Using transformers for page count prediction based on bibliographic metadata

University of Turku
Department of Computing
Master of Science (Tech) Thesis
Data Analytics
April 2025
Teo Kekäläinen

Supervisors:
Prof. Leo Lahti
Ph.D. Ville Laitinen

UNIVERSITY OF TURKU
Department of Computing

Teo Kekäläinen: Using transformers for page count prediction based on bibliographic metadata

Master of Science (Tech) Thesis, 52 p.
Data Analytics
April 2025

Bibliographic data science is a field of digital humanities which aims to enable the usage of bibliographic data for quantitative research. Due to inconsistencies, bibliographic data needs to be harmonized before it can be used for research. A part of the harmonization process is estimating the page count of documents based on short text descriptions which contain numbers, words and abbreviations. This thesis proposes a new approach for page count estimation which takes advantage of natural language processing to convert the text descriptions into vector form by using a pre-trained encoder-only transformer model. The vectors are then used to predict the page count by using an artificial neural network which is attached to the encoder-only model.

Two experiments were done in this thesis by training two machine learning models. First, a model was fine-tuned for page count prediction using a harmonized subset of the Finnish national bibliography, Fennica, which contained both the page count descriptions and numerical page counts. The second experiment was to use another harmonized subset of Fennica, to fine-tune the encoder-only part of the model using the masked language modeling task, which was done by using only the page count descriptions from the second dataset. After masked language modeling fine-tuning, the whole model consisting of the encoder-only model and the attached artificial neural network, was fine-tuned for page count prediction using the first dataset.

Both models were able to predict the page count of documents but had worse accuracy when predicting high page counts. The model that was first fine-tuned using masked language modeling performed better than the model that was only fine-tuned for page count prediction.

The experiments show that encoder-only models are able to predict the page count of documents, and that masked language modeling can be used to improve page count prediction performance.

Keywords: bibliographic data science, NLP, machine learning, regression

TURUN YLIOPISTO
Tietotekniikan laitos

Teo Kekäläinen: Using transformers for page count prediction based on bibliographic metadata

Diplomityö, 52 s.
Data-Analytiikka
Huhtikuu 2025

Bibliografinen datatiede on digitaalisten ihmistieteiden ala, jonka tavoitteena on mahdollistaa bibliografisen datan käyttö kvantitatiiviseen tutkimukseen. Bibliografisen datan hyödyntäminen tutkimuksessa vaatii datan harmonisointia datan sisältämien epäjohdonmukaisuuksien vuoksi. Dokumenttien sivumäärän arviointi lyhyiden tekstikuvausten pohjalta on osa harmonisointiprosessia. Tämä opinnäytetyö ehdottaa luonnollisen kielen käsittelyä hyödyntävää lähestymistapaa, jossa transformer-arkkitehtuuriin perustuvaa esikoulutettua encoder-mallia käytetään sivumääräkuvausten muuntamiseen vektorimuotoon. Tämän jälkeen dokumenttien sivumäärää ennustetaan vektoreiden pohjalta hyödyntämällä encoder-malliin liitettyä keinotekoista neuroverkkoa.
Työssä tehtiin kaksi koetta kouluttamalla kaksi koneoppimismallia. Ensimmäinen malli hienosäädettiin sivumäärän ennustamiseen käyttämällä Suomen kansallisesta bibliografiasta, Fennicasta, johdettua tietoaineistoa. Tietoaineisto sisälsi sekä tekstimuodossa olevat sivumääräkuvaukset että aiemmin arvioidut sivumäärät, joten ainestoa pystyttiin käyttämään ohjattuun oppimiseen. Toisessa kokeessa käytetiin toista Fennicasta otettua tietoaineistoa hienosäätämällä encoder-mallia sivumääräkuvausten pohjalta käyttämällä masked language modeling -tehtävää. Masked language modeling -hienosäädön jälkeen toinen malli hienosäädettiin sivumäärän ennustamiseen käyttämällä ensimmäistä tietoainestoa.
Molemmat mallit pystyivät ennustamaan dokumenttien sivumäärää, mutta niiden ennustuskyky heikkeni suurilla sivumäärillä. Masked language modeling -tehtävään hienosäädetty malli oli parempi sivumäärän ennustamisessa, kuin pelkkään sivumäärän ennustamiseen hienosäädetty malli.
Kokeet osoittavat, että encoder-malleilla pystytään ennustamaan dokumenttien sivumäärää ja, että masked language modeling -hienosäätö pystyy parantamaan sivumäärän ennustustarkkuutta.

Asiasanat: bibliografinen data tiede, NLP, koneoppiminen, regressio

Contents

List of Figures

List of Tables

1 Introduction

Bibliographic data are structured information that describes documents [1]. The information can describe the form, context or content of the documents, e.g. the type, language or title of a document. The documents can be in any form or medium. In this thesis, the focus will be on bibliographic data that describes printed documents such as books, maps or continuous publications.

Bibliographic data science [2] is an emerging field of digital humanities. It aims to enable the use of bibliographic data for quantitative analysis. It is focused on computational methods of harmonizing bibliographic data, and of integrating bibliographic data from multiple sources. The harmonization is done by, for example, removing spelling errors and standardizing terms. It is also possible to derive new features from the existing features such as the estimated amount of paper consumed by printing each document [3] which can be derived from a document's page count or physical dimensions. From a methodological perspective, it is also important that the methods used are scalable because collections of bibliographic data, bibliographies, can contain millions of entries.

Generally, most research on historical literature has focused on studying full texts, but bibliographic data can also be used as a research object. One advantage of bibliographic data is that it is often more structured, standardized and smaller in terms of size than full texts. It can also be used in combination with full text collections to provide additional context. [2] Bibliographic data has been used to

study e.g. the development of different book printing formats [2], the relation between book prices and demand in eighteenth-century Britain [4], and the history of book-printing in Finland and Sweden [3].

Using bibliographic data for research isn't simple, however. It requires both understanding the context of the data and dealing with issues in data quality and consistency. When bibliographic data were originally catalogued, the goal was to maintain as much information as possible about the original document, which is why inconsistencies such as misspellings were not fixed. Bibliographic data is often entered manually into databases which can lead to further inconsistencies [5]. Bibliographies also generally consist of data that has been catalogued over a long period, which can lead to differences in the standards used for the entries. Finally, the reasons why the data were originally collected can vary, which might lead to biases in the data. All of these aforementioned things need to be considered when using bibliographic data as a basis for quantitative research. [2]

The Finnish national bibliography, Fennica [6], is a bibliographic database containing information on documents published in Finland. The earliest documents described by the data were published in 1488. As of the time of writing this thesis, the database has over 1.2 million entries. The first Finnish national bibliography, which described literature published between the years 1544-1877, was made in the 1870s. Later the bibliography was extended to include entries up to the 1900s. Converting the national bibliography to electronic form began in 1978. Nowadays, almost all the printed bibliographies and index cards have been transferred to the Fennica dataset. The two datasets used in this thesis are harmonized subsets [7] of the Fennica dataset provided by the Turku Data Science Group.

The Fennica dataset uses MARC21 formats, which are maintained by the United States Library of Congress. MARC21 formats are standards for the representation and communication of bibliographic data and related information in machine-

readable form [8]. The formats exist for five types of data: bibliographic, holdings, authority, classification, and community information. The MARC21 format for bibliographic data specifies the format for describing, retrieving, and controlling bibliographic materials. There are different bibliographic data specifications for books, serials, computer files, maps, music, visual materials and mixed materials.

A MARC record has three sections: the leader, the directory and the variable fields. The leader defines the parameters for processing the record. The directory describes the tag, starting location and length of each field in the record. Finally, the data content of the record is in the variable fields which can be split into variable control fields and variable data fields. Each field is identified by a three-character tag. Fields can be grouped based on the first character of the tag which represents the function of the data contained within the fields, for example, the fields starting with 3 contain the physical description of documents in the bibliographic MARC specification. The remaining two characters of the tag describe the type of information stored in the field. The information in the variable data fields is stored in coded subfields. For example, the main variable field this thesis is concerned with is the field 300a, meaning the subfield *a* of field 300. [8]

1.1 Research objectives

The MARC21 300a field contains a short text description of the length of a document. The text description can be just a single number like "54", but it can also contain words or page ranges e.g. "5 pages, 5-45, 3 images". The entries can also be written in different languages. In the Fennica dataset, most entries are written in Finnish, but there are also entries written in other languages such as Swedish and English. The values also often use abbreviations such as "p." or "s." to describe the page count in English and Finnish respectively. Finally, the entries can contain Roman numerals which can appear alongside Arabic numerals in a single entry.

To be able to use the page count for quantitative analysis, the descriptions from the 300a field need to be mapped to numbers, which are estimates of the page count of the document. This has been done before [3] by parsing different values from the entries, then using them to calculate a page count estimate. The problem with this approach, however, is that it requires a lot of manual effort because rules such as regular expressions need to be defined to parse the values from the 300a field. The different languages, abbreviations and other inconsistencies make this a difficult task because the values need to either be harmonized or the rules for parsing them need to be robust enough to take every form of a term into account. Also, when new bibliographic data from another source is added, it can often require writing new rules since the new data might have different standards or entries written in new languages.

Instead of manually parsing a set of features from text, it is possible to use natural language processing (NLP) techniques, which are covered in section 2.3, to produce a numerical, vector representation of the text. One way of doing this is using encoder-only transformer [9] models, which are covered in sections 2.5 and 2.6. These models could allow for skipping the manual parsing process by producing a vector representation of each MARC21 300a entry, which could then be used as an input to an artificial neural network (section 2.2) to predict the page count of each document. This approach would take advantage of machine learning (ML) to learn the relations between the input and output automatically instead of using a manually defined function. There are data that includes both the MARC21 300a entries and the previous page count estimates, which can be used for training the artificial neural network using a supervised learning approach.

There are also data which have the MARC 300a field, but where the page count estimate is incorrect due to a problem with the harmonization function. Using these data to fine-tune the encoder-only model using the masked language modeling

(MLM) task, covered in section 2.6.1, could help the model produce more accurate representations of the MARC21 300a entries thus improving the page count prediction performance.

The research questions of this thesis can be formulated as follows:

RQ1 Can a transformer encoder model be used to predict the page count of a document based on the value of the MARC21 300a field?

RQ2 Can page count prediction performance be improved by fine-tuning the model for the masked language modeling task using unannotated data?

This approach would be more scalable than the previous approach, since it could easily be applied on large datasets without as much manual effort such as writing regular expressions or standardizing the data. Moreover, there are multilingual models, which can be used to generate the vector representations of the entries. Using multilingual models could help with processing data that is written in multiple languages.

1.2 Thesis structure

Chapter 2 is an introduction to the basics of machine learning and natural language processing. The goal of the chapter is to provide enough background information so that the process of training a transformer model can be understood. This includes explanations of the basic NLP data pre-processing steps needed, and an explanation of artificial neural networks. The chapter also explains the attention mechanism, which is a key component of the transformer models, before explaining the original transformer architecture and the BERT models used in this thesis.

Chapter 3 is an introduction to the datasets used in this thesis. The chapter describes the size and features of the dataset, and the processing done to the data.

The data processing includes filtering the data and cleaning the MARC21 300a entries.

Chapter 4 is dedicated to the training of the first model without MLM fine-tuning. This model will be referred to as the pure regression model since it does not take advantage of MLM fine-tuning. The model is trained for the regression task with the page count being the target variable while using the MARC21 300a field to make predictions. The chapter also explains the reasoning behind model selection and a description of the model's architecture. The chapter ends by visualizing and describing the results on the validation set of the regression dataset.

Chapter 5 describes the process of fine-tuning the second model, which will be referred to as the MLM fine-tuned model, for both MLM task and regression task. The chapter also explains the reasoning behind MLM fine-tuning and contains the results obtained on the validation set of the regression dataset.

Chapter 6 concludes the thesis. The chapter starts by comparing and analyzing the performance of the two models on the test set of the regression dataset. Then, the chapter answers the research questions, addresses the limitations of the results, and proposes directions for further research. Finally, the chapter ends by summarizing the thesis.

2 Machine learning and natural language processing

This chapter introduces the relevant concepts in machine learning (ML) and natural language processing (NLP) that are needed to understand the process of training and using a transformer model for regression. The chapter also explains artificial neural networks and their training process, the attention mechanism which is used by the transformer models, the original full encoder-decoder transformer architecture, and the encoder-only BERT model, a variant of which is used in this thesis.

2.1 Machine learning

Machine learning (ML) is a branch of artificial intelligence (AI). An often used definition for machine learning is the one by Mitchell [10]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." The advantage of ML is that it allows for solving tasks that are too complicated to be solved manually i.e. tasks where it would be impractical or even impossible to manually define rules for all possible cases, such as the page count prediction done in this thesis. [11, p. 99]

ML algorithms, which are also called models, learn by processing examples that consist of a set of features. For example, in the datasets used in this thesis, each

example is a set of bibliographic metadata which describes a single document. This set includes features such as the title of the document, the year the document was published and the primary language of the document. Each example can be represented as a vector of set length where each entry is a feature. These vectors can then be collected into a so-called design matrix so that each row of the matrix is an example, and each column is a feature. [11, p. 98-99, 106]

Supervised learning is a branch of ML, where the goal is to predict the value of some target variable based on the available features. In supervised learning, the desired values of the target are available and used for training the model, i.e. there is a ground truth that the predictions of the algorithm can be compared to. [11, p. 105] Mathematically, a supervised learning algorithm can be represented as a function $f(X_i) = \hat{Y}_i, \hat{Y}_i \sim Y_i$, that takes the vector of features X_i as input and produces an estimate $\hat{Y}_i$ of the real target value Y_i for each example i in the dataset.

Supervised learning requires annotated data which means that the data needs to include the value of the target variable, i.e. the data is a set of input-output pairs where the input is the set of features passed to the algorithm and the goal of the algorithm is to produce the correct output. Annotation is typically done manually which can be a time-consuming process, therefore there are fewer annotated than unannotated data available [12].

The two most common supervised learning tasks are classification and regression. In classification, the goal is to predict the value of some discrete variable i.e. assign each example a label that belongs to a finite set of labels [11, p. 100]. For example, learning to detect spam email is a binary classification task, meaning that each email is assigned one of two labels: "spam" or "not spam". It is also possible for an example to be given multiple labels, this type of classification is called multi-label classification. A common way to evaluate classification performance is measuring the accuracy of the model, which is simply the proportion of examples where the

model predicted the correct label. Alternatively, the error rate of the model, which is the proportion of predictions that are incorrect, can also be used. [11, p. 103-104]

Regression means predicting a numerical value based on the input features [11, p. 101]. The page count prediction done in this thesis is an example of a regression task. Common performance metrics, that will also be used in this thesis for measuring regression performance, are mean squared error (MSE, equation 2.1), and mean absolute error (MAE, equation 2.2). Defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.1}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}(|Y_i - \hat{Y}_i|) \tag{2.2}$$

Where Y_i is the actual value of the target variable and $\hat{Y}_i$ is the value predicted by the ML model for each example i. For these metrics, a good model would get a value that is as close to zero as possible since the metrics measure the error rate in the predictions.

Unlike supervised learning, unsupervised learning does not require annotated data. Instead of comparing the model's output to a ground truth, unsupervised learning methods aim to learn useful properties about the structure of the dataset without having the ground truth available. Clustering is an example of an unsupervised learning task, where the goal is to form some predefined number of clusters out of similar examples. [11, p. 146-150]

Self-supervised learning is a type of unsupervised learning where labels are generated automatically from unannotated data. This can mean, for example, predicting some part of the input based on other parts of the input or corrupting a part of the input, then predicting the value of the corrupted part. The masked language modeling (MLM) [13] task used in this thesis is a self-supervised learning task, since it consists of predicting the value of a masked word based on the other words in a

text sequence. [12]

Generally, in ML, the goal is not to obtain the best performance on the data used to train the model, but to obtain the best performance on new data that the model has not seen before. A model's ability to perform well on unseen data is known as generalization. Generalization can be measured by splitting the dataset into a training set and a test set. The training set is then used to train the model and the test set is used to estimate the model's generalization after training. It is very important that the model does not see any of the test set examples during training because then the performance on the test set does not reflect generalization anymore. [11, p. 110]

The capacity of a ML model, meaning its ability to fit different functions, needs to be adjusted based on the data and task. If a model's capacity is too high, it will perform far better on the training set than the test set, which is known as overfitting. If a model's capacity is not high enough, however, it will underfit meaning that the model is not able to perform well enough on the training set [11, p. 110-114]. If a model's capacity is optimal, it will obtain good performance on both training and test set.

One way to improve a model's generalization is to use regularization. Regularization means any modifications made to a model that aim to reduce the generalization error without reducing the training error. [11, p. 120]

Hyperparameters are settings that control a ML model's behaviour. The hyperparameters are not changed by the ML algorithm itself during training, instead they are set before training. Many hyperparameters, such as the amount of layers in an artificial neural network, affect the model's capacity. These hyperparameters can not be learned by the ML algorithm during training since that would always lead to selecting values that result in the model having maximum capacity, and the model overfitting the training data. Hyperparameters are often optimized by

splitting another subset from the training data which is called the validation set. Hyperparameter optimization can then be done by training a model with different hyperparameters on the training set, and then evaluating the model's performance on the validation set to select the hyperparameters that obtained the best performance. Once optimal hyperparameters have been selected the generalization performance can be evaluated on the test set. [11, p. 120-121]

2.2 Artificial neural networks

An artificial neural network (ANN) is an ML model made of interconnected layers of processing units which are called artificial neurons. The connections between the layers are weighed and the model learns by adjusting the values of these weights during training [14, p. 5]. The simplest ANN consists of an input layer and an output layer.

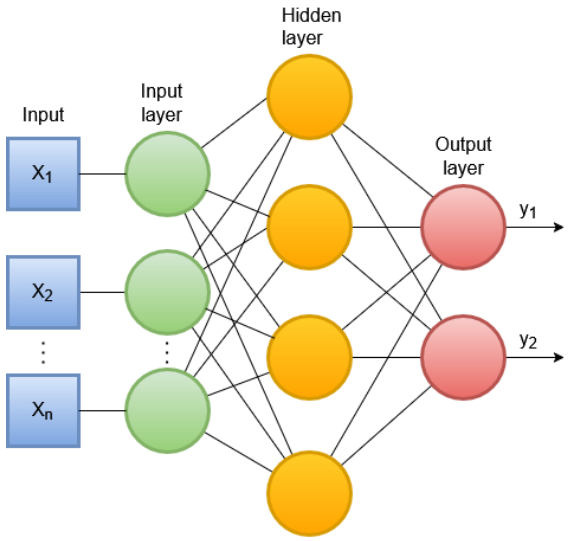

Figure 2.1: ANN with one hidden layer, n inputs and two outputs

ANNs also often have one or more hidden layers between the input and output layers. The amount of hidden layers and neurons per layer are hyperparameters that are adjusted based on the nature and complexity of the target task. Feedforward

neural networks are ANNs where the data passes only in one direction: from the first layer to the last layer. [14, s. 21-23] Figure 2.1 shows a fully-connected feedforward neural network with a single hidden layer and two outputs. Fully-connected means that each artificial neuron of a layer is connected to every neuron of the subsequent layer.

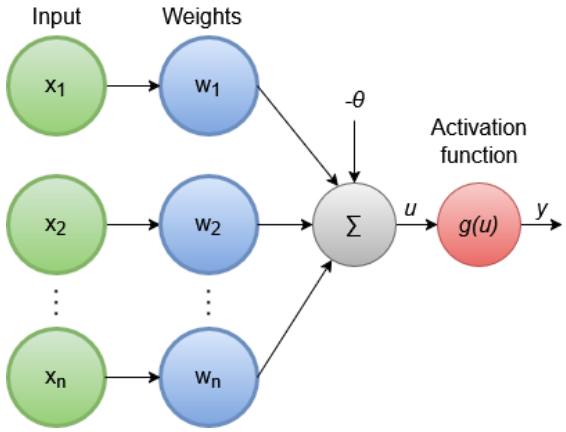

Figure 2.2: Artificial neuron with n inputs and n weights

The original artificial neuron is the McCulloch and Pitts neuron which was introduced in 1943. It is a mathematical function that takes in some amount of inputs $x_1, ..., x_n$ and produces a single output y. Each input has a weight w_i associated with it. The inputs are multiplied by the weights and summed. An activation threshold or bias θ is then subtracted from the sum of the weighted inputs. Generally, the activation threshold can be seen as the limit which the weighed sum of inputs needs to reach for the neuron to be activated and produce an output. The sum of weighted inputs and activation threshold is known as the activation potential u :

$$u = \sum_{i=1}^{n} w_i \cdot x_i - \theta \tag{2.3}$$

. [14, p. 11-13]

The activation potential is passed on to the activation function g, which generally limits the value to be in some range, and the result y is the output of the neuron

[14, p. 12]. Figure 2.2 shows an artificial neuron with n inputs and n weights.

The input layer of an ANN is responsible for receiving the input data. The data is also often normalized or standardized to help with mathematical precision. Most of the processing in an ANN occurs in the hidden layers where patterns associated with the data and the target task are extracted. Finally, the output layer is responsible for producing the final output of the network. The size of the output layer is adjusted based on the task. For example, when using an ANN for regression, the output layer only has a single neuron and the output of the neuron is the model's prediction. For classification tasks, the number of neurons in the output layer is equal to the amount of target labels. [14, p. 21-23]

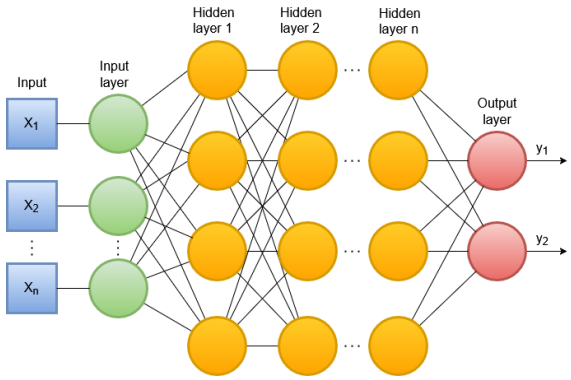

Figure 2.3: A deep neural network with n hidden layers and two outputs

2.2.1 Deep neural networks

A deep neural network (DNN) is an ANN with multiple hidden layers. Representation learning methods, such as deep-learning (DL) using DNNs, are used to automatically learn representations from raw data (see figure 2.3 for a schematic representation). DL builds multiple levels of representations from the raw data with each level having a slightly higher abstraction level. The main advantage of deep learning is that it reduces the amount of manual feature engineering [15].

A convolutional neural network (CNN) is a type of DNN that has at least one

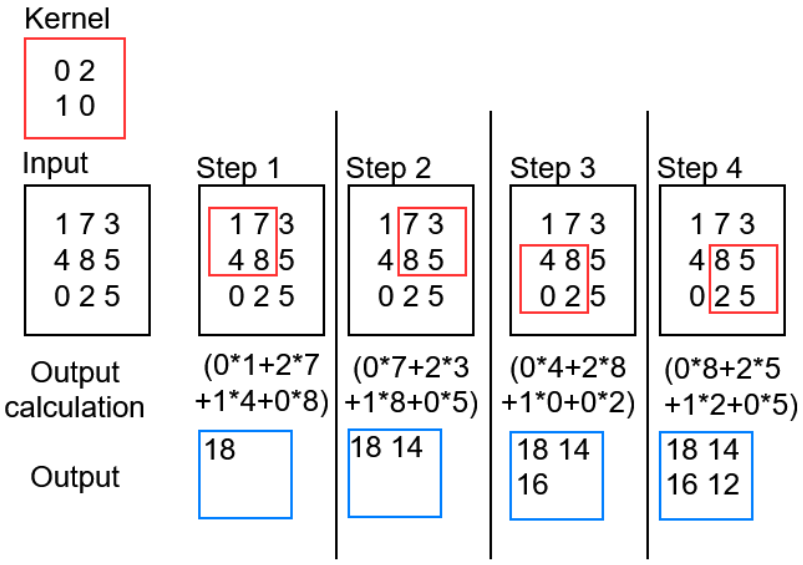

Figure 2.4: Convolution operation done on a 2D input

layer which uses the convolution operation [11, p. 330]. The convolution operation works by moving a convolution kernel, which consists of weights that are learned during training, over the input. Figure 2.4 shows how the convolution operation is applied on a 3x3 input to produce a 2x2 output. During each step of the operation, the element-wise product of a part of the input and the weights of the kernel is computed, then the values of the element-wise product are summed to get the output for each step. CNNs are generally used on input data that is in n-dimensional form. For example, images can be represented as a 3D array that consists of the RGB values of each pixel in the 2D image. The details of CNNs are outside of the scope of this thesis, for more information on CNNs, see chapter 9 of the Deep Learning textbook by Goodfellow *et al.* [11].

Recurrent neural networks (RNN) are another type of DNN. RNNs are generally used for processing sequential data such as text, which can be seen as a sequence of words. RNNs work by processing the input one element at a time while also maintaining a hidden state, that contains information about the elements that were processed earlier. The hidden state is updated at each step and given as an additional

input to the next step of the model. The final hidden state of an RNN can be used as a representation of the entire input sequence, although RNNs can have difficulties with longer inputs, since the impact of earlier elements decreases as the distance between the first elements and the current element increases. [15] More advanced RNN architectures such as the long short-term memory (LSTM) architecture [16] are better at processing longer inputs. For this thesis, the details of RNN architectures are not relevant. For more information on RNNs, see chapter 10 of the Deep Learning textbook by Goodfellow *et al.* [11].

2.2.2 ANN training

Although types of ANNs, such as the self-organizing maps [17], can be used for unsupervised learning, this thesis focuses on using ANNs for supervised learning. ANNs are trained for supervised learning by adjusting the network's parameters, i.e. the weights and biases, to optimize the value of some objective function on the training set. [15] The objective function is either minimized or maximized, depending on the function. For example, when training an ANN for regression, the model's parameters are adjusted so that the value of the MSE loss is minimized.

The parameter adjustment works by calculating the gradient of the cost function with respect to the model parameters. If the goal is to minimize the objective function, e.g. when using MSE, then the cost function is the same as the objective function. However, if the goal is to maximize the objective function, such as classification accuracy, then the cost function is the negative of the objective function. The gradient of the cost function is calculated by using a method called back-propagation [18], which works by calculating the gradient starting from the outputs of the model and then working backwards through the layers. [15]

After calculating the gradient, model parameters are adjusted in the opposite direction of the gradient, since that direction is where the cost function's value

decreases the most. The size of the adjustment is determined by a hyperparameter called learning rate which is a positive real number. The learning rate can be static or adjusted during training depending on the optimization method. [11, p.85-86]

The simple optimization method of adjusting the parameters based on the gradient of the cost function is called gradient descent. Usually, instead of calculating the gradient of all outputs at once, the training is done in batches which is known as stochastic gradient descent. [15] More advanced optimization methods, such as the Adam optimizer [19], which involve additional hyperparameters beyond just the learning rate, are also used for training ANNs.

Two common methods of regularization for ANNs, that are also used in this thesis, are dropout [20] and weight decay. Dropout works by temporarily removing some artificial neurons and all of their connections from the network during training. The probability for neurons to be removed is a hyperparameter. Weight decay means that the model will prefer weights with smaller values. The amount of preference for smaller weights is determined by another hyperparameter. [11, p. 119]

2.3 Natural language processing

Hirschberg and Manning [21] define natural language processing (NLP) as "the subfield of computer science concerned with using computational techniques to learn, understand, and produce human language content". Common NLP tasks include e.g. machine translation, part-of-speech (POS) tagging and text classification. Machine translation is an example of a sequence-to-sequence task where both the input and output of a model is a sequence of text. POS tagging means labeling each word in a sequence based on the type of word such as verb, adjective or noun. POS tagging is an example of a sequence labeling task where each part of a sequence is given its own label. Finally, text classification refers to classifying a text sequence e.g. a sentence. Text classification is an example of a sequence level task whereas POS

tagging is an example of a token level task. The page count prediction task done in this thesis is an example of text regression which is a sequence level task.

Tokenization is a necessary data pre-processing step in NLP. It is the process of splitting a sequence of text into a sequence of typographic units that are called tokens. Tokens are often just words, but words can also be split into multiple tokens, which is called subword tokenization. Tokenization can be done in many ways, starting with simply splitting text based on whitespace. A more advanced approach is to use a dictionary to match character sequences as tokens. Dictionary-based tokenization methods can struggle with out-of-dictionary tokens, since the size of the dictionary is limited. [22, p. 185]

One of the key challenges in NLP is finding a way to represent text in a numerical format so that it can be used as the input for a ML model. The simplest way to represent a token is to represent it as a one-hot vector, meaning a vector where a single value is one and every other value is zero. These vectors are N-dimensional, where N is the size of the dictionary, and the non-zero element is the element i, where i is the index of the token in the dictionary. The problem with this approach, however, is that the resulting vectors are very sparse which makes computation inefficient. Also, the representations cannot be used to measure the similarity of tokens and they do not take context into account. [23]

Word embeddings, which are dense vectors of real numbers, are a more sophisticated method to represent the meaning of words or tokens. The advantage of word embeddings is that they are more efficient computationally due to their density, and that they can be used to compare the semantic similarity of words. In theory, words that have a similar meaning should have embeddings that are close to each other. [23]

Word embeddings can be categorized into static and contextual embeddings. Static embedding methods, such as CBOW [24] and GloVE [25], assign each word a

single representation that is not changed when training a model for a downstream task such as text classification. More recent methods, such as BERT [13] and ELMO [26], however, can produce contextual embeddings, which are adjusted during training to better reflect the context of the training data. [23]

2.4 Attention

Attention is a mechanism which allows ML models to learn what parts of the input are relevant during training. It is based on assigning weights to different parts of the input that show how relevant the parts of the input are for the target task. Originally, a mechanism similar to attention was proposed for computer vision by Hinton and Rochelle in 2010 [27], and the term attention was popularized by Mnih *et al.* in 2015 [28]. However, in this thesis the focus is on using attention for NLP tasks. In NLP, attention was originally used for machine translation by Bahdanau *et al.* in 2015 [29], but since then it has been used for various other tasks [30]. Attention is the key mechanism behind the transformer models used in this thesis [9].

Attention works by mapping a sequence of key vectors K to a distribution of weights. The keys represent the input sequence in some way such as word embeddings. It is also common to use a query element q, that defines how input elements should be emphasized. A vector e of energy scores is calculated from the keys and query by using a compatibility function f. Each energy score e_i in e represents the relevance of a key k_i in K.

$$e = f(q, K)$$

The energy scores are transformed to a distribution of attention weights a by passing them to a distribution function g, $a = g(e)$. The weights produced by the attention mechanism reflect the relevance of each element in the input to the target task, with

respect to the query and keys.

Many models also use an additional input sequence for computing attention. This sequence is known as the values V, which can be seen as another representation of the input data represented by the keys, with each element of V corresponding to only one element of K. The values can be combined with the attention weights a to produce a set of weighed representations of the values Z. These weighed representations can then be merged to produce a compact representation of the input known as the context vector c. [30]

The compactness of the context vector makes it easier to represent longer sequential inputs, which was one of the major downsides of using RNN hidden states to represent inputs, because the context vector will focus on the relevant elements of the input instead of assigning the same importance to all the elements in an input sequence. Since the context vector condenses information, it also takes fewer computational resources to process it compared to the original representation such as a sequence of word embeddings. In this sense, attention can also be seen as a way to condense inputs into a compact form. [30]

Self-attention means computing attention based on only the input sequence, meaning that both the query and the keys are taken from the same sequence. The most common approach to self-attention works by applying multiple steps of attention to an input sequence. On each step, a different element of the input sequence is used as the query. This approach leads to contextual embeddings, that reflect the relevance of each element in the input to all the other elements of the input. Self-attention in particular helps with the problem of representing longer inputs, since it allows each element of the input to affect the representations produced by the attention mechanism. [30]

Multi-head attention refers to using multiple attention functions which are computed in parallel on the same input with each attention head having its own set

of learnable weights for projecting the keys, queries and values. The context vectors produced by the different attention heads are merged together to produce the final representation at the end of each step. Multi-head attention is particularly suitable for ambiguous data e.g. for data where words can have multiple meanings because multi-head attention allows the representations produced by the attention mechanism to combine information from different interpretations of the input. [30]

2.5 Transformer architecture

The transformer ANN architecture, introduced in 2017 by Vaswani *et al.* [9], is based on using multi-head self-attention without recurrence or convolution, which reduces the amount of sequential computation thus increasing the parallellizability of the model. The increased parallelizability allows more efficient training using large amounts of data. Models based on the transformer architecture have achieved state-of-the-art results in many NLP tasks by taking advantage of a transfer learning approach, where the models are first trained on a large amount of unannotated data using a self-supervised task and then fine-tuned on smaller amounts of annotated, domain-specific data for the desired downstream task such as text classification. [31]

The full transformer architecture consists of two parts: the encoder stack and the decoder stack, which are shown in figure 2.5. The encoder and decoder stacks consist of some amount of encoder and decoder blocks, respectively. The number of blocks in each stack varies from model-to-model. In the original transformer architecture, the encoder stack consisted of six identical encoder blocks. The encoder stack produces a sequence of contextual representations of the input tokens and the decoder generates tokens based on the encoder's output and the previously generated tokens. [9]

The encoder blocks contain two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward neural network. Each sub-layer also has a residual connection around it which is followed by layer normaliza-

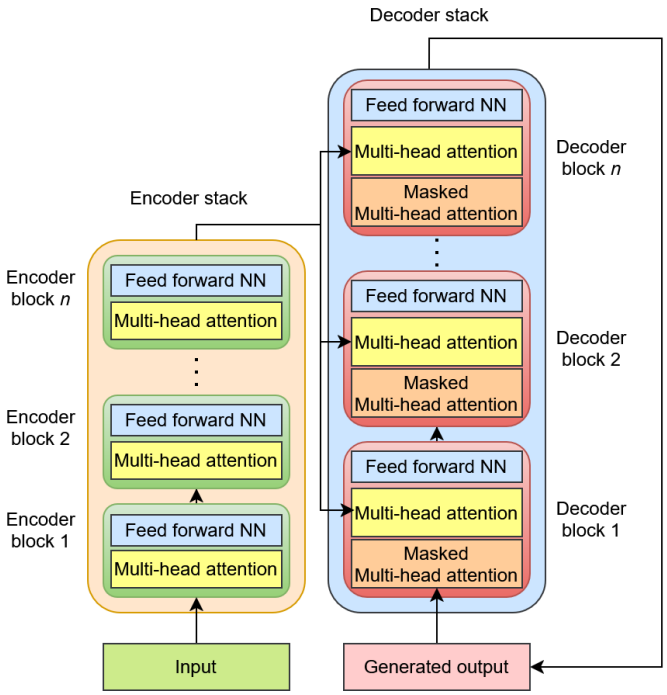

Figure 2.5: Simplified illustration of the transformer architecture inspired by Figure 1 in Vaswani et al. [9] The model consists of an encoder and a decoder stack which have N encoder and decoder blocks. The encoder stack produces a sequence of contextual representations of the input, which the decoder uses alongside the previously generated output to generate more tokens. For simplicity, the residual connections and layer normalization included in the encoder and decoder blocks are not shown.

tion. The residual connection means that the output of each sub-layer is a combination of the input of the sub-layer and the output of the computation done in the sub-layer i.e. the output of the multi-head attention or feedforward network. On the first encoder block the self-attention mechanism is computed over the input of the model and on the other blocks self-attention is calculated based on the output of the previous encoder block. The output of the encoder stack is a sequence of token representations, with each input token having an n-dimensional, contextual vector representation [9]

The decoder stack consists of N identical decoder blocks, which also contain the same self-attention layers, fully-connected layers and residual connections as the encoder blocks. However, the decoder blocks also have an additional attention sub-

layer which performs multi-head attention on the output of the encoder stack i.e. the contextual representations produced by the final encoder block. The self-attention layer of the decoder stack calculates attention over the previously generated output which is masked so that the attention is calculated based only on the earlier tokens in the output sequence. In the original architecture, the output of the decoder stack is a sequence of scores for the next token, these scores are turned into a probability distribution by using the softmax function. The token that is generated can then be sampled from the probability distribution. [9]

The transformer also has two embedding layers that share the same set of weights, one for the encoder stack and one for the decoder stack, that use learned embeddings to transform the output and input tokens to vector representations. Since the transformer uses neither convolution nor recurrence, special positional encodings need to be added to the learned embeddings of the encoder and decoder stacks so that the model can make use of the order of tokens in the input sequence. These positional encodings contain information about the relative or absolute position of tokens in the sequence and have the same dimensionality as the input embeddings so that they can be summed together. [9]

Transformer-based models can be split into three categories based on which parts of the transformer architecture they use: encoder-only, decoder-only and encoder-decoder models. Decoder-only models, such as the Generative Pre-Trained Transformer (GPT) models [32], are generally used for language generation. Decoder-only models are pre-trained using the self-supervised language modeling task, where the goal is to predict the value of a token based on its preceding tokens. Encoder-only models, such as BERT [13], are pre-trained using the MLM task, and they are used for downstream tasks related to language understanding e.g. sequence classification and question answering. Encoder-decoder models, such as T5 [33], contain both parts of the transformer architecture and they are generally used for sequence-to-

sequence downstream tasks, such as machine translation. Encoder-decoder models are pre-trained using denoising tasks where the goal is to reconstruct a text sequence which has been corrupted in some way. [31]

2.6 BERT

BERT (Bidirectional Encoder Representations from Transformers) is an encoder-only transformer model that was introduced in 2019. When BERT was introduced, it achieved state-of-the-art results in eleven NLP tasks. The model was pre-trained using the masked language modeling (MLM) task, an example of which is shown in figure 2.6, and next sentence prediction (NSP) task using two English text corpora containing a total of 3.3 billion words. BERT can be fine-tuned for various downstream tasks by using smaller amounts of annotated data. [13]

BERT uses a form of subword tokenization, called WordPiece [34], with a dictionary length of 30,000. Two special tokens are also added to each input sequence: the [CLS] token and the [SEP] token. The [CLS] token is a special classification token that is added to the start of each input sequence. The final contextual representation of the [CLS] token can be used to represent the meaning of the whole input sequence in sentence-level tasks. The [SEP] token is used separate sentences from one another. [13]

2.6.1 Pre-training

BERT uses bidirectional self-attention which allows the model to pay attention to the whole input sequence limited only by the length of the context window. Using masked language modeling (MLM) for pre-training was the main innovation that allowed using bidirectional self-attention. MLM was required because models trained using the regular LM task couldn't use bidirectional attention since that would allow

Figure 2.6: Simple example of masked language modeling

the model to see the token being predicted, thus trivializing the prediction process. During the MLM pre-training process, 15% of the tokens were masked. Out of the masked tokens, 80% were replaced by the special [MASK] token, 10% were replaced by a random token and 10% were left unchanged. Instead of just using the [MASK] token, the other alterations were made to reduce the difference between pre-training and fine-tuning because the [MASK] tokens are not used in downstream tasks. [13]

The NSP task was used for pre-training because it allows the model to learn the relationship between two sentences, benefiting tasks such as question answering (QA). Each training example for NSP consists of two sentences: sentence A and sentence B. The sentences are chosen so that 50% of the time sentence B is the sentence that follows sentence A in the original text, and 50% of the time sentence B is just a randomly chosen sentence from the corpus. The model then predicts whether sentence B follows sentence A or not. [13]

2.6.2 Downstream tasks

There are two approaches to using BERT for downstream tasks: the feature-based approach and the fine-tuning approach. In the feature-based approach the pre-trained BERT is used to produce contextual representations of the input without doing any fine-tuning. These representations are then used as the input for another model. [13]

In the fine-tuning approach BERT is initialized with the parameters from pre-training. Then a fully connected feed-forward network is attached to the top of the BERT architecture. This attached network is generally called a head, e.g. classification head in the case of classification. The outputs of BERT are pooled and then

fed to this network. The outputs and inputs of this network are adjusted based on the task. For example, in text classification the network takes the contextual representation of the [CLS] token as input and has an output for each target label. Each output produces a real number called a logit. To get the class prediction, the softmax function is used over these logits to turn them into a probability distribution which shows the probability of each label. In token-level tasks, the input representations of all tokens are used as the input of the attached network. The objective function used during fine-tuning is also selected based on the downstream task. During training, the parameters of both the BERT model and the head ANN are adjusted, although some parameters can also be frozen depending on the approach. [13]

2.6.3 BERT variants

There are two variants of the original BERT model: large and base. The large version contains 24 encoder-blocks (also called transformer blocks) whereas the base version contains 12 encoder-blocks. The base model has a total of 110 million parameters and 12 attention heads, and the large version has 340 million parameters with 16 attention heads. [13]

The BERT team also released a multilingual version of BERT that supports 104 languages. The multilingual version was trained on each language's Wikipedia. The supported languages were chosen based on the size of their Wikipedias. [35] There are also monolingual BERT-based models for languages other than English such as FinBERT for Finnish [36], CamemBERT for French [37] and BETO for Spanish [38]. These monolingual models have been trained on text written in their target languages.

DistilBERT is a smaller version of BERT that was trained using knowledge distillation in which a smaller student model is trained to reproduce the behaviour of a larger teacher model. DistilBERT retained 97% of the performance of the origi-

nal BERT-base model on the General Language Understanding Evaluation (GLUE) benchmark while being 40% smaller in terms of parameters and 60% faster at inference time [39]. This thesis uses the distilled version of the multilingual BERT model.

3 Data inspection and processing

In this chapter, the datasets used in this thesis are examined. The chapter contains descriptions of the two datasets, and an explanation of how the data were processed to create the final datasets: the regression dataset and the MLM dataset.

3.1 Regression dataset

The first dataset is a harmonized subset of the full Fennica dataset described in section **??**. This dataset contains bibliographic metadata describing documents published between the years 1488-1800. In total, the original dataset has 19,152 examples before doing any filtering such as removing null values. The first dataset will be referred to as the regression dataset since it will be used to fine-tune a model for the regression task.

Table 3.1: Some example values of the pagecount and pagecount_orig fields

pagecount_orig	**pagecount**
"[8] s., s. 481-544, [4] s."	76
"[8], 56 s"	64
"4 s"	4
"[26] s."	26
"X, 255, XXII s., [3] karttalehteä (taitettuna) :"	284

The dataset has a total of 34 features such as the title, publication year, language and type of the document. The only two features used for this thesis are "pagecount_orig" and "pagecount". The "pagecount_orig" field contains the value of the

MARC21 300a field, which is a short text description of a document's page count, and the "pagecount" field contains the curated page count estimate for each document, which is an integer. Table 3.1 shows five examples taken from the dataset. It can be seen that the length and complexity of the "pagecount_orig" field varies from example-to-example.

Out of the 19,152 documents described in the unfiltered dataset, 18,393 are books, 733 are maps, 20 are continuous publications and 6 are music. The described documents are written in 18 different languages with Swedish, Latin and Finnish being the most popular having 8,329, 6,976 and 2,958 documents written in each language, respectively. The mean page count of all the documents in the dataset is 34 pages with a standard deviation of 126.

Figure 3.1 shows the distribution of documents by publication decade. The figure shows that most of the documents were published near the end of the 18th century with barely any documents being published before the 17th century. The last decade, 1800, also has far fewer entries because the dataset only contains information of documents released during the first year of that decade.

3.1.1 Data processing

To prepare the final regression dataset, the data was first filtered by only selecting the two relevant features: "pagecount" and "pagecount_orig". Then all the examples that had a missing value for either feature were removed. This removed a total of 159 examples resulting in a dataset with 18,992.

After removing missing values, the duplicate examples with regard to the values of the selected features were removed, i.e. the dataset was filtered so that each pair of values for “pagecount_orig” and “pagecount” appears only once in the dataset. These duplicates aren't necessarily duplicates in the sense that they describe the same documents, but having multiple of the same input-out pairs in the data would

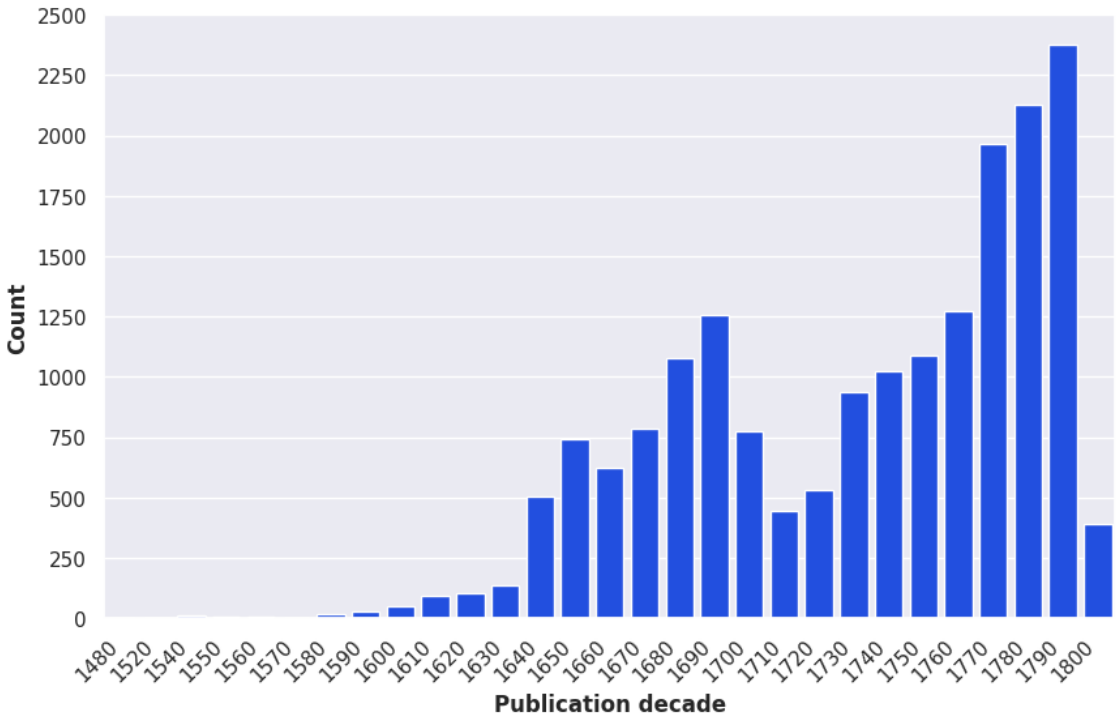

Figure 3.1: Count of documents by publication decade in the regression dataset

skew the results, because then it would be possible to have the same input-out pairs appear in both the test and training set, which would make the results overly optimistic. Removing these duplicates left a total of 3,583 examples out of the original 18,992. Most of the removed examples had very simple "pagecount_orig" values such as "1 p." and low page count values.

One outlier that had a page count of 11,080 was removed from the dataset because it affected the loss during training, since its page count was over 8,000 pages larger than the second largest page count in the dataset. Next, the examples with a page count of 1 were checked because in the second dataset, which will be used for MLM-finetuning, there were a lot of incorrect page count values of 1. The dataset contained eight examples that had a page count of 1. Out of these eight, four were removed because the "pagecount_orig" field described a higher page count e.g. an example with a "pagecount_orig" value of "201 [se on 199] s, 1 s" (in English:

Table 3.2: Comparisons of the original "pagecount_orig" and cleaned "pagecount_orig" fields

Original	**Cleaned**
"[48] s."	"48 s"
"[2] s."	"2 s"
"[4], 16 s."	"4 16 s"
"[26] s."	"26 s"
"[4] s., s. 461-476"	"4 s s 461-476"

"201 [it is 199), 1p.") was removed. Finally, the 42 values that had a page count of 0 were removed since it is not possible to have a printed document with 0 pages.

Afer filtering the data, the next step was to clean the values of the "pagecount_orig" field. This was done by writing a custom function, which first removed all the extra whitespace. Then dots, commas, square brackets, question marks, colons and semicolons were removed. Finally, the strings were converted to lowercase. The values were cleaned to reduce the amount of noise in the data. Table 3.2 shows the difference between the original and cleaned values of the "pagecount_orig" field for a few examples.

After pre-processing, the data was shuffled and then split into training, validation and test sets using an 80-10-10 split. The final regression dataset had a training subset of 2,828 examples, a test subset of 354 examples and a validation subset of 354 examples. The final regression dataset included only two features: "pagecount_orig" and "pagecount".

3.2 MLM dataset

The second dataset is also a harmonized subset of the Fennica dataset. This dataset will be referred to as the MLM dataset. The dataset consists of documents published between the years 1809 and 1917, which means that the dataset has no overlap with the regression dataset since the earliest documents of the MLM dataset were published 9 years after the last documents in the regression dataset. The unfiltered

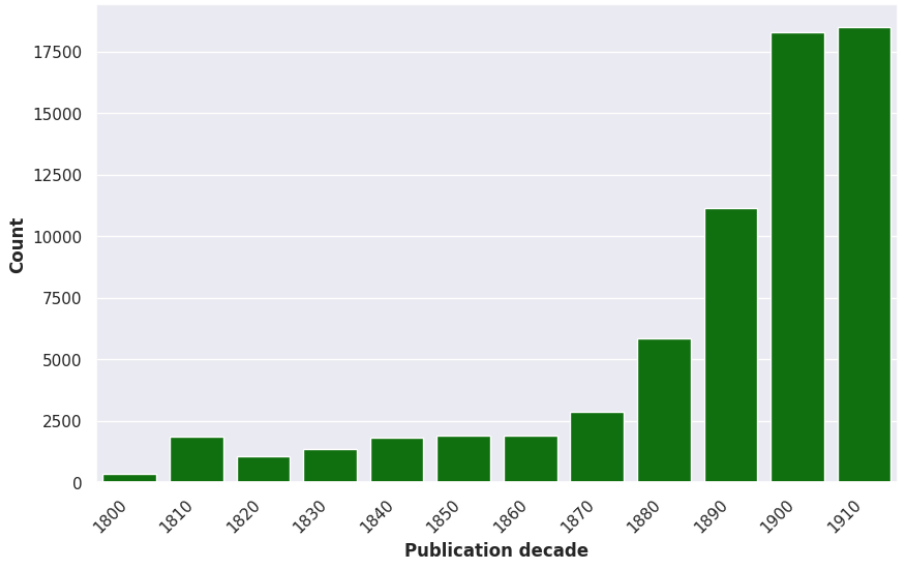

Figure 3.2: Count of documents by publication decade in the MLM dataset

MLM dataset contains a total of 66,890 entries. Figure 3.2 shows the amount of documents published for each decade in the dataset. The figure shows that most of the documents were published in the last four decades of the dataset's time window.

The MLM dataset contains a total of 21 features, including both the "pagecount_orig" and "pagecount" features that were also in the regression dataset. The values of the "pagecount" feature, however, are incorrect in the MLM dataset which is why the dataset cannot be used for regression fine-tuning. The reason for the incorrect page count values is that the function used to harmonize the data did not work correctly. The values of the "pagecount_orig" feature can still be used though for MLM fine-tuning since MLM is a self-supervised task which can be done with unlabeled data.

For the MLM dataset, only the "pagecount_orig" feature was chosen. After discarding all the missing values, the dataset had 60,998 examples left. After that, the entries were cleaned with the same function that was used for the regression dataset in section 4.3. After cleaning the "pagecount_orig" values, all duplicates were removed, leaving the dataset with 15,536 examples left.

Finally, to avoid data leakage, the MLM dataset was filtered by removing all values that also appeared in the regression dataset i.e. all values of the cleaned "pagecount_orig" field that also were in the regression dataset. After filtering out these values, the dataset was left with 15,260 examples. After processing, the dataset was shuffled and divided into a training set of 13,734 and a test set of 1,526 using a 90-10 training-test split. The validation set was omitted because the model's hyperparameters will not be tuned for MLM fine-tuning.

4 Training the pure regression model

This chapter focuses on the training of the first model which is fine-tuned on the regression dataset. The chapter explains the reasoning behind model selection, the data pre-processing done before training, the training set, hyperparameter optimization process, and finally, the results achieved on the validation set of the regression dataset.

4.1 Model selection

The model chosen for this thesis is the "distilbert-base-multilingual-cased" model [1]. A multilingual model was chosen because the values of the "pagecount_orig" field are written in multiple languages such as Finnish, English and Swedish. The fact that most of the entries were written in Finnish also limited the selection to models that support Finnish. The model is a distilled version of the BERT base multilingual model [35]. The model has 134M parameters compared to the 177M parameters of the original multilingual BERT model, and the model is about twice as fast on average for inference and training.

The distilled model was chosen for this thesis because the goal is not to obtain the best possible performance, but to instead see if encoder-only models are viable for the page count prediction task. The distilled model's performance should be close to the performance of the original model [39]. The lower number of parameters can

[1] https://huggingface.co/distilbert/distilbert-base-multilingual-cased

also prevent overfitting because the regression dataset does not contain that many examples in total. Finally, the model was fine-tuned locally using a single Nvidia RTX 4070 GPU. Therefore it is beneficial to use a model that doesn't require as much computational power.

4.2 Model architecture

The model used for regression, shown in figure 4.1, consists of a custom regression head and the pre-trained DistilBERT model. The regression head has two fully-connected layers with a dropout layer between them. The first fully-connected layer has 768 inputs and 768 outputs and the second fully-connected layer has 768 inputs and a single output. The last fully-connected layer uses linear activation whereas the first uses the ReLU activation function. The model was implemented in Python by using the PyTorch and Transformers libraries.

The DistilBERT model produces a contextual representation, which is a 768-dimensional vector, for each token in the input sequence. The contextual representation of the special [CLS] token is used as the input of the regression head. The predictions produced by the model are scaled up by multiplying each prediction by 100. The predictions are scaled up because without scaling, the model failed to predict page counts of more than 400. A few different ways of scaling the predictions were tried, such as min-max normalization, but multiplying by 100 was chosen due to its simplicity and effectiveness. The scaling is done at the end of the model's forward pass, so that the loss is calculated based on the scaled values. After scaling, the predictions are also rounded to the nearest integer because the page count is an integer.

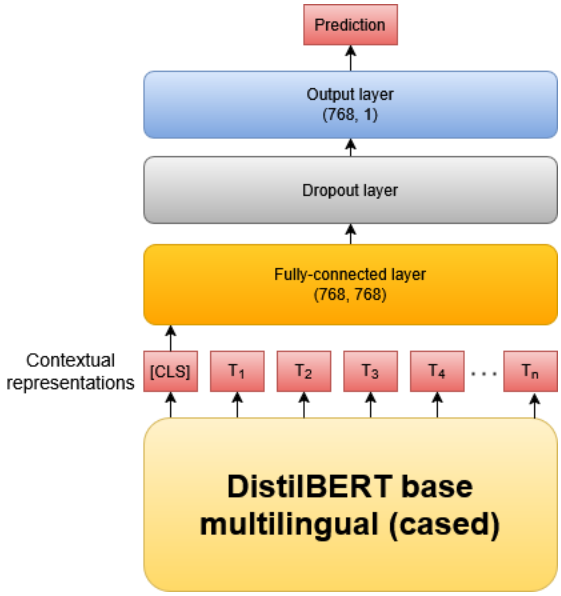

Figure 4.1: Architecture of the whole regression model showing the regression head attached to the DistilBERT model. The first number in brackets shows the amount of inputs and the second the amount of outputs for the fully-connected layers.

4.3 Data preprocessing

To pre-process the data for training the model, the values of the cleaned "page-count_orig" feature were tokenized and then the tokens were converted into a list of input ids. Each unique token in the model's dictionary has its own input id. The ids are then given to the model as input. Figure 4.2 shows the tokenization process for a single example. The tokenization was done by using the DistilBERT model's tokenizer from the transformers library.

Finally, when the inputs were passed to the model, they were padded dynamically so that the length of every input vector was the same for all the examples in a single batch. This is done by adding special [PAD] tokens to the examples, which are then masked by using an attention mask that marks them with a value of zero, meaning that the model will not pay attention to the [PAD] tokens during training.

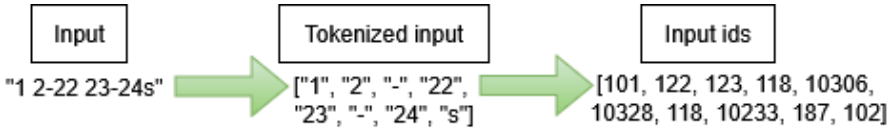

Figure 4.2: An example of tokenizing an input, then converting the tokens to input ids. Note that the input ids list has two more elements than the tokenized input due to the [CLS] and [SEP] tokens being added.

4.4 Training setup

The model was trained using the training set and the performance was evaluated on the validation set after each epoch. For regularization, early stopping with a patience of 3 was used, which means that training was stopped if the model's performance on the validation set didn't improve in 5 epochs. After training, the model checkpoint that performed the best on the validation set was loaded. The model was trained using a batch size of 16 due to GPU memory limitations. A weight decay of 0.01 and a dropout probability of 0.1 were also used for regularization.

The hyperparameter optimization was done by using the optuna library [40]. After trying out optimizing the learning rate, batch size, weight decay, activation function and dropout probability, only the learning rate was chosen for the final hyperparameter optimization since changing it had a larger effect on the model's performance than changing the other hyperparameters. The AdamW optimizer [41] was used with a β_1 value of 0.9 and a β_2 value of 0.99. The values control how the learning rate is adjusted during training. The final hyperparameter optimization consisted of 15 trials where optuna was used to suggest learning rates between 10^{-5} and 10^{-4}.

Each trial consisted of training the model for 10 epochs, or until early stopping was triggered i.e. until the model obtained the lowest MSE loss on the validation set. After training, the model's performance was evaluated on the validation set. The best hyperparameters were chosen by taking the hyperparameters used in the

trial that achieved the smallest MSE. After hyperparameter optimization, the final model was trained for 10 epochs using the best performing learning rate from the trials which was 2.96×10^{-5}.

4.5 Results

The final model achieved an MSE of 1,728 and an MAE of 16.40 on the validation set. The median absolute error (AE) of the predictions was 5 and the 3rd quartile AE was 13. This shows that most of the predictions were close, but that there were some outliers which increased the mean absolute and squared errors. The highest absolute error was 333, and there were a total of 14 predictions that had an absolute error of more than 100.

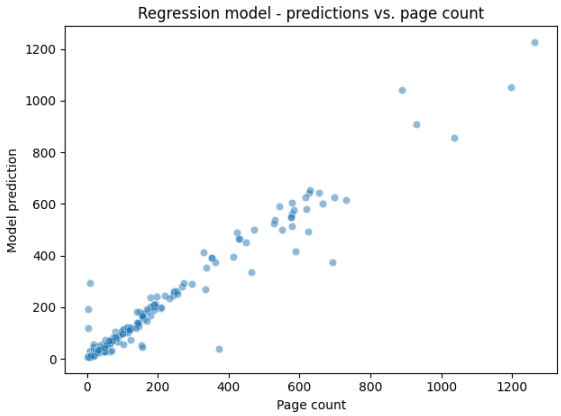

Figure 4.3: The predictions of the regression model vs. the page count values on the validation set.

Figure 4.3 shows the difference between the model's predictions and the real page count values. The figure shows that there were some large errors in predictions for documents that had a low page count. The figure also shows that the validation set only has a few documents with a page count greater than 800. The low amount of high page count documents makes it difficult to judge the model's ability to predict high page count values.

5 Training the MLM fine-tuned model

This chapter explains the training process of the second model, which is first fine-tuned on the MLM task before fine-tuning it on the regression task. The chapter explains the MLM fine-tuning process and the reasoning behind the MLM fine-tuning, summarizes the regression fine-tuning and hyperparameter optimization processes, and analyses the metrics the MLM fine-tuned model achieved on the validation set of the regression dataset.

5.1 MLM fine-tuning

The goal of fine-tuning the model using the MLM task is to help with domain adaptation. The multilingual DistilBERT model used in this thesis was pre-trained on large amounts of general textual data, but the values of the "pagecount_orig" field can differ a lot from general text because they contain a lot numbers, abbreviations and specific terminology. Fine-tuning for the MLM task may help the DistilBERT model to adapt to the bibliographic data domain. The MLM fine-tuning will only affect the base DistilBERT model, whereas the regression head will only be fine-tuned using the regression dataset.

The approach of first fine-tuning the model on the MLM task before regression fine-tuning was inspired by the Universal Language Model Fine-tuning (ULMFiT)

approach [42], introduced by Howard and Ruder in 2018. In the ULMFiT approach, a pre-trained language model was fine-tuned on domain-specific unannotated data using the language modeling (LM) task. After LM fine-tuning the model was then fine-tuned for text classification using a smaller amount of annotated data.

The approach taken in this thesis differs from the ULMFiT approach in a few ways. First, the original ULMFiT approach used an older language model based on the LSTM architecture, whereas in this thesis a transformer model is used. The second difference is that this thesis uses the bidirectional MLM task instead of the unidirectional LM task used by ULMFiT. The usage of MLM instead of LM is enabled by using the encoder-only architecture [13]. Finally, ULMFiT was aimed at improving text classification performance whereas in this thesis the goal is to improve text regression performance.

The dataset used for the MLM fine-tuning, which was originally introduced in section 3.2, only has the cleaned values of the "pagecount_orig" field. During data pre-processing, the values of the "pagecount_orig" field were tokenized and turned into input ids using the DistilBERT model's tokenizer similarly to section 4.3. After tokenization, the input ids of each example were copied so that they could be used as labels during training. Finally, 15 % of all the tokens were masked using the data collator from the transformers library. The masking process replaced the input ids of the original tokens with the id of the special [MASK] token.

To train the model for the MLM task, a learning rate of 2e-5 was used alongside a weight decay of 0.01 and a batch size of 16. The hyperparameters were not optimized for the MLM task, since the MLM fine-tuning is only done to improve regression performance, and the larger size of the MLM dataset made training the model slower, which would have slowed down the hyperparameter optimization process. The training was done with the model's performance being evaluated on the MLM dataset's test set after each epoch. Early stopping with a patience of 3 was

used during training and the best performing parameters were loaded at the end of training. The model was trained for 11 epochs in total, but the best performance on the validation set was achieved after 8 epochs. The loss metric used during training was cross-entropy loss.

5.2 Regression fine-tuning

After MLM fine-tuning, the model was fine-tuned for regression similarly to section 4.4 using the regression dataset. The model architecture was the exact same as for the pure regression model except that the MLM fine-tuned DistilBERT was used instead of the original multilingual DistilBERT model.

The same approach was used for hyperparameter optimization which consisted of 15 trials where the Optuna library was used to suggest learning rates between 10^{-5} and 10^{-4}. The optimized learning rate used for training the final model for 10 epochs was 2.52×10^{-5}. Other than the learning rate, all the hyperparameters were the same as for the pure regression model: a weight decay of 0.01, batch size of 16 and a dropout probability of 0.1 and the AdamW optimizer was used with β_1=0.9 and β_2=0.99.

5.3 Results

For the MLM task the metric used to evaluate the model's performance was perplexity, which is the exponent of the cross-entropy loss. Before MLM fine-tuning, the model achieved a perplexity of 716.51 on the MLM dataset's test set, which dropped down to 5.71 after training. The significant drop in perplexity shows that the MLM fine-tuning made the DistilBERT model adapt to the domain of bibliographic data, at least when it comes to the MLM task.

For the regression task, the MLM fine-tuned model achieved an MSE of 2,540 and

an MAE of 17.34 on the validation set of the regression dataset. These results are worse than what the pure regression model achieved, although the final comparison will be done in the next chapter where both models will be evaluated on the test set. Although the MLM fine-tuned model's median AE of 5 was the same and the 3rd quartile AE of 10 was lower than what the pure regression model achieved, the MSE and MAE were both worse. One of the reasons for this is that the largest AE of the MLM fine-tuned model was 539, and the model had 13 predictions with an AE greater than 100. The results show that the MLM fine-tuned model had bigger outliers in terms of AE which is why the MSE of 2,540 in particular is a lot worse than the MSE of 1,728 the pure regression model achieved. Due to the small size of the validation set, outliers can have a large effect on MSE.

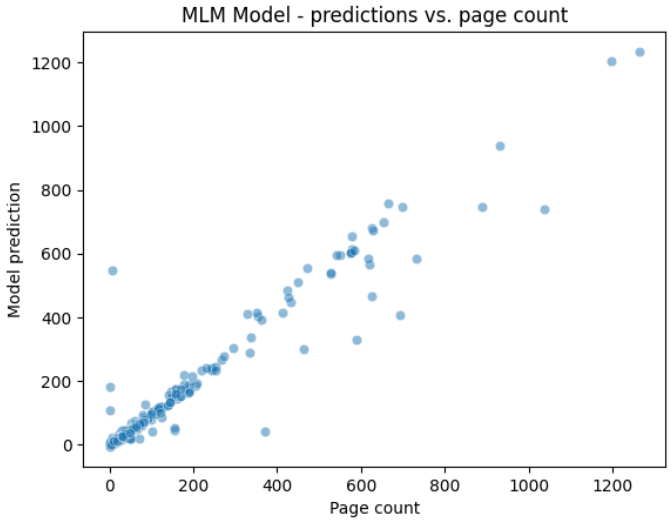

Figure 5.1: The predictions of the MLM fine-tuned model vs. the page count values on the validation set. It can be seen that there were some large outliers in AE for documents that had a page count that was near one. In particular, the largest AE of 539 is shown in the middle-left side of the figure.

Figure 5.1 shows the predictions compared to the page counts on the validation set. The figure shows that the prediction that had the biggest AE (539) was for a document with a very low page count. The example had a "pagecount_orig" value of "8 s 655 se on 639 s 1 s" ("8 p. 655 it is 639 p 1 p" in English) and a page

count of 8. The value of the page count feature for this example might be incorrect, since the description contains more numbers than just 8. If the value is incorrect, the MLM fine-tuned model's prediction of 547 pages might be closer than the pure regression model's prediction of 295 pages.

The results obtained in this chapter show that the MLM fine-tuning did not lead to improved performance on the validation set. However, the final effectiveness of the MLM fine-tuning will be shown in the next chapter, where the performance of the MLM fine-tuned model and the regression model are compared on the test set of the regression dataset.

6 Conclusion

The chapter starts by comparing the two models' performance on the test set of the regression dataset. The chapter also contains discussion about the results, limitations of the results, suggestions for future research and a final summary of the thesis.

6.1 Model comparison

Table 6.1 shows the metrics for both models on both the validation and test subsets of the regression dataset. The models were evaluated only once on the test set after both models had been trained so that the decisions made when training the models were not affected by the results obtained on the test set. On the test set, the MLM fine-tuned model achieved an MSE of 3,597 and an MAE of 17.50. The pure regression model achieved an MSE of 4,971 and an MAE of 21.11. The MLM fine-tuned model performed worse on the validation set, but outperformed the pure regression model on the test set. Both models had a noticeably worse MSE on the test set, and the pure regression model's MAE was also noticeably worse on the test set than the validation set.

Both models had a median AE of 5 on the test set, which is the exact same value they achieved on the validation set. The 3rd quartile AE was 12 for the pure regression model and 11 for the MLM-finetuned model on the test set. The 3rd quartile AE values achieved on the test set were also close to the values achieved

on the validation set. This shows that the main difference between the test and validation set performance is that the test set had more outliers with a large AE. The differences could also be explained by the fact that the models' hyperparameters were optimized using the validation set, which lead to the models slightly overfitting the validation set.

Table 6.1: Results for both models on the test and validation regression sets. The best metrics achieved on both validation and test set are bolded. The MLM fine-tuned model performed better on the test set, even though its performance on the validation set was worse.

Model	**Regression dataset subset**	**MSE**	**MAE**
Pure regression	Validation	**1,728**	**16.40**
MLM fine-tuned	Validation	2,540	17.34
Pure regression	Test	4,971	21.11
MLM fine-tuned	Test	**3,597**	**17.50**

The largest AE on the test set was 915 for the pure regression model and 829 for the MLM fine-tuned model, this example can be seen on the bottom right corner of the graphs for both models in figure 6.1. The example had a "pagecount_orig" value of "s 943-" and a page count value of 943. The page count for this example should have been simple to predict, however, it was by far the largest AE for both models being 465 greater than the second largest AE for the pure regression model, and 517 greater for the MLM fine-tuned model.

The hyphen in the "pagecount_orig" value is most likely the reason why the prediction was off by such a large amount, although it is hard to know precisely how the models make predictions. The hyphens were not removed when cleaning the data because the values of the "pagecount_orig" feature contain page ranges, such as "250-300 p.", which have a different meaning to just having two page numbers back-to-back e.g. "250 300 p.". However, perhaps more careful cleaning of the data, such as removing hyphens that are not between two numbers, would improve model performance.

Figure 6.1 shows a side-by-side comparison of the two models' predictions on the

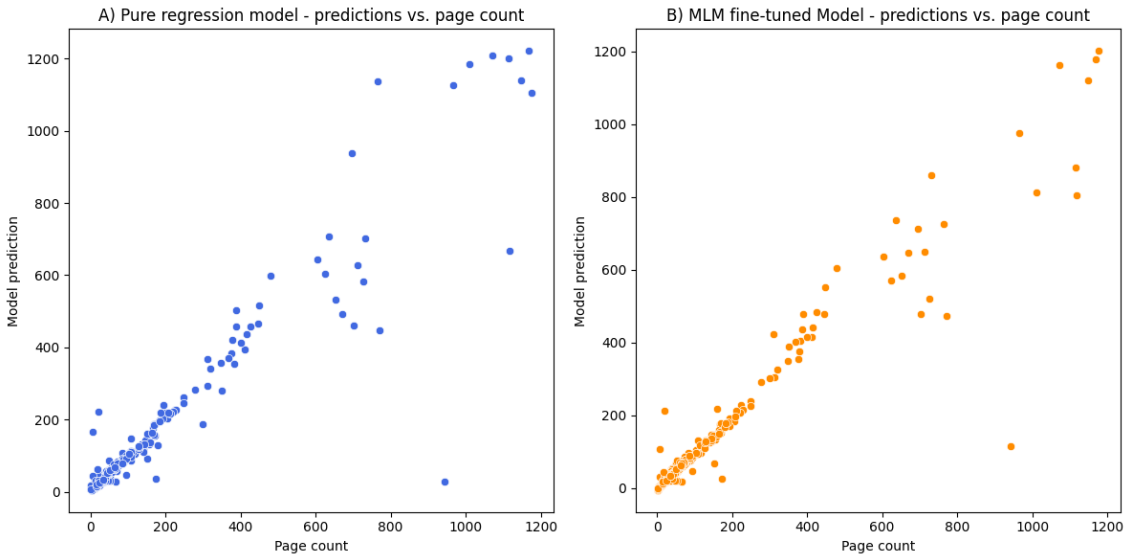

Figure 6.1: Predictions and page counts for both the pure regression (graph A) and MLM fine-tuned (graph B) models on the test set. The MLM fine-tuned model had both a lower MAE and MSE than the pure regression model. The MLM fine-tuned model was more accurate on high page count documents in particular.

test set. The figure shows that the test set had more examples with a higher page count than the validation set, and that both models struggled more with predicting higher page count values. However, the MLM fine-tuned model performed better on the high page count predictions, which is why it obtained better metrics. The results also show that the MLM fine-tuned model generalized better because it obtained better results on the test set and had a smaller difference between test set and validation set performance.

Figure 6.2 shows the distribution of AE for the test set predictions of both models. The figure shows that the MLM fine-tuned model had slightly more predictions that had a low AE, although the regression fine-tuned model also had a low AE for most of its predictions. In total, out of the 354 test set predictions, the MLM fine-tuned model had 263 and the pure regression model had 247 predictions where $\text{AE} \leq 10$.

In general, the small size of the test and validation sets makes it difficult to

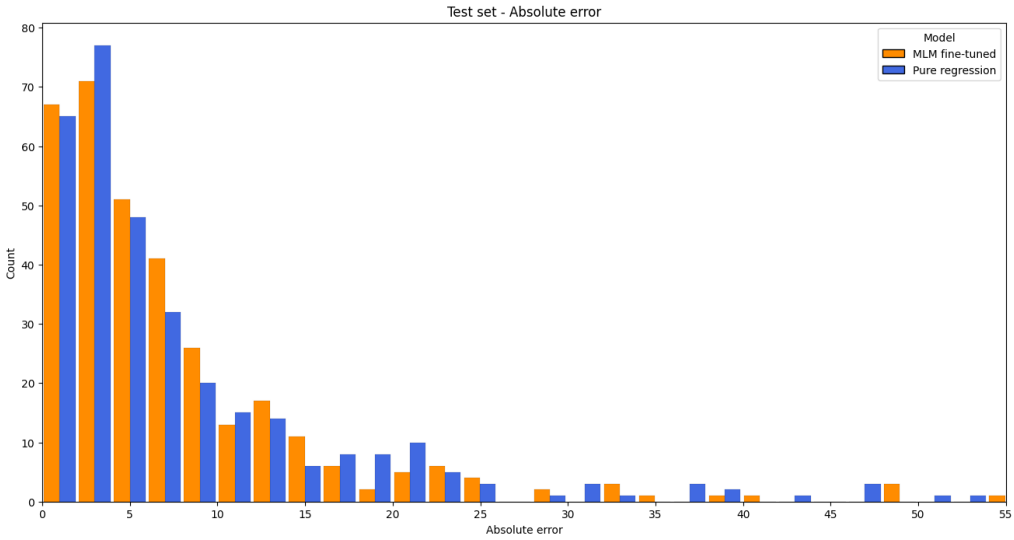

Figure 6.2: Distribution of the predictions' AE for the test set. The pure regression model is shown in blue and MLM fine-tuned model is shown in orange. The 29 predictions for the pure-regression model and the 23 predictions for the MLM fine-tuned model that had an AE higher than 50 were excluded from the graph. The MLM fine-tuned model had more predictions where AE ≤ 10.

evaluate model performance, since just a few predictions being off by a large amount will affect the MSE noticeably. For example, if the example with the highest AE is removed from the results of both models, the MSE drops down to 2,614 for the pure regression model and down to 1,660 for the MLM fine-tuned model. This is a good demonstration of how MSE is affected by outliers, especially when the dataset is small. Removing the example would lower the MAE of the pure regression model to 18.58 and the MLM fine-tuned model's MAE to 15.20 which shows that MAE is more robust to outliers than MSE.

6.2 Discussion

The first research question of this thesis was "Can a transformer encoder model be used to predict the page count of a document based on the value of the MARC21 300a field?". The results obtained on the test set show that the models trained in this thesis can reliably predict the page count of documents that have fewer than

around 600 pages, but struggle to predict the page count of longer documents. The validation set only had a few high page count documents, which could explain why the final models were less accurate at predicting high page count values, because the models' hyperparameters were optimized using the validation set.

The results show that a transformer encoder model could potentially achieve better performance. For example, a larger model trained with more data could achieve better results. The results were also pessimistic because of the removal of duplicates that was done in section 4.3. For real world bibliographic data, it is likely that there would be some duplicate values of the MARC21 300a field, particularly for shorter documents. For example, there could be multiple examples that have a MARC21 300a field value such as "2 pages" or "2 p." for documents that have a page count of two. The reason the duplicate values were removed in this thesis was to avoid having overly optimistic results due to the same "pagecount_orig" and page count values appearing in both test and training sets of the regression dataset.

The second research question was "Can page count prediction performance be improved by fine-tuning the model for the masked language modeling task using unannotated data?". The MLM fine-tuned model obtained noticeably better metrics on the test set, even though it performed worse on the validation set. This shows that the MLM fine-tuned model had better generalization, meaning that it performed better on unseen data. Based on the experiments of this thesis, it can be concluded that MLM fine-tuning can improve a model's regression performance, at least if the amount of annotated data that can be used for regression fine-tuning is limited.

6.3 Limitations

The size of the validation and test sets might threaten the validity of the results since a few outliers can skew the metrics. Model performance on the test set was evaluated only once, but for the validation set, there was some variance in the metrics for

models trained using the same hyperparameters. Therefore, to get more accurate results, more data should be used for both training the model and evaluation of model performance.

The datasets used in this thesis were both subsets of the Finnish national bibliography, Fennica. Most of the MARC21 300a entries which were used as the input, were written in Finnish. The experiment results might not reflect the general applicability of transformers models to page count prediction. For example, a model trained on data taken from another national bibliography, with entries written mostly in another language and potentially with other standards, may perform differently. Also, the documents described by the datasets were published between the years 1488 and 1917, which means that most of the bibliographic data was catalogued a long time ago. More recently catalogued bibliographic data may use different standards for page count entries, which can affect the effectiveness of the approach taken in this thesis. This limitation could be fixed by using a more diverse dataset which contains examples from multiple bibliographies.

Another potential limitation is that the annotated data used for regression fine-tuning did not describe documents that were longer than 1,714 pages. The one outlier example that had a page count of 11,080 was removed due to how it affected the training loss. Generally, the models were less accurate at predicting higher page counts. Therefore, the models used in this thesis might be more applicable to types of documents that have a lower page count. Using a more balanced dataset, which contains more examples with a high page count, for regression fine-tuning could remove this limitation.

Finally, even though some examples that had an incorrect page count value were removed from the regression dataset, there might still have been some incorrect values left in the dataset which affected the results negatively. For example, the validation set had the example with a MARC21 300a value of "8 s 655 se on 639 s

1 s" and a page count of 8. The page count value appears to be incorrect, since the MARC21 300a value also contains other page count values. The MLM fine-tuned model had its worst AE on this example, which affected the metrics obtained on the validation set. More careful processing of the annotated data could potentially lead to improved results.

6.4 Future research

The simplest approach for future research would be using more annotated data for fine-tuning the model for regression. Having more data would reduce the uncertainty of the validation and test set results while also potentially improving performance. In particular, having more examples with a higher page count could lead to better performance. More data could be obtained by integrating data from multiple bibliographies into a single dataset. This would allow the evaluation of models on entries that are written in various languages. Trying the approach of this thesis on data which describes more recent documents could also be interesting since it would show how the cataloging standards affect model performance.

The model selection in this thesis was limited by the small size of the regression dataset and the lack of computational resources, which meant that a smaller model had to be selected. Another limiting factor was that most of the MARC21 300a entries were written in Finnish which is why a multilingual model that supported Finnish was chosen. Without these limitations, there are many different encoder-only models that could be used. If more data were obtained, some larger models such as the large version of the BERT model could be used. There are also other, more recent encoder-only models that could be used, such as ModernBERT [43].

With enough unannotated data, the full pre-training of an encoder-only model could be tried. However, this would require a large amount of computational resources and might not be worth it just for page count prediction. The MLM fine-

tuning done in this thesis mimicked the typical MLM pre-training of an encoder-only model, and it improved the model's performance on the downstream regression task. Full pre-training could potentially lead to even more improvement.

Finally, in cases where the amount of annotated data are limited, more sophisticated training methods could be tried. These methods include freezing some layers of the base encoder-only model either permanently or for some amount of time, so that the parameters of the regression head could be adjusted without the encoder-only model starting to overfit the training data. Another solution to having fewer data, would be trying more efficient models, which can obtain good performance with less computation and memory usage [44]. However, the more efficient models might not support Finnish, which is why data written in other languages might be needed to take advantage of them.

6.5 Summary

In this thesis, the research objective was to predict the page count of documents based on the value of the MARC21 300a field, which is a bibliographic data field that describes the length of a document. The values of the field are short text descriptions, that vary from single numbers to long descriptions containing many abbreviations, words and even roman numerals. Due to the varying standards used for describing the documents, the task of mapping these text values to numerical page counts is non-trivial.

The approach chosen for this thesis, was to use an encoder-only transformer model to produce contextual, vector representations of the MARC21 300a entries. These vectors were then passed to a regression head, which was a simple artificial neural network with two fully-connected layers, that produced the final page count predictions. The advantage of this approach is that it doesn't require manual feature engineering, such as parsing different values from the text descriptions. Using an

encoder-only model also allowed taking advantage of a transfer learning approach, since the model had been pre-trained on a large amount of data by the model's authors. The transfer learning was important, since the amount of annotated data used in this thesis was limited.

Two harmonized subsets of the Finnish national bibliography, Fennica, were used in this thesis. The first dataset, which was called the regression dataset since it was used for regression, contained descriptions of documents released between the years 1488 and 1800. This dataset contained both the MARC21 300a values and previously estimated page counts for each document. The other dataset, called the MLM dataset since it was used for masked language modeling (MLM), contained metadata on documents published between the years 1809 and 1917. Many of the page count estimates of the second dataset were incorrect, which is why the MLM dataset could not be used for regression fine-tuning.

However, an experiment, where the second dataset was used to fine-tune the base encoder-only model using the masked language modeling (MLM) task was done to see if unannotated data could be used to improve regression performance. To evaluate the effectiveness of MLM fine-tuning, a model was fine-tuned only on the first dataset for the regression task. This model was called the pure regression model and its performance was compared to the performance of the MLM fine-tuned model, which was fine-tuned for the MLM task and the regression task.

The final result was that the MLM fine-tuned model performed better on the test set of the regression dataset. However, both models had some outlier predictions that had a large absolute error (AE), the small size of the regression dataset lead to these outliers having a large effect on the performance metrics. In particular, both models performed worse on documents that had a page count greater than around 600.

Overall, it can be concluded that the approach of using an encoder-only model

for page count prediction is viable. The results of this thesis might not be applicable to other bibliographies, since most of the MARC21 300a entries used for the experiments of this thesis were written in Finnish, and the latest documents described by the datasets were published in 1917. For future research, using more data or other encoder-only models should be considered. The approach could also be tried on data from other bibliographies.

References

[1] T. Umerle, G. Colavizza, E. Herden, *et al.*, "An Analysis of the Current Bibliographical Data Landscape in the Humanities. A Case for the Joint Bibliodata Agendas of Public Stakeholders", May 2022, Publisher: Zenodo. DOI: 10.5281/ZENODO.6559857.

[2] L. Lahti, J. Marjanen, H. Roivainen, and M. Tolonen, "Bibliographic Data Science and the History of the Book (c. 1500–1800)", *Cataloging & Classification Quarterly*, vol. 57, no. 1, pp. 5–23, Jan. 2019, ISSN: 0163-9374, 1544-4554. DOI: 10.1080/01639374.2018.1543747.

[3] M. Tolonen, L. Lahti, H. Roivainen, and J. Marjanen, "A Quantitative Approach to Book-Printing in Sweden and Finland, 1640–1828", *Historical Methods: A Journal of Quantitative and Interdisciplinary History*, vol. 52, no. 1, pp. 57–78, Jan. 2019, ISSN: 0161-5440, 1940-1906. DOI: 10.1080/01615440.2018.1526657.

[4] I. Tiihonen, L. Lahti, and M. Tolonen, "Print culture and economic constraints: A quantitative analysis of book prices in eighteenth-century Britain", *Explorations in Economic History*, vol. 94, p. 101 614, Oct. 2024, ISSN: 00144983. DOI: 10.1016/j.eeh.2024.101614. (visited on 02/08/2025).

[5] M. Tolonen, J. Marjanen, H. Roivainen, and L. Lahti, "Scaling Up Bibliographic Data Science", *Digital Humanities in the Nordic and Baltic Coun-*

tries Publications, vol. 2, no. 1, pp. 450–456, May 2019, ISSN: 2704-1441. DOI: 10.5617/dhnbpub.11118.

[6] *Fennica – the Finnish National Bibliography | Kansalliskirjasto*. [Online]. Available: https://www.kansalliskirjasto.fi/en/services/fennica-finnish-national-bibliography (visited on 03/12/2025).

[7] T. D. S. Group, *Fennica metadata conversions: Statistical monitoring and analysis*, Feb. 2025. [Online]. Available: https://fennica-fennica.2.rahtiapp.fi/ (visited on 03/23/2025).

[8] *The MARC 21 Formats: Background and Principles*. [Online]. Available: https://www.loc.gov/marc/96principl.html (visited on 03/12/2025).

[9] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention Is All You Need", Aug. 2023. DOI: 10.48550/arXiv.1706.03762.

[10] T. M. Mitchell, *Machine learning* (McGraw-Hill series in Computer Science). New York: McGraw-Hill, 2013, ISBN: 978-0-07-042807-2 978-0-07-115467-3.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[12] J. Gui, T. Chen, J. Zhang, *et al.*, "A Survey on Self-Supervised Learning: Algorithms, Applications, and Future Trends", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, pp. 9052–9071, Dec. 2024, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2024.3415112.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", May 2019. DOI: 10.48550/arXiv.1810.04805.

[14] I. N. Da Silva, D. Hernane Spatti, R. Andrade Flauzino, L. H. B. Liboni, and S. F. Dos Reis Alves, *Artificial Neural Networks*. Cham: Springer International Publishing, 2017. DOI: 10.1007/978-3-319-43162-8.

[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14539.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory", vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[17] T. Kohonen, "The self-organizing map", *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, Sep. 1990, ISSN: 00189219. DOI: 10.1109/5.58325.

[18] D. E. Rumelhart, G. E. Hintont, and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, Oct. 1986. DOI: https://doi.org/10.1038/323533a0.

[19] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, arXiv:1412.6980 [cs], Jan. 2017. DOI: 10.48550/arXiv.1412.6980.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[21] J. Hirschberg and C. Manning, "Advances in natural language processing", *Science (New York, N.Y.)*, vol. 349, pp. 261–266, Jul. 2015. DOI: 10.1126/science.aaa8685.

[22] J. Eisenstein, *Natural Language Processing*. The MIT Press, Oct. 2019, ISBN: 978-0-262-04284-0.

[23] Q. Jiao and S. Zhang, "A Brief Survey of Word Embedding and Its Recent Development", in *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, Chongqing, China: IEEE, Mar. 2021, pp. 1697–1701, ISBN: 978-1-7281-8028-1. DOI: 10.1109/IAEAC50856.2021.9390956.

[24] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient Estimation of Word Representations in Vector Space*, arXiv:1301.3781 [cs], Sep. 2013. DOI: 10.48550/arXiv.1301.3781.

[25] J. Pennington, R. Socher, and C. Manning, "Glove: Global Vectors for Word Representation", in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162.

[26] M. Peters, M. Neumann, M. Iyyer, *et al.*, "Deep Contextualized Word Representations", in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202.

[27] H. Larochelle and G. E. Hinton, "Learning to combine foveal glimpses with a third-order Boltzmann machine", in *Advances in Neural Information Processing Systems*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., vol. 23, Curran Associates, Inc., 2010.

[28] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention", in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14, Montreal, Canada: MIT Press, 2014, pp. 2204–2212.

[29] D. Bahdanau, K. Cho, and Y. Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate*, May 2016. DOI: 10.48550/arXiv.1409.0473.

[30] A. Galassi, M. Lippi, and P. Torroni, "Attention in Natural Language Processing", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32,

no. 10, pp. 4291–4308, Oct. 2021, ISSN: 2162-2388. DOI: 10.1109/TNNLS.2020.3019893.

[31] B. Min, H. Ross, E. Sulem, *et al.*, "Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey", *ACM Computing Surveys*, vol. 56, no. 2, pp. 1–40, Feb. 2024, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3605943.

[32] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding by Generative Pre-Training", en, 2018.

[33] C. Raffel, N. Shazeer, A. Roberts, *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer", *Journal of Machine Learning Research*, vol. 21, Jun. 2020.

[34] Y. Wu, M. Schuster, Z. Chen, *et al.*, *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*, Oct. 2016. DOI: 10.48550/arXiv.1609.08144.

[35] *Bert/multilingual.md at master · google-research/bert*, en. [Online]. Available: https://github.com/google-research/bert/blob/master/multilingual.md (visited on 03/09/2025).

[36] A. Virtanen, J. Kanerva, R. Ilo, *et al.*, "Multilingual is not enough: BERT for Finnish", Dec. 2019. DOI: 10.48550/arXiv.1912.07076.

[37] L. Martin, B. Muller, P. J. O. Suárez, *et al.*, "CamemBERT: A Tasty French Language Model", 2020. DOI: 10.18653/v1/2020.acl-main.645.

[38] J. Cañete, G. Chaperon, R. Fuentes, J.-H. Ho, H. Kang, and J. Pérez, "Spanish Pre-trained BERT Model and Evaluation Data", Aug. 2023. DOI: 10.48550/arXiv.2308.02976.

[39] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter", Mar. 2020, arXiv:1910.01108 [cs]. DOI: 10.48550/arXiv.1910.01108.

[40] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework", in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[41] I. Loshchilov and F. Hutter, *Decoupled Weight Decay Regularization*, Jan. 2019. DOI: 10.48550/arXiv.1711.05101.

[42] J. Howard and S. Ruder, *Universal Language Model Fine-tuning for Text Classification*, May 2018. DOI: 10.48550/arXiv.1801.06146.

[43] B. Warner, A. Chaffin, B. Clavié, *et al.*, *Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference*, Dec. 2024. DOI: 10.48550/arXiv.2412.13663.

[44] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient Transformers: A Survey", *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–28, Jun. 2023, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3530811.