Universiteit Leiden
The Netherlands

Opleiding Informatica

Deep Text Matching
in E-Commerce

Tim Poot

Supervisors:
Suzan Verberne & Terry Peng

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

10/01/2020

Abstract

In this thesis we take a look at how useful deep information retrieval in e-commerce can be. Specifically we take a look at the performance of Google's BERT at trying to predict the relevance between queries and product descriptions as provided by Home Depot in a Kaggle competition. We try one binary classification model for insight purposes and three regression models to see how they perform against a simpler, more traditional model. Unfortunately, none of the models are able to beat the more traditional method and with current results it is impossible to conclude why this is. It does provide possible leads that could be followed for further research on both deep information retrieval and more specifically BERT in e-commerce.

Contents

1 Introduction

1.1 Text Matching in E-Commerce

E-commerce is the act of buying or selling products via online web shops. This digital platform is ever-growing and in the year 2019 alone there will have been $3.46 trillion worth of e-commerce sales [1]. For businesses running such platforms it is of great importance to have a search engine that produces results similar to what the user is looking for. For instance, if a user is looking for a cocktail-dress but isn't familiar with the term cocktail-dress, they might search for 'little black dress'. In this situation it is important for a search engine to look past just the words in the query but understand the search intent of the user. More accurate matches might encourage the user to spend more money on a business platform.

One way of improving the performance of the search engine is altering the information that is used to determine how good a query-product pair is. These fields of information might include things like the product title, product description or product attributes. It is very common for web-shops to save queries and click-through data of customers using their website, creating a data set that can easily be used to determine how relevant a product is to a query. Training models on such a data set can provide a better understanding of the search intent of the user, especially if the data set consists of a lot of rows. Using our 'little black dress' example from before, including these words in the product description of black cocktail-dresses might make such dresses easier to find for the user.

However training such a model often requires a deep understanding of the products and might involve costly feature engineering. Features in product descriptions describing clothes might vastly differ from features describing home appliances. Finding such features can be an expensive process, especially if a web-shop sells a wide variety of items.
But even for shops that limit themselves to only one category of products, a constant influx of new products can change the behaviour of the user. This too leads to frequent retraining of the web-shop's model.

This is why a model that is based on deep text matching instead of feature engineering can be very useful. Deep text matching takes two texts and gives some output, depending on what the problem is. Many such models exists for a various natural language processing (NLP) problems, such as matching of questions/answers and paraphrase detection [GFJC19]. Deep text matching tries to capture a general understanding of texts, making them widely applicable if there is a model that suits your needs.
As of late there has also been an increase in word embeddings [PNI+18] and models [DCLT19] that are trained on very large corpora and as such have a very broad understanding of natural text. These models with very little additional fine-tuning can obtain state of the art results in various NLP tasks.
If such a model can be trained to accurately predict how relevant a product is to a user based on their query, it could be deployed in all sorts of e-commerce markets. Although some fine-tuning is still necessary, it is still much less costly and less time consuming than extensive feature engineering. Hence why we will be researching how a deep text matching model compares to a more traditional

[1] https://www.digitalcommerce360.com/article/global-ecommerce-sales/

feature engineering model. In specific we will look at Google's BERT as a deep text matching model and how it can be best used to in e-commerce.

1.2 Thesis overview

First we will take a look at work that has already been done in information retrieval (IR) in E-commerce in chapter 2. We will both take a general look, talk about the Home Depot data-set in specific and end with a look at state of the art deep text matching models. In chapter 3 we will describe the various methods that we will try, discussing both different models but also pre-processing. Results of these methods will be shown in chapter 4 and discussed in chapter 5. Finally in chapter 6 we will conclude if the ways in which we applied deep text matching can be used as an alternative to feature engineering in e-commerce.

2 Background

Information retrieval (IR) is a very broad term, even when we limit it within the bounds of NLP. Very loosely put, any act of targeted retrieving of information from unstructured documents is information retrieval. The most widely know example of IR would probably be Google's search engine. Here you have web pages from the entire internet as unstructured documents and you want to retrieve the most relevant web page to a user inputted query. To determine the most relevant web page a lot of features are considered, including features in the content of the web pages but also in the meta data about the pages.
Another, non digital, example of this would be a library. Here you have a great amount of unstructured documents in the form of books. Say you want a book about the history of tea, this is the target. You would go to a librarian or use the library's database to find the book that best suits the target.
One could argue that the books as documents are actually structured. After all they are usually sorted by genre, author's name or other factors in a library. However this ordering is not an inherent property of the books. Rather the library uses features of the book to make searches easier. This is called feature engineering and is explained later in this chapter.

For our purposes we want to estimate the relevance of a query-document pair based on nothing else but the search query, the product title, the product description and the product attribute as provided by Home Depot. In 2016, they held a competition on Kaggle[2] where they provided competitors with user queries, corresponding documents about a product and a relevancy label. With this competition they were hoping to improve their own search engine. The contest is long over but the data is still available and late submissions can still be made to see how well a model performed.

2.1 Information Retrieval in E-Commerce

Traditionally IR in any field of NLP meant feature engineering [QLXL10], e-commerce is no exception [SWHL06] [KSSZ17]. Feature engineering is extracting features from unstructured texts that make it easier to train machine learning models. Within feature engineering we can make a distinction between these domain dependent and independent features.

Also common are ontology based approaches [CHH+07]. Where a document is conceptualised, meaning that all entities are extracted and are connected via certain relations such a 'is a category of', in essence creating a graph. With this graph the similarity between two words can be estimated and also in what way they are similar by looking at the distance of the words in the graph and what type of relations they have. Although a common approach, it is not directly relevant to this thesis.

Domain independent features are features that can be extracted from any data-set, regardless of its contents. A very common example of this is called term frequency-inverse document frequency (TF-IDF) [SB88] where you check how often a word from one string occurs in another string (term frequency) and is normalised with how frequently the term appears in documents. This way common words that hold little information have a low TF-IDF and rare words a higher TF-IDF.

[2] https://www.kaggle.com/c/home-depot-product-search-relevance/overview

Another feature might be to take the length of the longest common sub-sequence between two strings.

Domain dependent features can only be used on certain data-sets and the most effective of such features can require a lot of knowledge about the domain and the data set. Some more simple examples for the Home Depot data set would be things like material, dimensions and brand names. If there is a match between the query and the document for any of those features, it is very likely that the document is at least somewhat relevant to the search intent.

It is easy to see that domain independent features do not rely on the content of the data set. As long as we are talking about natural language, any data set is going to contain common and less common words and sub sequences. Many of these features are widely known and available. TF-IDF is even implemented for users in Scikit-Learn[3] , one the most common python libraries in machine learning. Since they are so readily available, they cost next to nothing to implement are used in some form in many NLP applications.

Domain dependent features can be much more costly to engineer. Coming up with the examples stated earlier can be done by taking a quick glance at the data set. Implementing the extraction isn't much more difficult, just look for certain matches to pre-determined substrings like 'wood' for materials and 'Delta Vero' for brands. However with proper expertise of the domain, very specific features that are also highly effective can be engineered. It is hard to give a realistic example without proper expertise of home appliances but for the sake of illustration here is what such a feature might look like: Say you know exactly how many screws a product includes and based on substrings in the query you could make a close approximation of the amount of screws needed, than you have a very effective feature. But the knowledge required for this would be very specific and even with the knowledge, creating a database for over a hundred thousand products and their corresponding screw count would be a very time consuming job. This is why feature engineering can be a very expensive process, both in time and resources.

2.2 Deep Information Retrieval in E-Commerce

As the name suggests, deep information retrieval combines Deep Learning and IR in an effort to gain similar or better results compared to traditional methods while decreasing the costs. It uses neural networks to retrieve information from texts. An example from e-commerce would be Microsoft's research into training Deep Learning models with clickthrough data as labels [HHG+13]. As neural networks performs matrix multiplications on vectors to get their output, these texts need to be converted to vectors before they can used as input. These vectors are often called the embeddings of a text.

Many methods of creating embeddings exist. A simple way would be to calculate the TF-IDF of each word and convert each text to a vector with a length equal to the amount of words it have where every words is represented by its TF-IDF score. This does embed some understanding of the text into the vector, if you have a large enough data-set changes are many of these scores are unique and a document containing a lot of common words probably has little in common with a

[3]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

document which contains mostly rare words.
Yet it is also easy to see that a lot of information is lost in this embedding. Say there is web-shop with a catalogue consisting of a lot of different headphones and TV-monitors but only a few earbuds. If a customer is looking to buy headphones, the embedding is more likely to indicate that TV-monitors are similar to headphones than earbuds.

This is why neural networks are also often used for creating embeddings. An example of such a method to create embeddings would be word2vec [MCCD13]. These types of embeddings learn what words are semantically similar to each other and gives such words similar embeddings. For instance, it can look in what context words are used. Going back to the example of the web-shop, since headphones and earbuds have similar descriptors it will probably pick up on the similarity of the two words.

This method can reduce the cost of information retrieval because it relies less on features. This does not mean however that it does not need to be configured in its own way. A proper embedding needs to be chosen but also choosing the exact model and tuning its hyper-parameters can be a lengthy process.

2.3 Previous work on the Home Depot dataset

When looking at other submissions for the Kaggle competition, we see a wide variety of approaches have been taken. Simply counting how often words in the query appear in the product title and description being amongst one of the most basic approaches. While simple, this method almost beats 50% of all other participants[4].

When taking a look at the top of the leaderboard, things become a great deal more complicated. First of all, participants in this performance range have not shared their code, making it very hard to deduce exactly what they did. Fortunately they do discuss their methods to various extends[5]. Top participants employed a wide variety of both feature engineering and neural networks. Things like brands, color, material, sizes and power usages are all extracted from text as domain dependent features. TF-IDF and longest common sub sequence can be found as domain independent features.
These features are then often put into multiple layers of models, either neural networks or something like RandomForest.

In most practical cases, this is what you see as well. Deep learning is used in combination with feature engineering to achieve top results. This can be done in many ways. For example, the numerical values of features may form an input vector for a neural network or vice versa, the output of a neural network may become a feature.

[4] https://www.kaggle.com/wenxuanchen/sklearn-random-forest
[5] https://www.kaggle.com/c/home-depot-product-search-relevance/discussion/20427

2.4 BERT

In 2018 Google introduced their Bidirectional Encoder Representations from Transformers, BERT. BERT is a deep learning model that has been state of the art for many NLP tasks ever since its initial paper [DCLT19].

Besides its performance, it comes with many other advantages. Pre-trained BERT models have been made available, among others one that was pre-trained on a corpus of 3.3 billion words, meaning that it contains a lot of knowledge about natural language out of the box. The knowledge comes from BooksCorpus (800 million words), a corpus specifically created for such training tasks, and Wikipedia (2500 million words). The size combined with the generic nature of the corpora give the model a robust understanding of natural language the can be applied in many different ways, as is evident by the variety of the GLUE tasks it excels in[DCLT19]. It also means that train times for BERT are relatively short, as it already understands the basics of natural language. Some training is still required, for example to pick up on intricacies of the data-set at hand.

The model is also very easily modified by taking the output of BERT and putting it in another model, also called a head. Every input node has exactly one output node which can easily be passed on. This is often how fine-tuning is done in practice: BERT is set with a low learning rate as to not diminish its already vast knowledge and the head has a much higher learning rate in the hopes that it can pick up on the things BERT doesn't already know.

No documents could be found on trying to use BERT on the Home Depot data set. More general papers about using BERT in e-commerce are also hard to find, probably because such methods are can be valuable business secrets that give businesses an advantage over their competition.
Regardless, BERT has proven itself in many NLP tasks, as the results listed in its own paper were replicated by others and it is even used as the base for newer, smaller models[SDCW19]. It is not strange to assume that it is able to contribute in e-commerce as well.

3 Methods

The goal of this thesis is to see how BERT compares to a more traditional feature engineering based approach in e-commerce. Specifically, we compare the root mean square error (RMSE) on the test data set as provided by Home Depot of 4 different implementations of BERT against one feature engineering model. Three of these BERT models are regression models, the fourth works with a simplified version of the problem and is a binary classification model. Besides comparing them to a baseline, they will also be compared to each other to see what implementation yields the best results.

3.1 Data

The data set consists of 74067 rows of train data, that is a query, a document and the relevance of the document to the query. The test data consists of 166693 query document pairs without relevance labels. Each query document pair is evaluated by at least 3 different human raters. Each rater can give a score range from 1, not relevant at all, to 3, very relevant. The average score is than taken, resulting in a relevance label that is a real number in the range $[1, 3]$.
It should be noted that the human rankers did not have access to the attributes but they did have access to product images, which are not present in the data set.

The data set comes in 4 different files.

- **train.csv**: Containing the queries, the product titles, the relevancy labels and a unique id for the query-document pair.
- **product_descriptions.csv**: Containing detailed product descriptions.
- **attributes.csv**: Containing dictionary like attributes for products.
- **test.csv**: Much like train.csv, this file contains queries, product titles and a unique id for each pair but it does not contain any labels. As the name suggests this is the test sets and these predictions determine the final score.

For all 4 files, each row also is a unique product-id. Such a product-id may occur multiple times in train.csv or test.csv but be paired with a different query. This is not the case for every product and there is no rule for how often a product can occur in the query document pairs.
For a query document pair, this is what all the relevant information may look like:

train.csv:

```
"id","product_uid","product_title","search_term","relevance"
2,100001,"Simpson Strong-Tie 12-Gauge Angle","angle bracket",3
```

product_descriptions.csv:

```
"product_uid","product_description"
```

100001,"Not only do angles make joints stronger, they also provide more consistent, straight corners. Simpson Strong-Tie offers a wide variety of angles in various sizes and thicknesses to handle light-duty jobs or projects where a structural connection is needed. Some can be bent (skewed) to match the project. For outdoor projects or those where moisture is present, use our ZMAX zinc-coated connectors, which provide extra resistance against corrosion (look for a ""Z"" at the end of the model number).Versatile connector for various 90 connections and home repair projectsStronger than angled nailing or screw fastening aloneHelp ensure joints are consistently straight and strongDimensions: 3 in. x 3 in. x 1-1/2 in.Made from 12-Gauge steelGalvanized for extra corrosion resistanceInstall with 10d common nails or #9 x 1-1/2 in. Strong-Drive SD screws"

attributes.csv:

```
"product_uid","name","value"
100001,"Bullet01","Versatile connector for various 90 connections and home repair projects"
100001,"Bullet02","Stronger than angled nailing or screw fastening alone"
100001,"Bullet03","Help ensure joints are consistently straight and strong"
100001,"Bullet04","Dimensions: 3 in. x 3 in. x 1-1/2 in."
100001,"Bullet05","Made from 12-Gauge steel"
100001,"Bullet06","Galvanized for extra corrosion resistance"
100001,"Bullet07","Install with 10d common nails or #9 x 1-1/2 in. Strong-Drive SD screws"
100001,"Gauge","12"
100001,"Material","Galvanized Steel"
100001,"MFG Brand Name","Simpson Strong-Tie"
100001,"Number of Pieces","1"
100001,"Product Depth (in.)","1.5"
100001,"Product Height (in.)","3"
100001,"Product Weight (lb.)","0.26"
100001,"Product Width (in.)","3"
```

Note that that attributes for 1 product are spread over multiple rows and are also formatted in a very unnatural text manner. The test data in test.csv is formatted exactly as train.csv except that it has no relevance column.

3.2 Shared Parameters

Most methods in this thesis will use some version of the HuggingFace implementation of BERT alongside all of the tools that come with the library, for more details see section 3.5. As such there are many parameters that are shared between the methods, even though the inner workings might differ. Unless otherwise stated, the following can always be assumed for models using BERT:

- The learning rate of the BERT model is set to $2e^{-5}$. It is set this low to avoid losing any of the pre-trained knowledge that is already in the model.
- The scheduler takes 100 warm-up steps and 1000 total steps
- The model is trained over 2 epochs, various tests showed that more epochs mostly prolonged training times with little increased performance
- The data-set is split in to 10% test-data and 90% train-data to avoid over fitting
- The loss function used for training is RMSE, calculated by taking the square-root of the built in mean square error PyTorch function[6]. A lower RMSE means a better performance.

3.3 Pre-Processing

3.3.1 Required

Before a model can be trained, some pre-processing is required. Since the data comes in 4 different files and all of the files contain useful information, we need to concatenate them into one big document as we can't enter the files separately. Luckily all files also include a column that contains a unique product-id, making it relatively easy to merge them all together.

Concatenating the product title and product description is easy. There are by far the most descriptions compared to titles or attributes so we choose to merge the descriptions and titles on product id on descriptions. This creates as many rows as there are descriptions while automatically discarding title duplicates that exist in train.csv.

The attributes are more challenging, albeit only slightly so. In the attributes file, each row contains a product id and one of many attributes corresponding to that product. Meaning one product is usually described over many rows of the file. The biggest challenge is determining how to concatenate these list like formatted attributes into natural text. Because these attributes often have nothing to do with each other, they are simply concatenated by adding a comma and a space in between the attributes, adhering to their initial list like formatting.
Once this is done the attributes are easily merged with the already merged title and description. Again this is done by doing a merge on the descriptions on the product id. Creating one big document that can almost be used as input to our model. Some `NaN` values do remain in the data set and can cause problems later on so these are replace by a single whitespace character.

From here on, little pre-processing is still required. BERT uses two special tokens that need to be at specific positions in the input.

[6] https://pytorch.org/docs/stable/nn.html#mseloss

- **[CLS]** needs to be at the very front of the string. BERT is a very versatile model and a lot of its uses, this token can be ignored. For text matching like this we will be doing however, the model can be trained to output relevancy of the texts in this token since it holds no contextual information.
- **[SEP]** Needs to be at the very end of the input and optionally can be in between the query and the document if they are concatenated into one input. Respectively indicating the end of the string or acting as a separator of the query and the document.

As the separator token already suggests, one can choose to concatenate the query and document pair into one big output. In this case you get one big input. Henceforth this input will be referred to as a concatenated input and looks like this:

`[CLS] query [SEP] document [SEP]`

You can also choose to input the query and document into BERT separately and then create a prediction via an ensemble, which we will discuss later in this chapter. We will refer to this input as a separated input the and it looks like this:

`[CLS] query/document [SEP]`

Now that we have our input in a usable format we can tokenize it, splitting the input into tokens. If we are using a concatenated input, we need to create a segment mask. A segment mask is an extra embedding that BERT uses to differentiate between the first and second text. The length of this mask is the same length of the tokenized input. The value of a point in the segment mask is 1 if it is part of the first text (up until the first [SEP] token), all other values are 0. This step can be skipped if separated input is used.
The tokens now need to be translated to their corresponding ids according to the word-embedding. Once that is finished we need to either truncate or pad each input since BERT requires all inputs to be of the same length. Padding in this case just means appending zero's at the end of the input until it is of maximum length. In this thesis the maximum length will always be 200, BERT can handle longer sequences but memory does quickly become an issue. If there are sequence masks, these need to be padded or truncated as well.
To help BERT differentiate between padding and actual text, another padding mask is generated. It is of max length and individual values of the mask are 1 if the corresponding index in the input is text (does not equal a 0) and 0 otherwise.

Now that we translated our input to embedding id's, padded or truncated it, created a padding mask and possibly created a segment mask, we can start training a BERT model.

3.3.2 Data cleaning

Besides pre-processing that needs to be done in order to provide valid inputs for BERT, some pre-processing to improve the performance has also been done. The Home Depot data set can appear quite noisy to BERT. There are some obvious culprits such as little bits of HTML (`<b>`, `&amp` etc.) that can decrease performance. Words containing HTML tags or parts of HTML tags are removed from the document.

The list like structure of the original attributes file also poses a problem. Many of the attributes were in literal itemised lists, each attribute starting with a bulletpoint. This bulletpoint is can be found in the attributes files as `Bullet[1-99]`. This provides no further understanding of the document so these strings are removed in their entirety as well.

Finally there is the issue of abbreviations of measurements in the data-set. When describing products in home appliances, it is very common to use all sorts of measurements. Feet for the dimensions of wood, pounds for things you want to attach to a wall or even cubic feet for a new washer. All of these examples can be found in the data set. But rather than using natural text for measurements, the data set has them as abbreviations such as 'ft.', 'pds.' and 'cu.'. For a model that is specialised in home appliances, this might not be a problem. Unfortunately BERT isn't trained on any specific field but rather trained on a very large general language corpus, as stated in section 2.4, and it might not pick up on these abbreviations. Thus these abbreviations are replaced by their natural language counterparts so 'ft.' becomes feet, 'pds.' becomes pounds and 'cu.' becomes 'cubic'.

3.4 Baseline

Many people have submitted their predictions to the Kaggle competitions. Unfortunately only few contestants have chosen to share their code, rather opting to discuss their methods. Although this can provide a look into the methodology of other contestants, reproducing their results with only this info remains rather difficult.

Luckily there is a contestant that did decide to share their code that is both easy to understand and preforms reasonably well[7]. This user scores a RMSE of 0.487 which places him at position 1210 of the 2123 contestants on the leaderboard. The lowest (best) RMSE on the public leaderboard is 0.433.
It uses only three features: the length of the query in characters, the number of times words in the query can be found in the product title and the number of times words in the query can be found in the product description. These features are then put into a RandomForestRegressor which in turn passes its output to a BaggingRegressor which gives the final output. The entirety of the train set is used to train the model, so there is no evaluation step.

This method will be reproduced and used as a baseline, mainly because it is an ideal combination of performance and simplicity. Simplicity is important not only for accurate recreation but also because we do not expect to beat the more complex entries. These entries can use over a dozen of features and do extensive pre-processing on the data set. Trying to beat these entries would involve an amount of work that is outside the scope of this thesis.
That being said, it still performs relatively well, beating both Kaggle benchmarks[8].

[7] https://www.kaggle.com/wenxuanchen/sklearn-random-forest
[8] https://www.kaggle.com/c/home-depot-product-search-relevance/leaderboard the public leaderboard

3.5 HuggingFace Transformers

HuggingFace Transformers is a library that made an implementation of BERT as described in it's original paper. Even the creators of BERT acknowledge this third party implementation[9]. It includes both the cased and uncased variants and the basic and large model, already pre-trained. Cased and uncased can intuitively be used for cased and uncased texts respectively. The basic and large model have the same basic infrastructure but the large model consists of more hidden layers, a larger hidden size and more self attention heads. Ultimately leading to longer train times and varying degrees of increased performance[DCLT19]. It also comes with the BERT tokenizer, an optimiser and various schedulers.
The library is well documented[10] and easy to use hence why all methods using BERT will use the basic HuggingFace implementation along with its tokenizer, optimiser and a scheduler. This thesis uses the basic model of BERT. HuggingFace also comes with versions specialised in one certain NLP task. These have undergone additional pre-training for their specific task. Examples of such tasks would be sequence classification, next sentence prediction or predicting if a sentence is the direct follow up of another sentence.

3.5.1 Next Sentence Prediction

Out of all the modified versions of BERT, we are especially interested in the model specialised in next sentence prediction (NSP). Out of all of the out-of-the-box models, this model most closely resembles our problem. Checking if two texts are semantically similar is important to both query-document pairing and next sentence prediction. Unfortunately it is not a perfect fit. The model only works as a binary classification model, so it can only predict 0 for relevant or 1 for irrelevant. Because of this we also need to map our labels from real values in the range $[1, 3]$ to either 0 or 1, ultimately giving us a simplified version of our problem.

Despite these caveats, running this model on our data set can still be useful. There is no previous work that suggests BERT will be effective at all given our data-set. Since this version is easy to implement, designed for a problem similar to ours and already had additional pre-training, it is a useful indicator for the possible performance of BERT with our data set. Although we should keep in mind that this is a greatly simplified version of our problem, with simplified labels and predictions.

To train this model we need to first translate the labels from real values to binary values. Since the real values are between 1 and 3, giving us the average of 2, all values of 2 and above translate to a 0. All values below 2 translate to a 1. Do note that this means that a label of 0 means the next sentence **is** a direct follow up of the previous sentence, a label of 1 means no such relation exists. As a function:

[9] https://github.com/google-research/bert update of november 5th 2018
[10] https://huggingface.co/transformers/index.html

$$f(x) = \begin{cases} 0, & \text{if } x > 2 \\ 1, & \text{if } x \leq 2 \end{cases}$$

Once the labels are simplified, the model takes a concatenated input, making a corresponding segment mask also required. It outputs a prediction and a loss if labels are provided. For the prediction: the classifier token of the output is passed to a linear transformer that gives a final output like a linear regression problem. Meaning that it gives a array of size *number_labels* in our case 2 where each value represents a probability of it's corresponding index being the correct label. The loss is calculated with a cross entropy loss function.

3.5.2 Classification token

The simplest way of getting a real value prediction from BERT would be to put the concatenated output into BERT and take the value of the output of the classification token (CLS token) as a prediction. We do not expect this method to yield great results, HuggingFace even warning users that it should not be used this way[11]. It is however the easiest way to get a prediction without simplifying the problem and will server as a benchmark for all the more complicated models used in this thesis.

Just taking the unmodified representation of the classification token as outputted by BERT is unfortunately impossible. Each representation of a token in BERT is a vector of 768 real numbers (for the base model), and the problem requires just one real value. To solve this problem, we take the representative vector of the CLS token and put it in a neural network that consists of one linear layer that takes 768 inputs and outputs only one real value.

3.5.3 Cosine similarity

In linear algebra, cosine similarity measures the distance of two non-zero vectors. It has been used before in NLP tasks, for example to predict similarity between movies [FH03]. Since BERT has the CLS token whose output is a vector that serves as a aggregate sequence classifier, we can take the the output of the token on the query and on the document and measure the cosine similarity between the two.

To do this we need to use a separated input. After BERT gives its output, we take both vectors corresponding to the CLS token and calculate the cosine similarity between them. The cosine similarity is calculated as follows:

$$similarity = \frac{CLS_q \cdot CLS_d}{\|CLS_q\| \cdot \|CLS_d\|}$$

Where CLS_q and CLS_d are the vectors representing the CLS token for the query and the document respectively.

[11] https://huggingface.co/transformers/model_doc/bert.html#bertmodel under outputs, then *pooler_output*

The cosine of any given value can only be in the range of $[-1, 1]$ where we need values in the range of $[1, 3]$. Since both ranges are of equal size, we can solve this problem by simply taking the cosine similarity and adding 2.

Cosine similarity is a very intuitive concept, the lower the angle between two vectors, the closer they are in a space thus the more similar they are. Although usually the measurement is used with term frequency vectors [Hua08]. Cosine similarity provides as a simple method for getting an output from a separated input.

3.5.4 Multilayer Perceptron

A more complicated approach would be an ensemble containing a neural network. An ensemble takes the output of two separate models and gives a final prediction based on the two separate outputs. Any model that uses a seperated input is an ensemble. An ensemble can consist of any neural network or other method but for our purposes we will use a multilayer perceptron (MLP). To be more specific, we will use a MLP that takes a vector of length $768 * 2$ as input, this is two times the size of the CLS token in the of the base BERT model. As input the ensemble will take both CLS tokens from the output of the query and document from a BERT model. It will have two fully connected hidden layers, each decreasing the amounts of nodes in the layer. The first hidden layer has a size of 512 and the second layer has a size of 256. The output layer only has one node which when trained will contain the final output: a real value between 1 and 3. Between each hidden layer a ReLU activation function is applied. This is a very simple function:

$$f(x) = max(0, x)$$

As mentioned earlier needs a separated input. Now that there is a whole new neural network in our model we have to keep a couple of things in mind. This new model has entirely untrained weights. First we need to initiate the weights but when doing this we have to be careful not too initiate the BERT weights as well, as this would mean we would lose all of the knowledge it already has defeating the purpose of the model. To do this we pass a pre-trained BERT model to our ensemble after its weights are initiated.
Another point of attention is the learning rate. Earlier we specified that we do not want this to be too high since this might change the weights of the BERT model too much, and this is still very much true. However now we also have a entirely untrained neural network that does need acquire all of its knowledge from scratch. To make sure the new model can learn at a fast enough rate and that the BERT model doesn't lose its knowledge, we implement different learning rates for the different parts of our ensemble. The learning rate for BERT remains $2e^{-5}$, the learning rate for the new model is set to $2e^{-2}$.

The hope is that the untrained model will pick up on the intricacies of the data-set. Although BERT is modelled on a very large corpus, home appliances has a great deal of jargon. Brand names, specific materials and other challenging strings are all common in the data-set. A new network with a high learning rate might pick up on these specifics.

4 Results

We were able to reproduce our baseline result of an RMSE of 0.487 and will use this as the main comparison for our regression BERT models. Because the NSP model is a binary classification model, RMSE is a not a useful metric. Because of the different metric of performance and the simplified nature of the model, it will not be used for direct comparison between regression models.

	RMSE
CLS token	0.544
Cos similarity	0.537
MLP	0.540
Random Forest baseline	0.487
Train average baseline	0.536

Table 1: Table showing the RMSE scores on the test set of the different regression models. The random forest baseline is obtained as described in section 3.4. The train average baseline is obtained by guessing only the average relevance of the training set on the test set.

4.1 Next Sentence Prediction

Our first set of results is from the Next Sentence Prediction model with a simplified version of the problem. Due to this simplification, the results obtained from this experiment differ from the ones from other models. Most notably is the RMSE score on the test set, since this is given by Kaggle in the official competition as a late submission score. Although the predictions from this model could be submitted, the simplified nature of this experiment means that it would have a unrepresentative high RMSE.
The prediction/truth scatter graph is also omitted, since the model only has binary output this graph tells us no significant further information.

Another thing to note is that the metrics for measuring performance for this model also differ from the other models. Because RMSE works with distance from the truth it is better suited for regression. The simplification of the experiment means that the problem now works as a binary classification problem. As such a loss function that works better for such problems is used, namely cross entropy loss, which is also the value on the y-axis of the training performance graph. Just like with RMSE, a lower score means a better performance.
For evaluation a more intuitive flat accuracy metric is used, which is the percentage of correct answers the model has given in a given batch. This is because this particular experiment is to get a general feel as to if BERT can be useful given our data set. A flat accuracy provides a very easy look into its performance.
Because RMSE is not used in either graph, a line indicating the performance of the Random Forest benchmark is also not present.

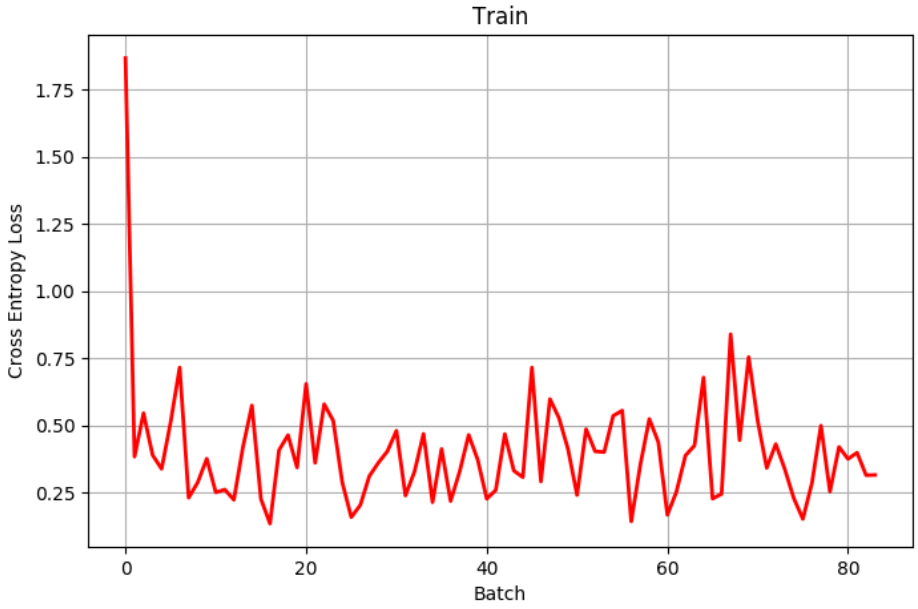

Figure 1: Cross Entropy loss per 100 batches in training

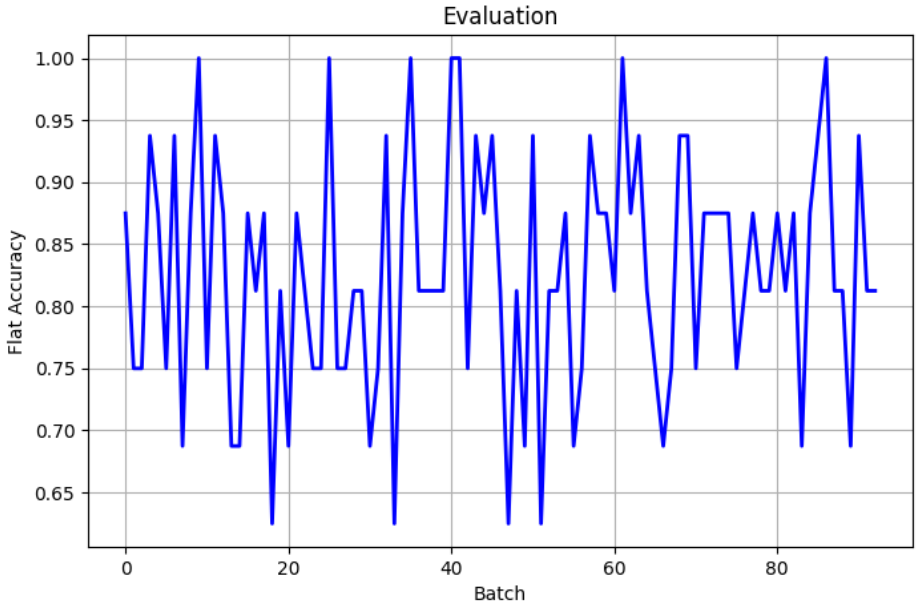

Figure 2: Flat accuracy per 10 batches in the evaluation process in training

These first results look promising. The model is quick to improve in training and is able to remain a cross entropy loss of around 0.4 fairly well. An increase in performance is less visible when looking at the flat accuracy in the evaluation in Figure: 2. This is to be somewhat expected as there is no substantial improvement to be seen in Figure: 1 after the first epoch either. The

first evaluation cycle comes after the first full epoch of the training cycles providing some explanation.

At a first glance a flat accuracy that barely drops below 70% is a promising first experiment but we have to take into account that the simplification of the problem comes with many caveats. When taking a look at figure 3, it is easy to see that only rarely a negative label occurs. This makes it very easy for the model to obtain a high accuracy by predicting a positive result way more than a negative result without taking the content of the data in mind.

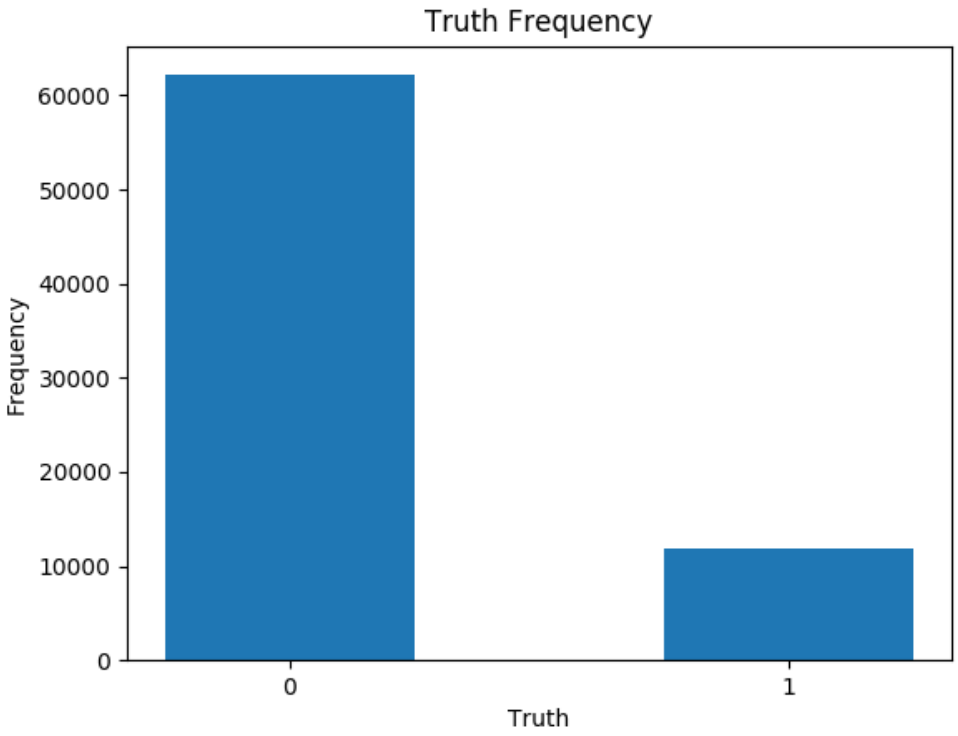

Figure 3: Amount of times labels occur in the train set

And when we look further into the amount of false positives in the predictions of the model in table 2, it is clear that this model isn't so much responding to the content of the query or document as it is repeating the most frequent label.

false positive	**true negative**
91%	9%

Table 2: Table showing the amount of false positives in percentages

4.2 Classification token

The format of the results from the classification token experiment are more inline with the other experiments. They include 2 graphs, one for performance during the training process, one for performance during evaluation in the training process and one scatter plot to compare predicted

values to their actual values.

With their being only one layer between the direct output of BERT and the final output, the hypothesis is that this model will not perform very well.

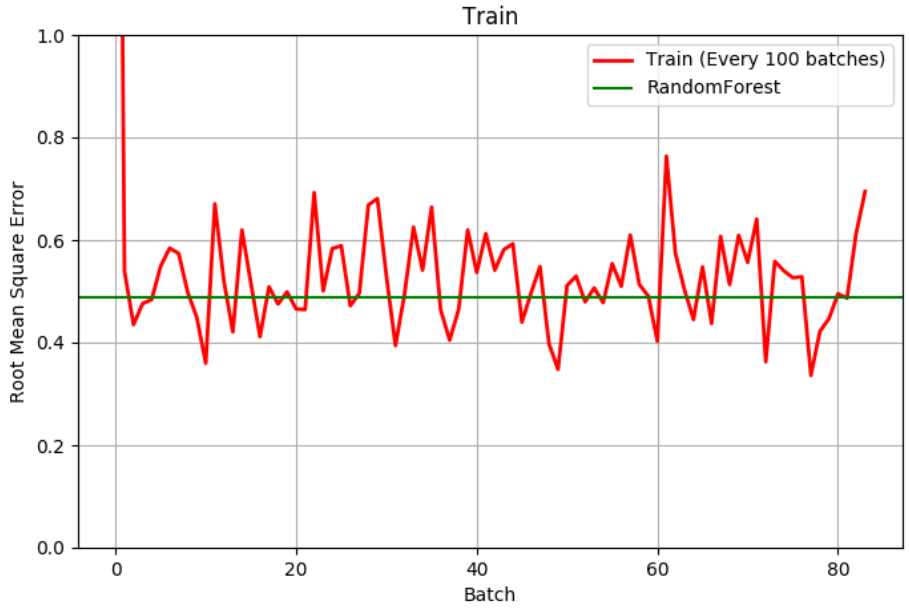

Figure 4: RMSE loss per 100 batches in training

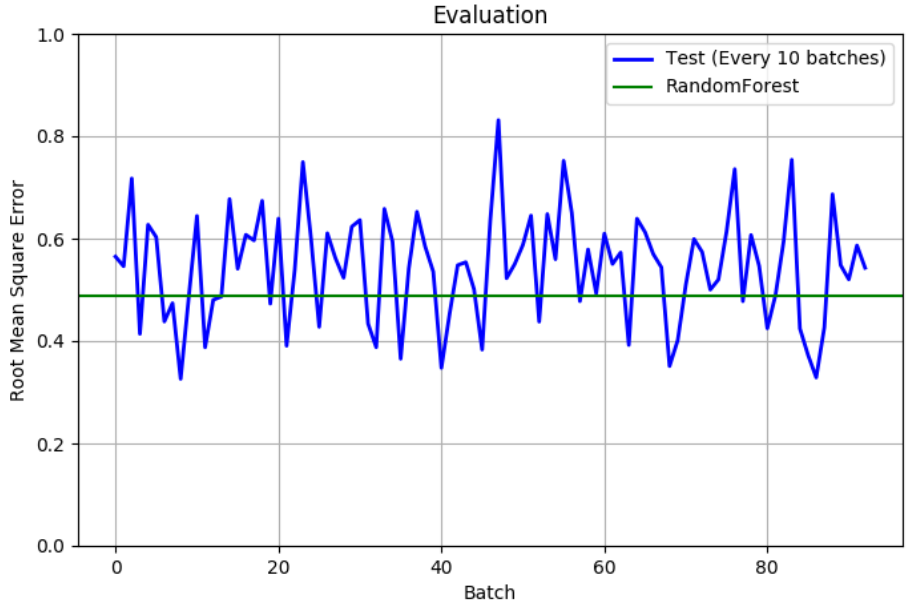

Figure 5: RMSE loss per 10 batches in evaluation in training

RMSE on test set: 0.544

The same symptom that shows in the NSP model seems to be present in this model. Although an initial gain in performance is clearly visible in figure 4, further improvement in either the training performance or in figure 5 cannot be found. Although the poor performance of this model was to be expected, the almost immediate stagnation was not.
If you look at the scatter plot in figure 6 that maps the predictions to the y-axis and the truths to the x-axis, you can clearly see that the model is not deviating from a value around 2.4 in it's prediction, which comes close to the average of the labels in the train set which is 2.3816.
Regardless, given the lack of complexity of the model an RMSE of 0.54 is not too bad, even though it is unable the beat a much simpler RandomForest approach.

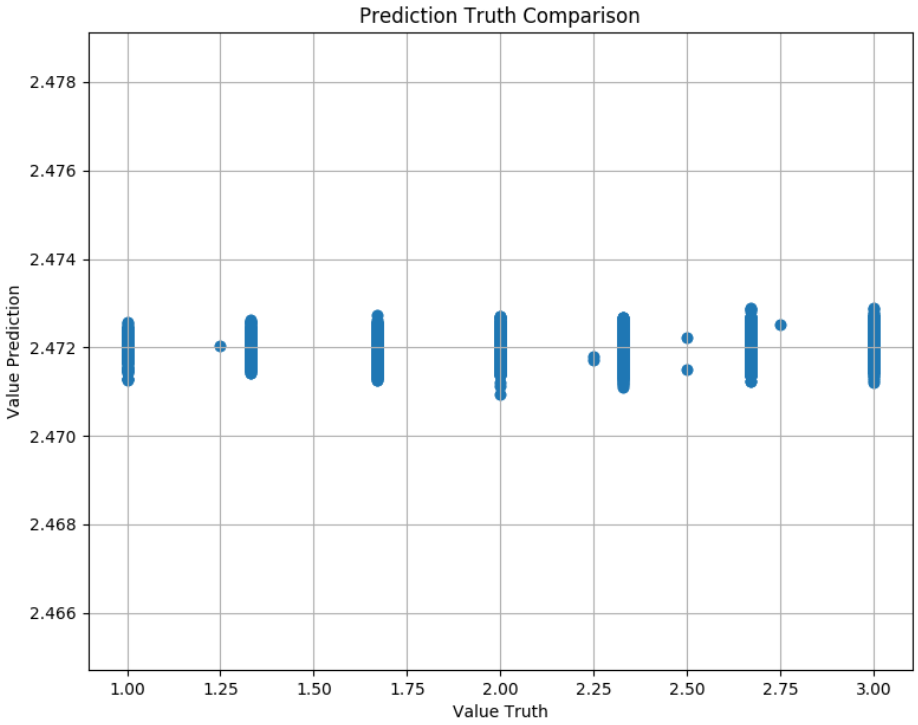

Figure 6: Scatter plot for the classification model with predictions on as y-values and truths as x-values. Predictions were taken from evaluation

4.3 Cosine Similarity

The next step up in complexity is the model using the separated input and takes the cosine similarity between the two CLS token outputs. There is no further neural network added to BERT. Also to be noted is that the same BERT model is used for the query and the document, so anything it learns from queries it also applies to documents and vise versa.

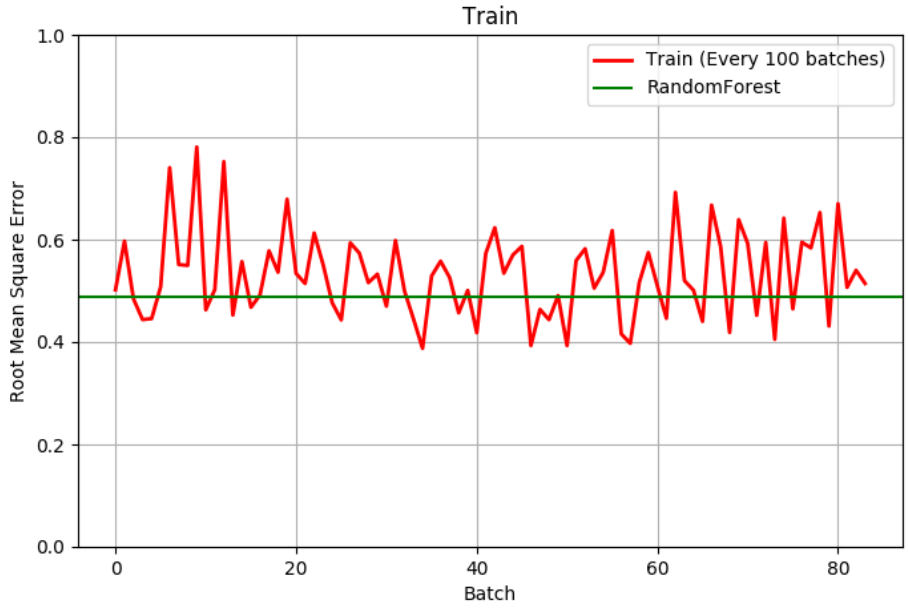

Figure 7: RMSE loss per 100 batches in training

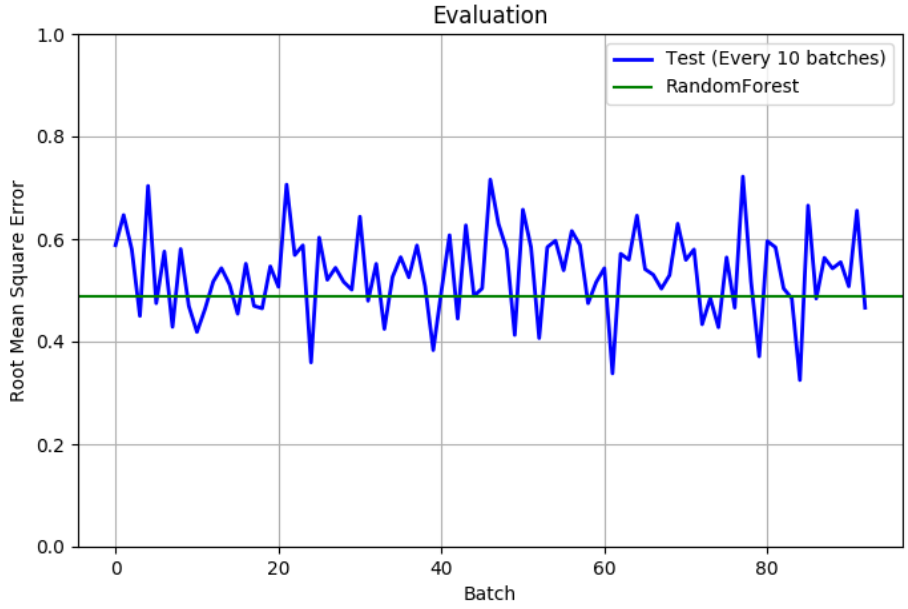

Figure 8: RMSE loss per 100 batches in training

RMSE on test set: 0.537

Again we see immediate stagnation in performance gains, both in training and in evaluation. It is remarkable that the the model starts out with such a low loss in training, around 0.5. Although

only one in every 100 batches is included as a data point, the initial batch should always be visible. It is of course not entirely impossible given that the RMSE of the first batch is near random given no previous training on the data set has occurred.
The scatter plot is also quite interesting. There is very little differentiation in the cosine similarity of the two cls tokens, meaning that the 'distance' of the two cls tokens remains mostly constant. This could be because the BERT model is again honing in on one value that gives a decent RMSE. Another explanation could be that the greater length of the document adds a somewhat constant distance from the query. Without further experiments it is impossible to say for sure what the underlying cause is.
We see another RMSE score that gets close but is unable to beat the RandomForest approach. Taking into consideration that the train time of this model is significantly longer because of having to run BERT twice per instance, once for the query once for the document, this result is somewhat disappointing.

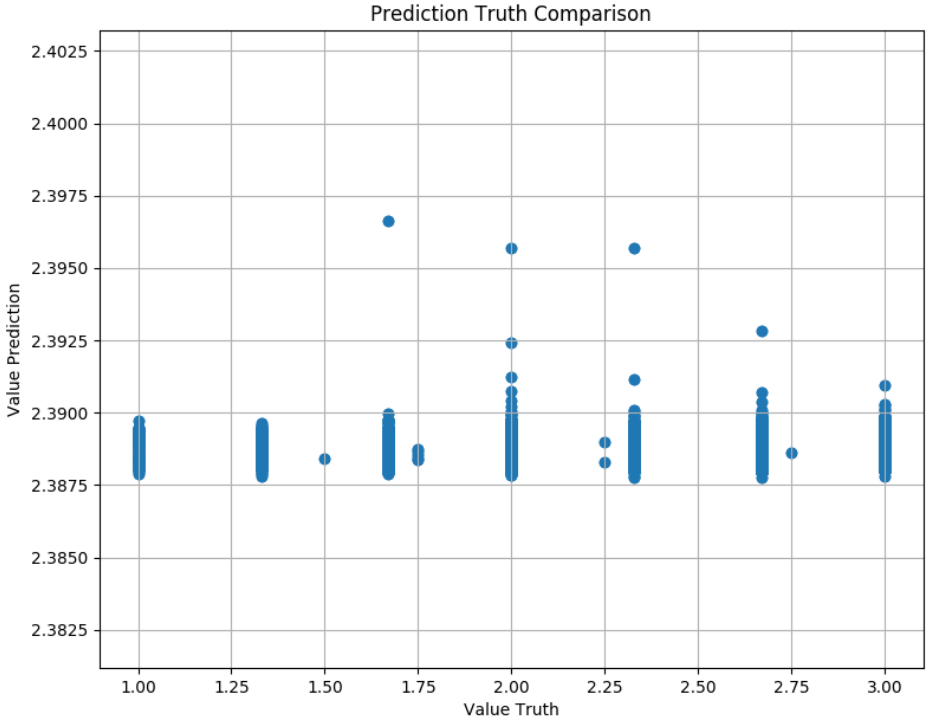

Figure 9: Scatter plot for the cosine similarity model with predictions on as y-values and truths as x-values. Predictions were taken from evaluation

4.4 Multilayer Perceptron

The final experiment in this thesis uses a BERT model with a multilayer perceptron with 2 hidden layers. This provides room for the model to pick up on the intricacies of the corpus at hand. As such, the hypothesis is that this model will outperform at least all other models that use BERT

and aren't working with a simplified version of the problem. Thus the hypothesis is that this model will outperform all other models included in this thesis.

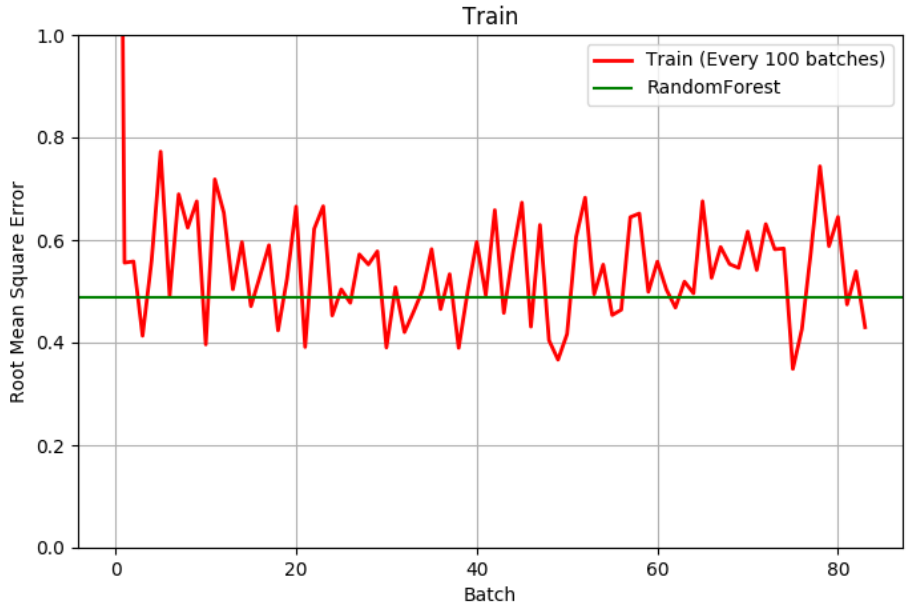

Figure 10: RMSE loss per 100 batches in training

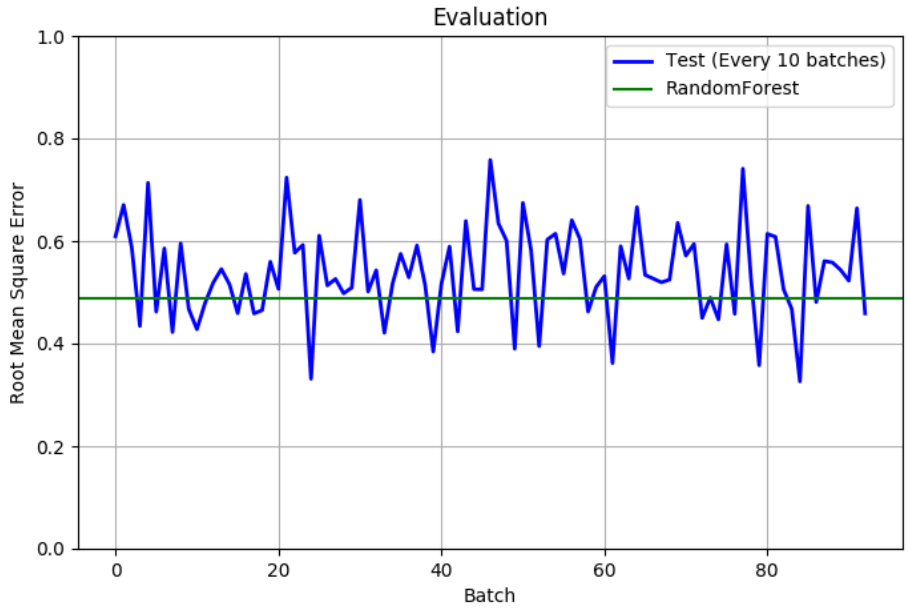

Figure 11: RMSE loss per 100 batches in training

RMSE on test set: 0.540

Compared to all previous results, these results provide little change. Again there is almost immediate stagnation in both training and evaluation cycles. It is curious, though not unique to this model, that the RMSE does drop below that of the random forest model occasionally, sometimes by quite a bit. It is however unable to maintain such a low RMSE.
The scatter plot is also more of the same, it deviates very little from one value. Nonetheless, the spread in predicted values is the biggest we have seen so far.
With an RMSE of 0.53972, this model is also unable to beat the RandomForest model. Given that this model has a whole other neural network as a head to its BERT model, this result is both unexpected and disappointing. Especially when taking into account the simplicity of the RandomForest model and its relatively very low training times.

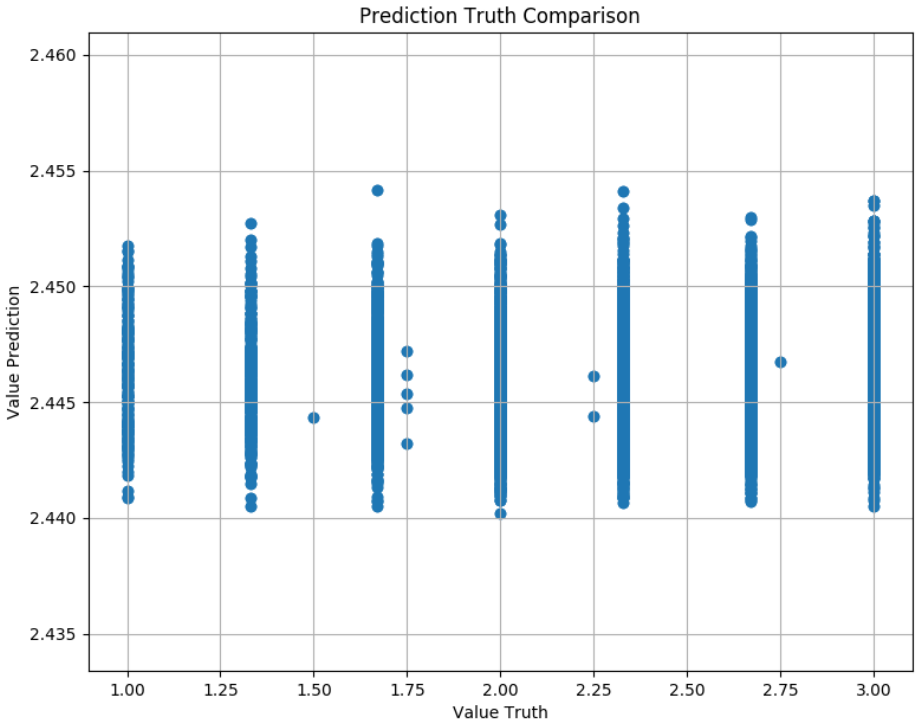

Figure 12: Scatter plot for the multilayer perceptron with predictions on as y-values and truths as x-values. Predictions were taken from evaluation

5 Discussion

5.1 Performances

The next sentence prediction model looks to be a promising start at first. The drop in training loss falls off very quickly and during the evaluation cycle there is little improvement at all, this is however not to be unexpected when performances early in training are as high as they are. Unfortunately when the seemingly good performance is dissected it quickly becomes clear that this it not quite the case. Even with the simplification of the problem by making it a binary classification task, it scores really bad when taking a look at false positives, missing 91% of the negative labels. This does not mean BERT cannot appropriately interact with the data set at hand. Mapping the real labels to binary values as done simply results in a huge imbalance in labels that represent relevance and labels that represent irrelevance. This makes it hard to say if BERT is a good fit for this task, which was the original goal of the experiment.
Not to imply that this experiment was for nought. The simplicity of this experiment provides a look into the workings of BERT and allows us to adjust our expectations for the results of other experiments.

This lack of simplicity becomes an issue when looking at the results of other experiments. Across all experiments we see almost immediate stagnation in performance after the first 100 batches. With the NSP model this is not too unexpected given that it immediately performs very well. The same cannot be said for models that work with regression and not with binary classification. These all hover around the same RMSE, both in training and in evaluation. The methods that were used to get a better look at the workings of the NSP model cannot be used here due to the fact that we are now facing a regression problem where there is no such thing as false positives. A scatter plot does provide some insight to the inner workings of our models.
Here too we see comparable behaviour between the regression models, all of them predicting the same value over and over with deviations of an order of magnitude as little as 10^{-3}. There are some outliers in both the cosine similarity and the MLP model but they are too few and they deviate too little to be of significance. So even though these scatter plots do provide additional information, it is still unclear as to why exactly these models perform this poorly.

Another unexpected result is how close the performances between all models are, with only a difference of 0.007 in RMSE between the best and worst performing model. As earlier stated in section 4.4, the expectation was that the MLP model would outperform all other models. Not only is it beaten by the cosine similarity model, the difference in RMSE on the test set between it and the CLS token model is also very small.
The cosine similarity model does not contain any new components that can learn in the same way a neural network can. Hence why it is surprising that it beats a model that has a neural network with a high learning rate specifically intended for picking up on extra details that other models could forget. It is possible that the BERT model at the heart of the cosine similarity model has learned to utilise its altered output. Although this is unlikely due to the low learning rate of BERT and it being specifically trained to respond to natural language.
Both the MLP and the CLS token model do contain such components. The CLS token model contains $768 + 1$ additional nodes of a neural network, 768 for each element of the CLS token

representation vector and 1 output node to represent the token as one real value. The MLP model contains a great deal more nodes: $768 * 2$ input nodes since it uses the separated input, $512 + 256$ nodes in hidden layer one and two respectively and again one output node. It could be possible that the initial output from BERT, before it reaches the next neural network, simply deviates too little for another network to pick up on the difference, something we will discuss in section 5.2. Regardless, with the vast amount of increase in room for additional knowledge, it is easy to see why it is surprising to see such a small improvement in performance. Especially given that the learning rate for the head of the MLP model was set much higher, meaning it should be able to get out of local minima easier.

5.2 Distribution of predictions

The scatter plots provide additional explanation for the poor performance of the models. Predictions are highly concentrated around one value, depending on the model this value differs slightly. The label seems to be unrelated to the prediction value. This could be due to a lack of correlation between the queries and documents in relevance, this is however unlikely as top scoring Kaggle entries use deep information retrieval methods. A more likely explanation would be that the model is unable to find find this correlation and that it is more optimal for the models to rely on a local minimum based on the proximity of the value of the predictions to the average of the labels of the data set, which is 2.3816.
This would offer an explanation why the ranking of the models was different than expected, with the cosine similarity model being the closest to predicting the average. The train-test split is always identical, but a random data sampler is used. Meaning that the order that the train data is supplied to the model does differ every time. This could result in slight variations in the final RMSE of the model. However it is strange that all of the models are approaching the mean, but none of the models are able to actually reach it given that this would get a better score.

Also interesting is the comparison of the scatter plots between the CLS token model and the MLP model. Even though the increase of the test score is very little, the difference between the highest and lowest prediction between the models is significant. The standard deviation of the CLS token model is 0.00025, whereas the MLP model has a standard deviation of 0.00222, making the standard deviation of the MLP model 8.88 times larger than its CLS token counterpart. This could imply that the MLP model was starting to react to the content of the data set or that the difference from the BERT outputs is simply too small.

All in all, no model was able to defeat the much simpler, much faster, RandomForest model. The exact reason for this is unclear. It could be that BERT is unfit for the data set but it is impossible to make such a conclusion without further research, as some results do hint that at least a MLP might end up working with further alterations. The proven wide applicability of BERT that can be found in many academic papers also make this conclusion implausible.
Since the model seems to respond very little to the content of the queries or the documents, if at all, another possibility could be that the problem lies within the data set. There are various properties of the data that suggest this. A lot of noise was already removed from the data as explained in section 3.3, this noise was fairly easy to remove but it could be that a lot harder to detect and remove noise still is in the data set. The data often doesn't resemble natural text as much as

the tasks BERT is used for in its original paper [DCLT19]. They contain a lot of brand names, measurements and acronyms/abbreviations. To get a grasp of what the data set looks like and the potential problems a deep information retrieval model faces, see the following product description taking from the data set. This description is taken from the data set before any sort of pre-processing.

"BEHR Premium Textured DECKOVER is an innovative solid color coating. It will bring your old, weathered wood or concrete back to life. The advanced 100% acrylic resin formula creates a durable coating for your tired and worn out deck, rejuvenating to a whole new look. For the best results, be sure to properly prepare the surface using other applicable BEHR products displayed above.California residents: see Proposition 65 informationRevives wood and composite decks, railings, porches and boat docks, also great for concrete pool decks, patios and sidewalks100% acrylic solid color coatingResists cracking and peeling and conceals splinters and cracks up to 1/4 in.Provides a durable, mildew resistant finishCovers up to 75 sq. ft. in 2 coats per gallonCreates a textured, slip-resistant finishFor best results, prepare with the appropriate BEHR product for your wood or concrete surfaceActual paint colors may vary from on-screen and printer representationsColors available to be tinted in most storesOnline Price includes Paint Care fee in the following states: CA, CO, CT, ME, MN, OR, RI, VT."

This example contains many of the aforementioned challenges. There are many instances of concatenated words that should be separate, which is not later fixed in pre-processing. Measurements are also present in a form that does not closely resemble natural text, the same can be said for brand names and abbreviations. These are all things that could potentially impact BERT's performance. The product title provides little extra information and the product attributes are even further from natural text, as discussed in section 3.3. All of these problems are potential explanations BERT's poor performance.

5.3 Future work

As there is still no definitive answer to our question of the usefulness of BERT for this data set or in e-commerce in general, much more research can still be done. Since all of the models have a very small range between their lowest and highest prediction, it might be useful to multiply the output of BERT (so before it goes to the head of the model) by some factor. The range might be very small but there still is a range. If the problem was that the outputs from BERT differ too little from each other for the heads, simply multiplying these outputs might solve the problem.

Another possible solution to the problem of the small range would be to increase the size of the head in the MLP model. Compared to the range of the CLS token model, there is a significant increase in the range of the predictions for the MLP model. It might be that adding a 2 hidden layer neural network head was a step in the right direction, but that the step wasn't big enough. Another seemingly obvious solution would be fine tuning the learning rate of the head of the MLP model. This was done for this thesis but due to the lack of improvement in performance (being either identical or worse) the results of these experiments were omitted.

When looking at the overall strategy of the models we see that it repeatedly predicts a value close to the average of the training set. This might be because there is a considerable imbalance between

the labels. In section 4.1 we can see that when simplified there aren't many query document pairs that are labelled irrelevant. This could be problematic for all models. To combat this imbalance the data set could be 'resampled' by for example omitting relevant pairs or duplicating irrelevant pairs[12].

If the problem does not lie with the small range of the base BERT model, it would be interesting to check the relation between the length of the document and the output of the model. This could be done by taking the cosine similarity model and a data set where the the contents of the documents are similar but the length differs greatly. Due to nearly all outputs being longer than 200 tokens, most of them are truncated meaning they are all eqaul to the max length of 200 tokens. The queries are also often quite short, so the difference in length for queries is also relatively small. possibly leading to the low range in the cosine similarity model.

It could also be that the problem is bigger than BERT and that only using a deep information retrieval approach without additional models is not the right fit for the data set. Many more deep IR models could be tested on the data set, for a possible candidates I recommend looking at Matchzoo [GFJC19].

5.3.1 Viability in e-commerce

One could question if BERT is a viable strategy in e-commerce at all. The Home Depot data set does contain it challenges but it in some ways it is also representative of the field. When looking at most online shops a similar collection of natural text data can be found, a concise product title, a product description that resembles natural text the most and a list of product attributes. It could be that this structure of data just doesn't work with BERT.
With our current information, it is still impossible to make such a conclusion though. Besides all previously stated possible solutions, the problem could also lie with the content of the data set. It does contain a lot of noise and home improvement specific terms. More pre-processing could still be done on the data, not only removing more noise but also removing sale specifics that have nothing to do with the product, like state abbreviations. And even though data sets for e-commerce are hard to find due to the competitive nature of the field, it would be very interesting to see how the model performs on data from online shops that have a different catalogue. Product descriptions about clothing for example often contain much less jargon.

[12] https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets

6 Conclusion

In total three different models that are based on regression were tested that all had Google's BERT model as a base model to work of. The goal was to score a lower RMSE than a simple RandomForest model to see if BERT could be useful in e-commerce. Unfortunately, none of the BERT models were able to beat the RandomForest models. Not only that but the ranking between the BERT models was also unexpected, with the cosine similarity model outperforming a more complicated multilayer perceptron model.

With the results from the performed experiments, it is impossible to conclude why the models failed to beat the RandomForest models or why the ranking between themselves is what it is. As such it is also impossible to answer if BERT could be useful in e-commerce, further research is still required.

When looking at scatter plots of the models we see that the range of the predictions of all three models is very small. When trying to improve the performance of the models, we would suggest trying to increase this range first. Possible approaches to achieve an increased range could be multiplying the output of BERT by some factor, adding more layers to a MLP head of BERT or resampling the data set.

References

[CHH+07] Philipp Cimiano, Peter Haase, Matthias Herold, Matthias Mantel, and Paul Buitelaar. Lexonto: A model for ontology lexicons for ontology-based nlp. In *Proceedings of the OntoLex07 Workshop held in conjunction with ISWC'07*, 2007.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[FH03] Michael Fleischman and Eduard Hovy. Recommendations without user preferences: a natural language processing approach. In *IUI*, volume 3, pages 242–244. Citeseer, 2003.

[GFJC19] Jiafeng Guo, Yixing Fan, Xiang Ji, and Xueqi Cheng. Matchzoo: A learning, practicing, and developing system for neural text matching. In *Proceedings of the 42Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, pages 1297–1300, New York, NY, USA, 2019. ACM.

[HHG+13] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. ACM International Conference on Information and Knowledge Management (CIKM), October 2013.

[Hua08] Anna Huang. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, volume 4, pages 9–56, 2008.

[KSSZ17] Shubhra Kanti Karmaker Santu, Parikshit Sondhi, and ChengXiang Zhai. On application of learning to rank for e-commerce search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 475–484, New York, NY, USA, 2017. Association for Computing Machinery.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[PNI+18] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[QLXL10] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13(4):346–374, 2010.

[SB88] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[SDCW19] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[SWHL06] Weifeng Su, Jiying Wang, Qiong Huang, and Fred Lochovsky. Query result ranking over e-commerce web databases. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, page 575–584, New York, NY, USA, 2006. Association for Computing Machinery.