipedia_title_trgm_mapping
  ADD FOREIGN KEY(article_id)
  REFERENCES articles(id);
```

Lastly, we alter the created table by adding the `drug` column as primary key and the `article_id` as a foreign key to the articles table.

4.3 Dataloading

4.3.1 Drug Classifier

To generate the classes for the drug classifier mentioned in 3.5.1 we use the following query:

```
SELECT
  drug
FROM
  prescriptions
GROUP BY
  drug
ORDER BY
  count(*)
LIMIT
  $class_limit
```

Listing 4.1: drug classifier classes query

The query orders the drugs by the number of prescriptions. This approach allows us to select the most prescribed medications. Additionally, we limit the list by `$class_limit` which we replace by the number of classes the classifier should predict. Furthermore, we create an id mapping in python for the result of the query where we assign each drug a class id between 0 and `$class_limit`.

After that, we use the following query to gather the positive classes for every admission note:

```
SELECT
  hadm_id, drug
FROM
  prescriptions
WHERE
  drug in $drugs
```

We represent the positive classes as a mapping from admission notes id to drugs. Furthermore, we use the `hadm_id` and `drug` columns from the prescription table to obtain the data for the mapping. In addition, we filter out prescriptions that contain drugs that we do not want to classify. Therefore, `$drugs` we replace by the result of Listing 4.1. To obtain the mapping, we group the result of this query with python by the `hadm_id` and map the drug to its assigned class id.

Finally, we use the following query to gather the admission notes:

```
SELECT
  id, text
FROM
  admission_notes
ORDER BY
  id ASC
```

We order by the `id` to obtain the admission notes in the same order for every run.

4.3.2 Bi-Encoder

For the Bi-Encoder, we use the following query to select the Wikipedia articles we want to use:

```
SELECT
  article_id
FROM
  (
    SELECT DISTINCT
      drug_wikipedia_title_trgm_mapping.article_id as article_id,
      prescriptions.hadm_id as hadm_id
    FROM
      prescriptions,
      drug_wikipedia_title_trgm_mapping,
      sections
    WHERE
      prescriptions.drug = drug_wikipedia_title_trgm_mapping.durg
      AND drug_wikipedia_title_trgm_mapping.article_id = sections.article_id
      AND drug_wikipedia_title_trgm_mapping.similarity >= 0.8
      AND sections.title = 'Introduction'
  )
GROUP BY
  article_id
ORDER BY
  count(*)
LIMIT
  $class_limit;
```

Listing 4.2: Bi-Encoder Wikipedia articles query

This query gathers the Wikipedia article ids of the drugs and orders them by the number of admissions the corresponding drug is used. This approach allows us to select the Wikipedia articles of the drugs which medical doctors most use. The query is nested. The inner Select statement creates distinct Wikipedia article ids and admission ids from the prescriptions table. The trigram similarity between the prescribed drug and the mapping table is greater or equal to 0.8, which corresponds to the similarity mentioned in 3.4.4. Also, the corresponding Wikipedia article should have an introduction section. The outer select then takes this set and groups it by the Wikipedia article id and orders it by the number of admissions. As before, we replace `$class_limit` by the number of drugs the Bi-Encoder should predict. As mentioned in 3.4.4, we found that similarity of 0.8 provides a good enough matching while linking as many drugs as possible. Also, as mentioned in 3.4.3, we use the introduction section as it is present in every Wikipedia document and summarizes the content of it well.

After that, we use the following query to obtain the article text:

```
SELECT
  article_id,
  text
FROM
  sections
WHERE
  article_id in $article_ids
  AND title = 'Introduction'
```

In this query, we use the `sections` table, filter out Wikipedia articles we do not want to use, and only take the introduction section. `articles_ids` is replaced by the Wikipedia article ids we wish to use.

As mentioned in 3.5.2, we filtered out admission notes which do not have any Wikipedia articles containing drug descriptions of drugs used in this admission. We use the following query to create a distinct set of admission ids that meet this requirement:

```
SELECT DISTINCT
  prescriptions.hadm_id
FROM
  prescriptions,
  drug_wikipedia_title_trgm_mapping
WHERE
  prescriptions.drug = drug_wikipedia_title_trgm_mapping.drug
  AND drug_wikipedia_title_trgm_mapping.article_id in $article_ids
```

Listing 4.3: Bi-Encoder admissions query

This query creates the above-mentioned distinct set by filtering the prescriptions by the Wikipedia article id containing the description of the prescribed drug. Therefore, we replace `$article_ids` by the result of Listing 4.2.

Afterward, we obtain the admission note texts using the following query:

```
SELECT
  id,
  text,
FROM
  admission_notes
WHERE
  id in $admission_note_ids
ORDER BY
  id ASC
```

Listing 4.4: Bi-Encoder admission note text query

For this query, we filter out admission notes we do not want to use. Therefore, we replace `$admission_note_ids` by the result of Listing 4.3. As for the drug classifier, we order the result by the admission id to obtain the admission notes in the same order for every run.

Then, we use the following query to obtain a mapping from admissions notes to Wikipedia articles containing the descriptions of the drugs used in this admission:

```
SELECT DISTINCT
  prescriptions.hadm_id,
  drug_wikipedia_title_trgm_mapping.article_id
FROM
  prescriptions,
  drug_wikipedia_title_trgm_mapping
WHERE
  prescriptions.drug = drug_wikipedia_title_trgm_mapping.drug
  AND drug_wikipedia_title_trgm_mapping.article_id in $article_ids
```

Listing 4.5: Bi-Encoder admission Wikeedia drug article mapping

We use the `prescription` and `drug_wikipedia_title_trgm_mapping` tables to obtain the data for the mapping. Furthermore, we filter out the Wikipedia articles we do not want to use. Therefore, we replace `$article_ids` with the Wikipedia article ids we want to use. To obtain the mapping, we group the results of this query with python by the `hadm_id`.

4.3.3 Article Classifier

As mentioned in 3.5.1, we created the article classifier to obtain results, which we could compare more easily with the Bi-Encoder approach. Therefore, we can reuse a lot of the code used for the Bi-Encoder data loading.

To obtain the classes of the article classifier, we use the query from Listing 4.2. Like the drug classifier, we assign each article id a class id between 0 and `$class_limit`.

As for the Bi-Encoder approach, we filtered out admission notes which do not have any Wikipedia articles containing drug descriptions of drugs used in this admission. Therefore, we use the query from Listing 4.3. We then use the query from Listing 4.4 to obtain the admission note texts.

Afterward, we use the query in Listing 4.5 to obtain the positive classes for every admission note. Then, we represent the positive classes as a mapping from admission note to Wikipedia article id. To create the mapping, we group the query result by the `hadm_id` and map the `article_id` to its assigned class id.

4.4 PreProcessing

We use the BERT word piece tokenizer from the Huggingface Transformers library to tokenize text content for BERT. The tokenizer also automatically adds the special tokens like *[CLS]* or *[SEP]*. We use the following code to load the tokenizer:

```
tokenizer = BertTokenizerFast.from_pretrained(bert_path, use_fast=True)
```

The `bert_path` variable contains the name of the pretrained BERT model. We use the `dmis-lab/biobert-v1.1` model as base for all of our experiments. Furthermore, we use the fast variant of the BERT tokenizer which is implemented in Rust instead of Python. Then, we use the following code to tokenize our text input:

```
tokenized = self.tokenizer.encode_plus(
  text,
  truncation=True,
  max_length=max_length,
  padding='max_length')
```

The `text` variable contains the text we want to tokenize. The `encode_plus` function accepts text in different formats. In our case, it accepts either a string containing a sentence or an array of strings containing multiple sentences. Furthermore, we truncate tokenized input if it gets longer than `max_length` and pad it to `max_length` if it gets shorter. This approach should make batching easier and should prevent that the training runs out of memory. After that, we use the following code to access the tokenized input.

```
input_ids = tokenized['input_ids']
attention_mask = tokenized['attention_mask']
```

`input_ids` is the tokenized input, while the `attention_mask` is the attention mask. The attention mask masks padded tokens.

4.5 Experimental Setup

We use PyTorch by Paszke et al. ((2019)) as the framework for hardware-accelerated deep learning. Therefore, we have implemented most parts of our training and evaluation pipeline in python. Also, we used PyTorch-Lightning by Falcon ((2019)) to add more structure to our code and reduce boilerplate. Furthermore, we use the Huggingface Transformers library by Wolf et al. ((2020)), which allows us to quickly load and use BERT and other pre-trained transformer-based models with PyTorch.

We use the following code to load pretrained BERT models:

```
bert = BertModel.from_pretrained(bert_path)
```

The `bert_path` variable contains the name of the pre-trained BERT model. We use the `dmis-lab/biobert-v1.1` model as the base for all of our experiments. Then we use the model as mentioned below to obtain embeddings.

```
embeddings = self.bert(input_ids, attention_mask=attention_mask, return_dict=False)
    [0]
```

The `input_ids` and `attention_mask` variables are the output from the tokenizer mentioned in 4.4. Additionally, we input multiple tokenized inputs as a batch at once. With `return_dict` set to `False` the model returns a tuple containing a vector with the model output. The returned tuple only has one entry. Therefore, we are extracting it using the `[0]` operation. In both the Classifier and Bi-Encoder, we use the output from the *[CLS]* token, which is the first token. To access this vector, we use the following code:

```
embeddings[:, 0]
```

Because we input multiple tokenized inputs as a batch at once, we need the whole slice of the first dimension containing the output from every item in the batch. So we then take the first output from the second dimension, including the output for every input token.

We trained the models inside a Kubernetes cluster and assigned one Nvidia A100 40GB GPU to each training instance.

4.5.1 Classifier

We trained four variants of the model. Two variants, we called drug classifier and article classifier in 3.5.1. The remaining variants come from both variants being trained with and without the long-tail. Furthermore, we use the Adam optimizer with a weight decay of 0.1 and a constant learning rate scheduler with 500 warmup steps. Also, We trained the models with 32-bit precision.

4.5.2 Bi-Encoder

We trained four variants of the model, two with single label loss and two with multi-label loss as mentioned in 3.5.2. Like for the classification approach, The remaining variants come from both variants being trained with and without long-tail. Additionally, We used the Adam optimizer with a weight decay of 0.1 and a constant learning rate scheduler with 500 warmup steps. We also trained the models with 32-bit precision.

4.6 Hyperparameter Optimization

We run Hyperparameter Optimization (HPO) to determine the optimal hyperparameters to run our experiments. Therefore, we used RayTune by Liaw et al. ((2018)). Unlike the training, we run the hyperparameter optimization at 16 Bit precision. Table 4.1 and Table 4.2 show what hyperparameters we tune for each approach and how we tune them.

Hyperparameter	Tuning method
Learning Rate	loguniform$(1 \cdot 10^{-5}, 5 \cdot 10^{-5})$
Batch Size	$16, 32$

Table 4.1: Classifier HPO

Hyperparameter	Tuning method
Learning Rate	loguniform$(1 \cdot 10^{-5}, 5 \cdot 10^{-5})$
Batch Size	$8, 16$
Activation Function	tanh, mish, sigmoid, no activation

Table 4.2: Bi-Encoder HPO

4.7 Summary

In this chapter, we first gave insights about the inner workings of our PostgreSQL based on the data preprocessing pipeline. Then, we showed how we import the data into PostgreSQL, which is available in different formats. Moreover, we showed how we use SQL to request the data from PostgreSQL, saving on poorly performing transformations on the Python side or time and memory expensive Python-based preprocessing steps. Additionally, we showed how we use the BERT word piece tokenizer from the Huggingface Transformers library for text sequence tokenization.

Next, we explain the general experimental setups of the different approaches we employ. Then, we showed how we use the BERT model from the Huggingface Transformers library to encode tokenized sequences. Finally, we present our hyperparameter optimization setup for the different approaches and what hyperparameters we are tuning.

Chapter 5

Evaluation

5.1 Introduction

In this chapter, go over our findings and how we archived them. First, we present our hypothesis. After that, we offer the evaluation metrics for the different approaches. Afterward, we present our results of the hyperparameter optimizations and experiments for the other models. Finally, we discuss our findings and explain what conclusions we draw from them.

5.2 Hypothesis

Bi-Encoder better than Classifier We expect the Bi-Encoder approach to perform better than the classifier approach because the classifier does not have any additional information about the classes representing drugs such as effects, risks, and so forth. We expect the classifier to infer this information while training. However, We provide this information to the Bi-Encoder with drug descriptions from Wikipedia. We hope this will help the Bi-Encoder create a relation between admission notes and drugs and therefore boost prediction performance.

Training with long-tail should perform significantly worse We expect model variants trained with long-tail to perform considerably worse than variants without it because the classes in the long-tail have fewer examples than the other classes. Therefore, they emerge less often, and also, fewer inputs leading to this class means the model does not generalize well on these classes. Especially, We expect that metrics that weight all classes equally perform even worse because we expect the prediction of drugs with fewer examples to be worse.

Multi-label Bi-Encoder should perform better than single-label Bi-Encoder We expect the Bi-Encoder variant using the multi-label loss to perform better than the variant using the single-label loss. As mentioned in 3.5.2 for the single-label loss, the model should only predict the selected drug as relevant for an admission note in a batch. This approach trains additional noise as other selected drugs might be relevant for an admission note in a batch. Thus, we expect the multi-label loss approach to perform better as its representation includes all relevant drugs for an admission note.

The precision score should be worse than the recall score We expect the precision score to be worse than the recall score for the classifier because we have a label imbalance meaning more negative than positive results due to the long-tail of the data. Therefore, it is easier to create false-positive samples than false-negative samples as a random positive prediction is more likely to be false than a random negative prediction. Therefore, we expect the precision score to be worse than the recall because false-positive predictions worsen the precision, and false-negative worsen the recall.

Bigger training batch sizes for the Bi-Encoder should yield better performance We expect that bigger batch sizes for the Bi-Encoder training yield better prediction performance because the number of relations between admission notes and drugs, the model has to predict for a batch rises by the square of the batch size. Therefore, the problem becomes harder for bigger batch sizes, and we expect this to lead to better performance.

5.3 Classification Evaluation Metrics

For evaluation, we take the sigmoid activated output mentioned in 3.5.1. As mentioned, the classification output is a vector where each dimension corresponds to a prescribable drug, with a value between 0.0 (not prescribed) and 1.0 (prescribed). The values of the vector are not binary but need to be because the target is binary. A drug can either be prescribed or not. Therefore, we use a threshold to obtain a binary value for every prescribable drug. If the value is above the threshold, we assume the drug as prescribed, and if it is below, we assume the drug as not prescribed. For all metrics except area under ROC (AUROC), we used a threshold of 0.5.

5.3.1 Accuraccy

The accuracy score is the ratio between the amount true predictions and the number of predictions. The disadvantage of the accuracy score is that it can be misleading for imbalanced datasets. Even so, it is widely used due to its simplicity. Therefore, we do not restrict our evaluation to the accuracy score. The following formula shows how we calculate the accuracy:

$$accuracy = \frac{|\{true\ predictions\}|}{|\{predictions\}|} \tag{5.1}$$

5.3.2 Recall

The recall score describes how many of the positive samples the model predicts positively. In our case, a high recall means that the predicted drugs contain most of the prescribed drugs. We use this metric to measure if the model predicts the values we want to predict. The following formula shows how we calculate the recall:

$$recall = \frac{|\{true\ positive\ predictions\}|}{|\{positive\ samples\}|} \tag{5.2}$$

5.3.3 Precision

The precision score describes how many of the positive predictions are actually positive. We calculate it by dividing the amount true-positive predictions by the number of positive predictions. A high precision score means that the prediction does not contain many wrong values. We use this metric to measure the noisiness of the output. The following formula shows how we calculate the Precision score:

$$precission = \frac{|\{true\ positive\ predictions\}|}{|\{positive\ predictions\}|} \tag{5.3}$$

5.3.4 F1

The F1 is the harmonic mean of precision and recall. A model with high F1 scores usually features good recall and precision. The following formula shows how we calculate the F1 score:

$$F_1 = \frac{2}{precission^{-1} + recall^{-1}} = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.4}$$

5.3.5 AUROC

The AUROC is the integral of the receiver operating characteristic (ROC) curve. This curve plots between the true positive rate (TPR) on the y-axis and the false positive rate (FPR) on the x-axis with different classification thresholds. Thresholds between 0.0 and 1.0 are used. The true positive rate is a synonym for recall. The following formulas show how the TPR and FPR is calculated:

$$\begin{aligned} recall = TPR &= \frac{|\{true\ positive\ predictions\}|}{|\{positive\ samples\}|} \\ FPR &= \frac{|\{false\ positive\ predictions\}|}{|\{negative\ samples\}|} \end{aligned} \tag{5.5}$$

We can interpret the AUROC score as the probability that the model ranks a random positive example higher than a random negative example. We use the macro AUROC variant, which calculates the AUROC for each class separately and then takes the mean over every class to obtain the final value.

5.4 Bi-Encoder Evaluation Metrics

As mentioned in 3.5.2, the output of the Bi-Encoder is a vector where each dimension corresponds to a drug description with the value representing the relevance of the description to the admission note. Because we use the dot product similarity, the output values are positive but not restricted upward. We interpret the model output as ranking, where lower values correspond to a higher rank for Recall@k and NDCG.

5.4.1 AUROC

Because the output of the dot product similarity is positive and not restricted upwards and the AUROC score needs values between 0.0 and 1.0, we use the sigmoid function to limit the values of the output between 0.0 and 1.0. We use this metric to compare the prediction results of the classification approach with this approach.

5.4.2 Recall@k

As mentioned above, we interpret the model output as ranking. As the user of such ranking most likely will not look through all the ranked results. Therefore, this metric looks at the top-most ranked items. We specify the number of items we consider with k. For example, with a k of 5, we consider the 5 top-most results. The Recall@k describes how many items in the k top most ranked results are relevant/positive. The following formula shows how we calculate the Recall@k score:

$$recall@k = \frac{|\{relevant\ predictions\} \cap \{top\ k\ highest\ ranked\ items\}|}{\max(|\{relevant\ items\}|, k)} \tag{5.6}$$

To not negatively affect the result, we take The maximum between the number of relevant items and k. We do this as the k top highest ranked results could at most contain k relevant items. Thus, a high Recall@k corresponds to the result having many or near to all relevant items. We measure the Recall@k for a k of 1, 5, 10, and 32.

5.4.3 NDCG

The Normalized Discounted Cumulative Gain (NDCG) is calculated by dividing the Discounted Cumulative Gain (DCG) by the Ideal Discounted Cumulative Gain (IDCG). In contrast to the Recall@k the NDCG, DCG and IDCG could also work with non-binary relevances. In our case, we use binary relevances like for the Recall@k. Like the Recall@k the NDCG, DCG and IDCG could be limited to the k top most ranked items. We do not use this limit in our case, and we could consider k to be set to the number of ranked documents. The DCG is the sum of the relevance of the ranked items weighted by their position. The following formula shows how we calculate the DCG score:

$$DCG_k = \sum_{i=1}^{k} \frac{relevance_i}{log_2(i+1)} \tag{5.7}$$

The disadvantage of the DCG is that the number of relevant items and the range of relevance values could have an impact on the score. Therefore, we divide the DCG by the IDCG to obtain the NDCG. The IDCG is calculated the same way as the DCG, but now we use the ideal ranking. The following formulas shows how we calculate the IDCG, and NDCG score:

$$IDCG_k = \sum_{i=1}^{k} \frac{truth_i}{log_2(i+1)} \tag{5.8}$$

$$NDCG_k = \frac{DCG_k}{IDCG_k} \tag{5.9}$$

We calculate the NDCG for every evaluation sample and take the mean of these values to obtain the final score.

5.5 Preprocession

5.5.1 AUROC

As mentioned above, we calculate the AUROC for each class separately. Therefore, we filter out classes with only positive or only negative samples to not divide by 0.0 when calculating the true-positive or true-negative rates.

5.6 Results

5.6.1 HPO Classifier

We use HPO to determine the best hyperparameter combination for our experiments. In 4.5.1, we mentioned we trained 4 variants of the model. We ran HPO on two variants, the drug classifier without long-tail and the article classifier without long-tail. The variants with long-tail inherit the hyperparameters from the variants without long-tail.

We found no significant difference between the model performance and the batch sizes used. We show the hyperparameters we settled on in Table 5.1. The models postfixed with "All" are variants with long-tail while the others are variants without long-tail.

Model	Batch Size	Learning Rate
Drug Classifier	16	$4 \cdot 10^{-5}$
Drug Classifier All	16	$4 \cdot 10^{-5}$
Article Classifier	32	$4 \cdot 10^{-5}$
Article Classifier All	32	$4 \cdot 10^{-5}$

Table 5.1: Classifier hyperparameters

5.6.2 HPO Bi-Encoder

As for the classifier, we use HPO to determine the best hyperparameter combination for our experiments. In 4.5.2, we mentioned we trained 4 variants of the model. We ran HPO on two variants, the multi-label loss Bi-Encoder without long-tail and the single-label loss Bi-Encoder without long-tail. The variants with long-tail inherit the hyperparameters from the variants without long-tail.

We found that bigger batch sizes lead to better performance. Furthermore, we found that the sigmoid activation function performs significantly worse than the others, while they perform more or less the same. We show the hyperparameters we settled on in Table 5.1. The model containing "MC" in the name refers to the multi-label loss variants, while the others refer to the single-label loss variants. The models postfixed with "All" are variants with long-tail while the others are variants without long-tail.

Model	Batch Size	Learning Rate	Activation Function
Bi-Encoder	16	$1.5 \cdot 10^{-5}$	mish
Bi-Encoder All	16	$1.5 \cdot 10^{-5}$	mish
Bi-Encoder MC	16	$1.5 \cdot 10^{-5}$	mish
Bi-Encoder MC All	16	$1.5 \cdot 10^{-5}$	mish

Table 5.2: Bi-Encoder hyperparameters

5.6.3 Experiments Classifier

Model	Accuracy	Precision	Recall	F1	AUROC
Drug Classifier	0.9735	0.7550	0.4165	0.5123	**0.8392**
Drug Classifier All	**0.9940**	0.7000	0.4294	0.5080	0.7799
Article Classifier	0.9224	**0.7604**	**0.5371**	**0.6135**	0.8112
Article Classifier All	0.9799	0.7590	0.5208	0.5972	0.8083

Table 5.3: Classifier results

Table 5.3 shows the results from the experiments. The naming is the same as mentioned in 5.6.1. We picked the results by selecting the highest epoch score for each metric around the highest AUROC result. The article classifier performs better when looking at the precision, recall, and f1 score. Again when looking at the AUROC score, the approaches differ not that much. The variants without long-tail overall perform slightly better than the variants with long-tail.

5.6.4 Experiments Bi-Encoder

Model	AUROC	Recall@1	Recall@5	Recall@10	Recall@32	NDCG
Bi-Encoder	0.6675	0.3888	0.3394	0.3234	0.4219	0.6614
Bi-Encoder All	0.6853	0.3294	0.2669	0.2278	0.1923	0.5270
Bi-Encoder MC	**0.7666**	**0.9550**	**0.8599**	**0.7780**	**0.7282**	**0.8870**
Bi-Encoder MC All	0.7474	0.9514	0.8584	0.7766	0.6454	0.8523

Table 5.4: Bi-Encoder results

Table 5.4 shows the results from the experiments. The naming matches the naming in 5.6.2. We picked the results by selecting the highest epoch score for each metric around the highest NDCG result. We found that the multi-label loss variants perform way better than the single-label loss variants. Furthermore, the variants without long-tail tend to perform better than the variants with long-tail.

5.6.5 Approach comparison

Model	AUROC
Article Classifier	**0.8112**
Bi-Encoder	0.6675
Bi-Encoder MC	0.7666
Article Classifier All	**0.8083**
Bi-Encoder All	0.6853
Bi-Encoder MC All	0.7474

Table 5.5: Approach comparison

We compare both approaches by the AUROC as it is the only metric that both approaches support. We also omitted the results from the drug encoder. It is only comparable to a limited extent with the other models because it is trained and validated on another result set. Also, we group models that we trained on the same result set. As you can see in Table 5.5, the AUROC of the classifier is generally higher than that of the Bi-Encoder.

5.7 Discussion

Bi-Encoder worse than Classifier approach We found that the Bi-Encoder approach performs worse than the classifier approach. Therefore, this does not confirm our hypothesis. However, we only measure the AUROC score for both models, and consequently, this is the only score we can compare. Furthermore, since the AUROC score is used for evaluating multi-label classifiers, it might not be that suitable for evaluating the Bi-Encoder, since it is a ranking model.

Training with long-tail only perform slightly worse We can show in 5.6.3 and 5.6.4 that the variants with long-tail only perform slightly worse than variants trained with long-tail. We expected the loss of the macro AUROC score should be even bigger because it is averaged over the classes and weights all classes equally. This finding indicates that the predictions of drugs with fewer examples might not impact prediction performance as much as we thought. Although this confirms our hypothesis, the remaining difference might be caused by not separately optimizing hyperparameters on these models. Nevertheless, it is not worth it to train without long-tail because rarely used drugs are particularly interesting because they are also considered less frequently.

Multi-label Bi-Encoder performs significantly better than single-label Bi-Encoder We found that the Bi-Encoder variant using the multi-label loss performs considerably better than the variant using the single-label loss. Therefore, we expected a less significant increase in performance between the model variants. Still, the additional noise in the data of the single-label loss variants seems to cause considerable damage to the prediction performance of these variants. This finding confirms our hypothesis.

Precision score is better than recall score All variants of the classifier feature a better precision score than the recall score, even the variants with long-tail. This finding refutes our hypothesis, and the classifier predicting fewer labels overall with higher accuracy might cause these scores.

Bigger training batch sizes for Bi-Encoder yield better performance We show in 5.6.2 that bigger training batch sizes lead to better performance, which confirms our hypothesis. However, although we only tested two batch sizes, the gain might slow down with bigger batch sizes suggesting that there might be a trade-off between batch size and computational efficiency.

Bi-Encoder performs significantly worse with sigmoid activation We found that the Bi-Encoder performs considerably worse if we use sigmoid activation for the feed-forward layers of the encoders. We did not expect this outcome since all the other activation functions seem to perform fine with similar good results while sigmoid is a negative outlier. Thus, there has to be some property of this activation function that makes it particularly unfavorable. Furthermore, if the activation function significantly impacts model performance, sigmoid could not be the only function, negatively impacting model performance.

5.8 Summary

In summary, we showed that our proposed multi-label loss approach for the Bi-Encoder performs significantly better than the single-label loss approach in every measured metric for this task. This finding suggests that our hypothesis of the noise introduced by the single-label loss approach hinders the model from unfolding its full potential. Surprisingly, we showed that all approaches only perform slightly worse when trained with the long-tail and that the prediction of classes with fewer examples is not significantly worse. Another surprising finding is that the precision score is higher than the recall score for all classification experiments, even for the experiments with the long-tail. Moreover, we showed that, as expected, the Bi-Encoder approach yields better performance with bigger batch sizes. Also, surprisingly we showed the Bi-Encoder performs significantly worse with the sigmoid activation of the linear layers of the encoder blocks, suggesting that this function has some property that makes it undesirable for this task or maybe even the Bi-Encoder. Finally, surprisingly, we showed that the Bi-Encoder approach performs worse than the Classification approach, contrary to our hypothesis that the additional information from the drug descriptions augments the model performance.

Chapter 6

Conclusion

6.1 Summary

This thesis aims to find a practical approach to generate helpful drug recommendations for a medical doctor through a textual patient representation using Transformer-based neural networks. First, we presented the foundational concepts of this thesis like the Transformer Architecture by Vaswani et al. ((2017)), BERT by Devlin et al. ((2019)), and the Bi-Encoder by Humeau et al. ((2020)). Moreover, we went through the related work of the outcome prediction and medication prediction tasks.

Inspired by recent advances in outcome prediction tasks, we proposed a medication prediction task definition in the methodology. Also, we presented a medication prediction dataset based on the MIMIC-III database by Johnson et al. ((2016)), the admission note dataset by van Aken et al. ((2021)) and Wikipedia. Furthermore, we proposed a Wikipedia medication dataset which we linked to MIMIC-III using Trigram matching. Moreover, We showed that the data on prescribed drugs in MIMIC-III is long-tail. Furthermore, we proposed multiple approaches to solve our proposed task, which are a classification approach using BERT and a Bi-Encoder approach using BERT-based encoders. Likewise, we proposed two variants of the Bi-Encoder, one using single-label loss and one using multi-label loss, which promises to evade the noise introduced by the single-label loss variant.

Following the Implementation, we went through or PostgreSQL base data processing stack. Then, we gave insight into our experimental setup and presented the hyperparameters we tune and how we tune them.

Continuing with the Evaluation, we showed that our proposed multi-label loss approach for the Bi-Encoder performs significantly better for every metric measured, confirming our hypothesis that the noise introduced by the single label loss approach lowers the performance. Surprisingly, we showed that training with or without long-tail does not impact the performance as much as expected for this task and that the prediction of classes with fewer examples is not significantly worse. Also, surprisingly we showed that the precision metric is higher than the recall score for the classifier approach independent of if we trained the model with or without long-tail. We also showed that, as expected, larger batch sizes yield better performance for the Bi-Encoder. Additionally, surprisingly we showed that the Bi-Encoder performs significantly worse using sigmoid activation on top of the encoder block of the Bi-Encoder, suggesting some property of the function is undesirable for the task or maybe even the Bi-Encoder. Furthermore, we showed that surprisingly the Bi-

Encoder approach performs worse than the Classification approach, contrary to our hypothesis that the additional information from the drug description augments the model's performance.

6.2 Future Work

6.2.1 Other architectures

There most certainly will be architectures that will solve the task better than the approaches named here. In addition, there are other learning to rank architectures like the PolyEncoder or CrossEncoder proposed by Humeau et al. ((2020)) which have shown better performance for other tasks than the Bi-Encoder does.

6.2.2 Better source for drug descriptions

There exist databases like DailyMed[1] from the FDA that deliver a better source for drug descriptions. It also offers the option to resolve to the National Drug Code (NDC). Therefore, there should be fewer potential linking errors between prescribed drugs and drug descriptions. The other problem with Wikipedia is that some prescribed drugs like Natrium Chloride are linked correctly, but the introduction section contains general information about the chemical compound rather than the medical use.

6.2.3 More activation functions

We found in 5.6.2 that the activation function of the feed-forward layer of the encoders has a significant impact on model performance and that the sigmoid activation function performs significantly worse. In contrast, the other functions perform more or less the same. Thus, experiments with more activation functions may yield more insight into which properties of the sigmoid function cause it to perform significantly worse or reveal better-suited activation functions. In addition, experiments on different datasets might indicate that some activation functions might be more suitable for the Bi-Encoder architecture than others or that the choice of activation function is problem-specific.

6.2.4 Dosing

There is a range of drugs whose effect depends on other factors than the active ingredient like dose or route. However, especially the dose significantly influences the drug's effect, like Aspirin, which can be used as a blood thinner or as a painkiller depending on the dose. Therefore, Adding this information to the prediction target might be interesting, as it is not practically useful without a dose recommendation.

[1] https://dailymed.nlm.nih.gov/dailymed/

6.2.5 Longer Sequences

In our example, we used a textual representation of drugs in the form of Wikipedia articles. However, the sequence length of BERT limits our ability on how much information we could encode into the final embeddings. Nevertheless, the sequence length might not be long enough to capture all aspects of a drug or patient, hurting performance. There are multiple approaches of increasing the sequence length of Transformer based models like the Longformer by Beltagy et al. ((2020)), or Reformer by Kitaev et al. ((2020)).

6.2.6 Drug Linking

Linking drug names to Wikipedia titles using trigram matching is most likely not the approach with the best quality results. We were also not able to link all drugs also including some of the most used drugs. Wrongly linked drugs introduce noise which might hurt performance. The dataset we used does also not include redirects, and we do not handle disambiguation pages properly. Using approaches that address drugs, which are only named using abbreviations better, is especially interesting. It is also possible to use full-text search, the Wikipedia search API, or a manually created matching to link drug names to Wikipedia articles.

Bibliography

D. Baptista, P. G. Ferreira, and M. Rocha. Deep learning for drug response prediction in cancer. *Briefings in Bioinformatics*, 22(1):360–379, Jan. 2020. doi: 10.1093/bib/bbz171. URL https://doi.org/10.1093/bib/bbz171.

M. Bayer. Sqlalchemy. In A. Brown and G. Wilson, editors, *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*. aosabook.org, 2012. URL http://aosabook.org/en/sqlalchemy.html.

I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020.

E. Choi, M. T. Bahadori, A. Schuetz, W. F. Stewart, and J. Sun. Doctor ai: Predicting clinical events via recurrent neural networks. In F. Doshi-Velez, J. Fackler, D. Kale, B. Wallace, and J. Wiens, editors, *Proceedings of the 1st Machine Learning for Healthcare Conference*, volume 56 of *Proceedings of Machine Learning Research*, pages 301–318, Northeastern University, Boston, MA, USA, 18–19 Aug 2016. PMLR. URL https://proceedings.mlr.press/v56/Choi16.html.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1423. URL https://aclanthology.org/N19-1423.

I. Deznabi, M. Iyyer, and M. Fiterau. Predicting in-hospital mortality by combining clinical notes with time-series data. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4026–4031, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.352. URL https://aclanthology.org/2021.findings-acl.352.

S. Eger, P. Youssef, and I. Gurevych. Is it time to swish? comparing deep learning activation functions across nlp tasks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4415–4424, Brussels, Belgium, Oct.-Nov. 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1472. URL https://aclanthology.org/D18-1472.

e. a. Falcon, WA. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

S. N. Golmaei and X. Luo. DeepNote-GNN. In *Proceedings of the 12th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*. ACM, Aug. 2021. doi: 10.1145/3459930.3469547. URL https://doi.org/10.1145/3459930.3469547.

A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014. URL http://arxiv.org/abs/1410.5401.

M. Hashir and R. Sawhney. Towards unstructured mortality prediction with free-text clinical notes. *Journal of Biomedical Informatics*, 108:103489, Aug. 2020. doi: 10.1016/j.jbi.2020.103489. URL https://www.sciencedirect.com/science/article/pii/S1532046420301179.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, June 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/cvpr.2016.90.

S. Hu and F. Teng. An explainable cnn approach for medical codes prediction from clinical text. *ArXiv*, abs/2101.11430, 2021.

K. Huang, J. Altosaar, and R. Ranganath. Clinicalbert: Modeling clinical notes and predicting hospital readmission. *ArXiv*, abs/1904.05342, 2019.

S. Humeau, K. Shuster, M. Lachaux, and J. Weston. Poly-encoders: Architectures and pre-training strategies for fast and accurate multi-sentence scoring. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SkxgnnNFvH.

G. Jang, T. Lee, S. Hwang, C. Park, J. Ahn, S. Seo, Y. Hwang, and Y. Yoon. PISTON: Predicting drug indications and side effects using topic modeling and natural language processing. *Journal of Biomedical Informatics*, 87:96–107, Nov. 2018. doi: 10.1016/j.jbi.2018.09.015. URL https://www.sciencedirect.com/science/article/pii/S1532046418301898.

A. E. Johnson, T. J. Pollard, L. Shen, L. wei H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark. MIMIC-III, a freely accessible critical care database. *Scientific Data*, 3(1), May 2016. doi: 10.1038/sdata.2016.35. URL https://www.nature.com/articles/sdata201635.

Y. Kim, S. Zheng, J. Tang, W. J. Zheng, Z. Li, and X. Jiang. Anticancer drug synergy prediction in understudied tissues using transfer learning. *Journal of the American Medical Informatics Association*, 28(1):42–51, Oct. 2020. doi: 10.1093/jamia/ocaa212. URL https://doi.org/10.1093/jamia/ocaa212.

N. Kitaev, L. Kaiser, and A. Levskaya. Reformer: The efficient transformer. *CoRR*, abs/2001.04451, 2020. URL https://openreview.net/pdf?id=rkgNKkHtvB.

A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, Sep 2019. ISSN 1460-2059. doi: 10.1093/bioinformatics/btz682. URL https://academic.oup.com/bioinformatics/article/36/4/1234/5566506.

R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

S. Liu, T. Li, H. Ding, B. Tang, X. Wang, Q. Chen, J. Yan, and Y. Zhou. A hybrid method of recurrent neural network and graph neural network for next-period prescription prediction. *International Journal of Machine Learning and Cybernetics*, 11(12):2849–2856, June 2020. doi: 10.1007/s13042-020-01155-x. URL https://doi.org/10.1007/s13042-020-01155-x.

D. Misra. Mish: A self regularized non-monotonic neural activation function. *arXiv preprint arXiv:1908.08681*, 2019.

J. Mullenbach, S. Wiegreffe, J. Duke, J. Sun, and J. Eisenstein. Explainable prediction of medical codes from clinical text. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, 2018. doi: 10.18653/v1/n18-1100. URL https://aclanthology.org/N18-1100.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017. URL http://arxiv.org/abs/1710.05941.

R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

J. Song, Y. Wang, S. Tang, Y. Zhang, Z. Chen, Z. Zhang, T. Zhang, and F. Wu. Local–global memory neural network for medication prediction. *IEEE Transactions on Neural Networks and Learning Systems*, 32(4):1723–1736, Apr. 2021. doi: 10.1109/tnnls.2020.2989364. URL https://doi.org/10.1109/tnnls.2020.2989364.

B. van Aken, J.-M. Papaioannou, M. Mayrdorfer, K. Budde, F. A. Gers, and A. Löser. Clinical outcome prediction from admission notes using self-supervised

knowledge integration. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pages 881–893. Association for Computational Linguistics, 2021. URL https://www.aclweb.org/anthology/2021.eacl-main.75/.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.

T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

G. Wu, J. Liu, and X. Yue. Prediction of drug-disease associations based on ensemble meta paths and singular value decomposition. *BMC Bioinformatics*, 20(S3), Mar. 2019. doi: 10.1186/s12859-019-2644-5. URL https://doi.org/10.1186/s12859-019-2644-5.

D. Zhang, C. Yin, J. Zeng, X. Yuan, and P. Zhang. Combining structured and unstructured data for predictive models: a deep learning approach. *BMC Medical Informatics and Decision Making*, 20(1), Oct. 2020. doi: 10.1186/s12911-020-01297-6. URL https://doi.org/10.1186/s12911-020-01297-6.