Using Machine Learning to Detect if Two Products Are the Same

A master thesis from

Bc. Jung Peter

Faculty of Electrical Engineering
Czech Technical University in Prague
Open Informatics, Artificial intelligence
Czech Republic
2020

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Jung Peter** Personal ID number: **486798**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence**

II. Master's thesis details

Master's thesis title in English:

Using Machine Learning to Detect if Two Products Are the Same

Master's thesis title in Czech:

Využití strojového učení pro detekování, kdy jsou dva produkty stejné

Guidelines:

The main goal of this work is to design a prototype of a system for predicting when two products (e.g. mobile phones, vacuum cleaners etc.) are the same product based on their textual description. The main research question is to evaluate feasibility of using character-level and word-level word embeddings for this task.
1. Design a system for predicting when textual descriptions of two products refer to the same product. Use insights from [2].
2. Empirically evaluate feasibility of models based on word embeddings for this problem.
3. Compare performance of character-level (e.g. [4]) and word-level word embeddings (e.g. [5]) for this task empirically. Discuss the results.

Bibliography / sources:

[1] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.
[2] More, A. (2017x). Product Matching in eCommerce using deep learning. Available online: https://medium.com/walmartlabs/product-matching-in-ecommerce-4f19b6aebaca
[3] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).
[4] Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2016, March). Character-aware neural language models. In Thirtieth AAAI Conference on Artificial Intelligence.

Name and workplace of master's thesis supervisor:

Ing. Ondřej Kuželka, Ph.D., Intelligent Data Analysis, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **04.02.2020** Deadline for master's thesis submission: **22.05.2020**

Assignment valid until: **30.09.2021**

Ing. Ondřej Kuželka, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

AFFIDAVIT

I declare that I am the sole author of this diploma thesis on the "Using machine learning to detect if two products are the same" using the literature and sources mentioned, and with my supervisor's great help.

Date, city and signature

THANKS

I would like to thank my supervisor, for accepting the idea of this work in the very beginning and his later professional guidance in terms of machine learning and academic writing.

Not smaller thank also belong to my professors during the last two years, who taught me artificial intelligence in general.

I also have to mention my work colleagues, for giving me access to the company databases, guiding me about their usage and computational power of their servers.

Contents

1 ANNOTATION

1.1 ENGLISH

In this work, we investigate ways to use machine learning in the e-commerce field, with an application for the problem of pairing different descriptions of the same product from various online shops. Even though we evaluate the methods developed in this thesis only on this problem, they could be used in various areas. In addition, we create a new REST API and use it to evaluate our model on real-world datasets. Specifically, we apply our methods for finding duplicates in an existing online catalog aggregating items from hundreds of e-shops.

1.2 SLOVAK

V tejto práci sa zamieravame na možnosti využitia strojového učenia v oblasti e-commerce. S konkrétnym využitím pre párovanie produktov a ich ponúk od roznych obchodov. Aj ked všetky metódy budú optimalizované pre toto použitie, ich techniky sa mozu neskor využiť aj na iné oblasti, ako napríklad obohacovanie katalógu produktov o nové parametre pre produkty alebo pokročilé formy vyhľadávania. V závere využijeme naprogramované REST API, ktoré využíva náš model, na evaluáciu nad reálnymi problémami, ktoré postihujú dnešné online katalógy produktov. A to zamezenie duplicitám a zle napárovaných ponúk od obchodov k produktom.

2 INTRODUCTION

E-commerce is one of the fastest-growing businesses in the world. The Czech Republic is the fastest-growing e-commerce market in Europe. It is predicted that the online retail industry will grow by 16 percent in the Czech Republic between now and 2021 [45]. The number of eshops in the Czech Republic alone is about 40907, while new ones are appearing every day. The market with e-commerce in the Czech Republic is at the time of writing worth 4.4 billion euros [12]. With so many e-shops on the internet, it is difficult to find trustworthy ones and even harder to find the best available price for the specific product. That is why online price comparison sites like Heureka, Amazon, or Walmart are doing so well.

However, with so much data (products and offers), it is becoming impossible for humans to match all offers to their corresponding products manually, e.g., to match all the different offers from hundreds of e-shops for iPhone 8, 64GB to the product iPhone 8, 64GB. Offers are being left unmatched (and thus hard to find), or mistakes are made, which can result in seeing a calculator in a product detail of our dog's favorite food.

In this diploma thesis, we propose a solution for product matching problem consisting of several components which should provide accurate product-offer matching gateway. Namely, deep learning will be used as a state-of-art technique for text processing, providing similarity measure between two titles of products, and other simpler signals will be used to help with the decision in cases of uncertainty.

Because of stated facts, it is hard to obtain enough training data for our applications, so we will try to generate as much as possible reasonable data for our use cases. Also, during our work, we were able to contribute to the open-source community, mainly the TensorFlow library, and we tested our solution at the real-world problem in the company. The scientific contribution will be mainly in terms of different text embedding and neural network architectures evaluation at this problem because best to our knowledge, there are not many papers describing this. This work can also be used as a starting point for many companies that consider applying machine learning solutions to their use cases.

3 BACKGROUND

3.1 EMBEDDING

In the Natural Language Processing field (or NLP), word embedding is currently a state-of-the-art technique to represent words and sentences as vectors of real numbers, where vocabulary is created to convert between string and vector representations. It involves the calculation of embedding from high dimensional space of human language to a continuous vector space with a much lower dimension.

Methods to generate this mapping include neural networks, dimensionality reduction on the word co-occurrence matrix, probabilistic models, and explicit representation in terms of the context in which words appear [41] [3] [38].

3.1.1 WORD-LEVEL

Word-level embedding is a standard approach in the NLP literature. The most popular ones are probably GloVe [40] or Word2Vec [41]. Word embedding can be usually trained by two methods, skip-gram or CBOW. The skip-gram model is meant to predict the context of the word, where CBOW is designed to predict words by the context. To better illustrate this, let's have the title "Samsung Galaxy S10e black". The skip-gram would take the word "Samsung" and it would try to predict the words "Galaxy", "S10e", "black." Where CBOW would take the words "Galaxy", "S10e", "black" and it would try to predict the word "Samsung".
However, world-level methods come with one significant disadvantage - Out Of Vocabulary words.

For example, if one shop names a product offer "Kibbles'n Bits Dog Food" and another shop names the same product, "Kibbles Bits Dog Food", the main characteristic of the product, its brand, would be lost. In this simple example, it could be prevented by advanced tokenization. However, not all situations can be prevented correctly in such a simple way.

3.1.2 CHARACTER AND SUB-WORD LEVEL

Character level and sub-word level embedding methods are not learning representations of whole words but their subparts and calculating word embedding based on them. The final output of the character embedding step is similar to the output of the word embedding step. However, handling of OOV words is much better than just assigning random vector or using a single <OOV> vector. Popular methods are FastText [21], CNN-LSTM neural networks [34] or Char2Vec [5].

Where instead of the whole word "Kibbles," we learn embedding for its characters "k-i-b-b-l-l-e-s" or subwords, for example, "ki-bb-les." Then, if a misspelled word or word in another form like "Kibblesn" would come, the dominant part of its embedding will be meaningful, and thus still providing a good base for the network.

In this work, we test empirically if character-level embeddings are better suited for the problem of product matching based. We show that models based on character-level word embedding generalize much better to unseen data.

3.1.3 DIMENSIONS

The usual dimension of embedding in literature is somewhere between 100 to 300 floats per word. While higher dimensions could theoretically capture more information, there is also more data needed to learn them. So choosing the proper dimension is an empirical task. We will investigate several dimensions and report their impact on final accuracy.

3.2 NEURAL NETWORKS

The first neural network was conceived by Warren McCulloch and Walter Pitts in 1943. Later in 1975, Kunihiko Fukushima developed first truly multilayered network [14]. Recently, by the availability of high-performance servers and big data, neural networks start to excel in fields like computer vision, speech recognition, machine translation, or text processing. Many great successes were achieved by various implementations of neural networks on real-world use cases, like Facebook's face recognition [48], autonomous driving [16], or Google's translation service [15] and searches.

3.3 CONVOLUTIONAL NEURAL NETWORKS

Convolution Neural Networks are a class of neural networks, most commonly applied to images. Convolution layer has a small matrix of weights that shifts over input data, calculating outputs. Because of that, they are also known as shift invariant or space invariant artificial neural networks. They have been successfully used in fields like image and video recognition, image classification, or natural language processing [17]. By having much fewer parameters to learn, in contrast to densely connected layers, they are much faster to train and less prone to over-fitting on training data. The 2D convolution operation is visualized in figure 1.

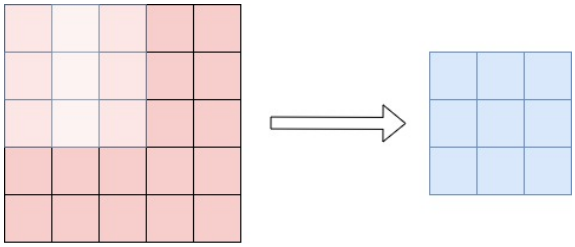

Figure 1: 2D convolution with one channel

3.4 RECURRENT NEURAL NETWORKS

In contrast with pure feedforward networks, Recurrent neural networks (RNN) are suitable for inputs of various lengths. They use their internal state to store previously seen inputs and take them into account when predicting output at the current timestamp. This can be seen as "learning" in the inference phase. Where vanilla feedforward neural network learns parameters in the learning phase and then uses them to predict output, RNN network updates its hidden weights at the inference phase too. Due to this, they were successfully used in tasks like handwritten text recognition [2] or speech recognition [37].

RNN captures long-range dependencies between tokens in a sequence. Long Short Term Memory Networks (LSTM) was developed to address the vanishing gradient problems of RNN. A basic LSTM cell consists of various gates to control the flow of information through the LSTM neurons. LSTM is suitable for sequence tagging or classification tasks where it is insensitive to the gap length between tags, unlike vanilla RNN or Hidden Markov Models (HMM).

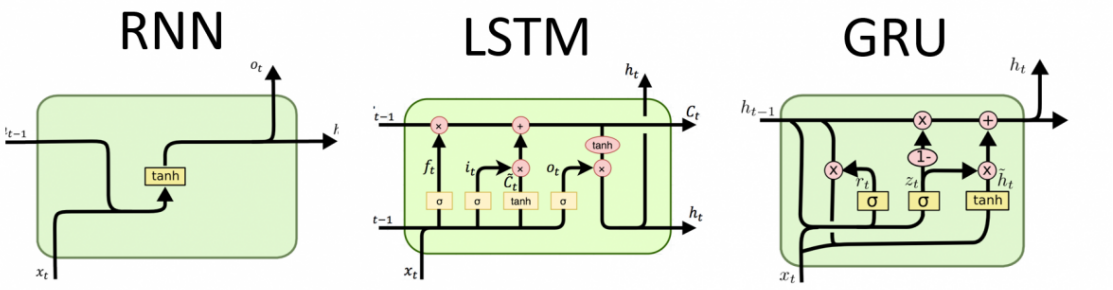

Figure 2: Different RNN units

Given an input e_t, an LSTM cell performs various non-linear transformations to generate a hidden vector state h_t. GRU, or Gated recurrent unit, is similar to the LSTM but lacks an output gate. GRU has fewer parameters than LSTM, so they are faster to train. However, they are usually outperformed by LSTM, due to the more complicated logic in the LSTM. All variants are shown in figure 2.

3.5 SIAMESE NEURAL NETWORKS

A twin neural network (or a Siamese Network) is an artificial neural network that uses shared weights while working in parallel on two different inputs to compute comparable output vectors [46] [7] [52], this is shown in figure 3. A twin network might be used in things as verification of handwritten signatures [9], face recognition [10], matching queries [54] or image recognition [63]. Siamese networks work very well in learning the similarity between inputs.

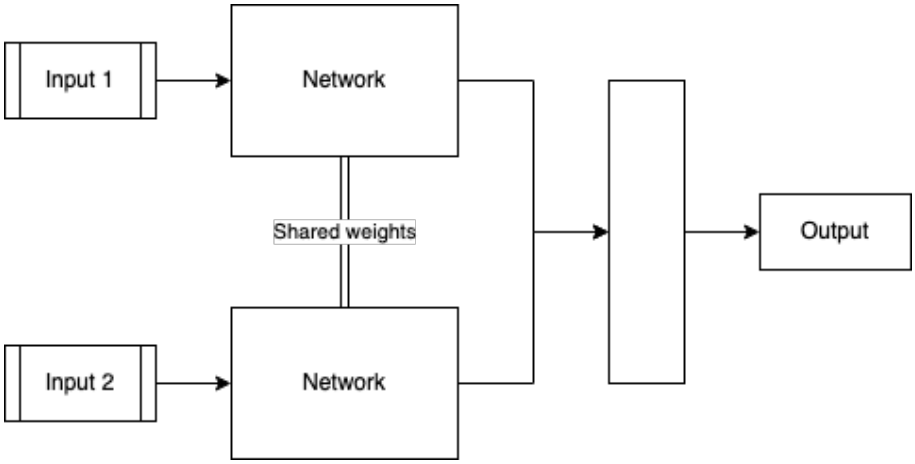

Figure 3: Siamese architecture

3.6 GRADIENT BOOSTING TECHNIQUES

Gradient boosting is another kind of machine learning techniques that can be used for regression or classification. Its prediction is based on an ensemble of several prediction models, for example, decision trees [58]. Similarly to deep learning, they allow optimizing arbitrary differentiable loss function.

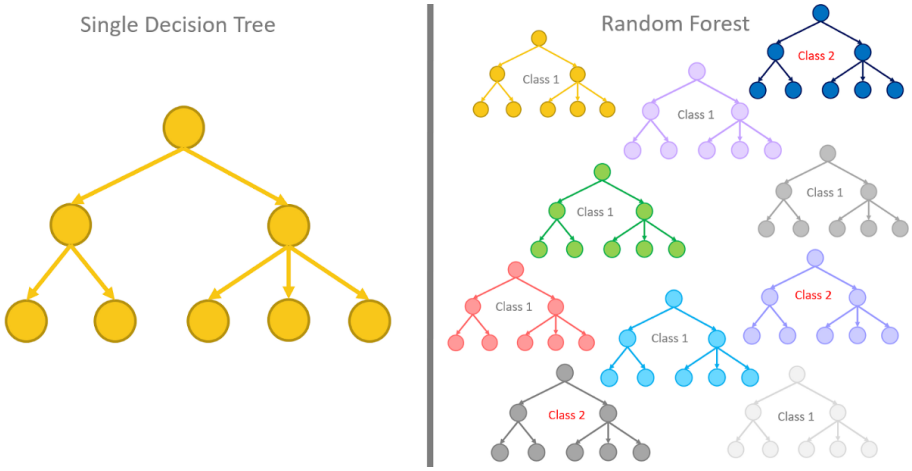

Figure 4: Random forest

4 PROBLEM DESCRIPTION

4.1 PROBLEM STATEMENT

As a product aggregator, we are continually receiving product offers that e-shops would like to put on our website. These offers can be either new products that we do not have yet or they can be offers for an existing product, e.g., already contained in the database from another e-shop. With big data flow of incoming offers, it is unrealistic to decide it in real-time, or even with days of delay, by humans. To successfully handle it, we need to have some kind of automatic system.

More technically, in our database, we have a table with more than 150 000 000 rows representing various products. In addition, we have the second table with more than 650 000 000 rows representing e-shops offers for those products. It would be incomputable to compare offers with all those products. So given incoming offer o_i, we firstly need to efficiently find a set of N product candidates

C. Afterward, evaluation of each pair o_i, c_j is made, and a product with the highest probability is chosen. If there is no product with enough probability of a match, a new product is created, or offer is delegated for manual review in case of uncertainty. Because it is hard to obtain enough valid training data, the generation of training pairs is needed based on the current database. This can also lead to contradictions in datasets, as it would be impossible to label everything manually. The evaluation will be made on two real-life problems, firstly elimination of product duplicates in the catalog and secondly elimination of mismatches. Those tasks will be performed on smaller categories of the product catalog and manually verified by humans. The primary signal of product offer matching will be similarity measurement based on their titles using deep learning. In addition to this, prices and attributes will be used as helpers in cases of uncertainty because, as we later show, a lot of products can not be matched only using their titles even by humans.

4.2 PREVIOUS WORK

With big e-commerce companies like Amazon or Walmart, we can assume that many undisclosed solutions exist. As product matching is mainly text and image processing tasks, some of them will probably use deep learning. However, only a few publications on this topic exists available online [1] [6] [65].

First, a simple solution that is usually deployed in practice is an exact match in the title or unique identifiers like EAN codes. However, based on our experience based on an analysis of a large commercial database, there is no standard form of titles, and checking for the exact match cannot group more than a few percent of product offers that describe the same product. A possible improvement of this simple strategy is to preprocess the titles, e.g., by tokenizing them and representing them as a bag of words. Offers classified in this manner are usually near 100% correct, but the total amount of matched offers is under 3-4%. This also heavily depends on the target category, where smartphones like "Samsung Galaxy S10e" have unique names, dog food like "Royal Canin beef in sauce 5kg" can have many variations and synonyms, resulting in an even lower number of matched offers. Goods like books come with unique identification attributes, which are often correct, but this is not the case for most of the other product categories. That is why EAN codes can sometimes be useful, but we certainly can not rely on them.

Walmart [1] describes an approach to product matching based on the combination of the following sources of information: title, image, description, and price of the product and offer. They used Concat CNN as architecture for title similarity, which did not perform in this work as well as they claimed, this can be for several reasons we will discuss later. Nevertheless, Walmart's article was the main source of information and inspiration in this work. Next, Cimri [6] presents a slightly more robust architecture resulting from Walmarts one. We do not test this architecture on our datasets because it is a more robust version of Walmarts one

Other signals that can be used to eliminate non-matching offers are their attributes. However, they are usually not supplied by eshops very trustworthy. Walmart has a good paper about its extraction from titles [44] using deep learning and sequence labeling techniques. Similarly, perhaps more robust, the technique is described by Amazon [65]. Next big players in commerce who experiments with machine learning product matching are Yahoo [62], ASOS, and Zalando [20].

5 IN ADDITION TO MACHINE LEARNING

5.1 FINDING PRODUCT CANDIDATES FOR INCOMING OFFERS

As product catalogs in today's e-commerce systems are typically huge, comparing incoming offers to each product would be computationally unrealistic. That is why we need some smart method to pre-select a set of products from the catalogs that are possible matches.

By utilizing word embedding as an input form for neural networks, we can compute sentence embedding of fixed length, which can be used for searching in the space of dense vectors, as is

shown in figure 5. To do this efficiently, we utilize the Facebook Faiss library. With FAISS, we can even search in big spaces that possibly do not fit in RAM. This will be not required at the moment, but after the first phase of development is done, and we will index all products from our database it is possible that we will need to take this into account.

Given a set of vectors x_i (embeddings) of some dimension d, we build a data structure in RAM that can be used for efficient vector search. After that, we can efficiently compute the embedding of incoming offers title and efficiently search for its product candidates by computing:

$$i = \min_i ||x - x_i|| \tag{1}$$

Where $||.||$ is the Euclidean distance because we already represent titles as vectors (embedding), all that is left to do is create a matrix of dimension $|P| * D$, where $|P|$ is a count of all products in the database and D is their vector representation dimension. Then, FAISS allows us to easily query for similar vectors and their indexes, which can be mapped to the original products.

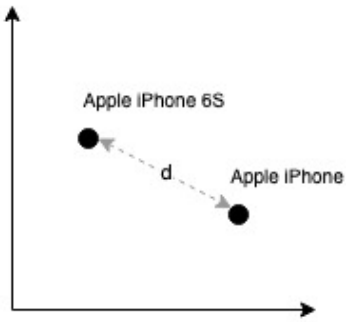

Figure 5: Baseline similarity using euclidean distance of embedding

5.2 PRICE OUTLIERS

An outlier is a data point, in our case price of the incoming offer, that lies outside the overall distribution of prices of offers already matched with the product. For example, if we compare prices of Royal Canin 1kg with incoming Royal Canin 10kg, we can except that it will be around ten times more expensive. That should be detected as an outlier. There are many ways to detect outliers, and we will discuss several of them because each gives a bit different results in different situations. We evaluate outlier tests only if the incoming price is lower or higher then minimum or a maximum of currents prices, respectively.

5.2.1 GRUBBS'S TEST

Grubbs test was published in 1950 by Frank E. Grubbs. It is used to detect outliers in a univariate data set assumed to come from a normally distributed population [61]. Grubbs comes with one and two sided variation and it is defined for following hypothesis:

H_0: There are no outliers in the data set
H_a: There is exactly one outlier in the data set

The test statistic is defined as

$$G = \frac{\max_{i=1\ldots N} |Y_i - Y|}{s} \tag{2}$$

And then the hypothesis of no outliers is rejected at significance level α if

$$G > \frac{N-1}{\sqrt{N}} \sqrt{\frac{t^2_{\alpha/(2N),N-2}}{N-2+t^2_{\alpha/(2N),N-2}}} \tag{3}$$

with $t^2_{\alpha/(2N),N-2}$ denoting the upper critical value of the t-distribution with N - 2 degrees of freedom and a significance level of a $\frac{\alpha}{2N}$.

5.2.2 BARTLETT'S TEST

Barlett test decides if k samples are from populations with equal variances. It is used to test the null hypothesis, H_0 that all variances of population are equal against the second option that at least two are different. In our case, we will have only two populations, one with old prices and one with new price added [59].

If there are k samples with sizes n_i and sample variances S_i^2 then Bartlett's test statistic is

$$X^2 = \frac{(N-k)\ln S_p^2 - \sum_{i=1}^{k}(n_i-1)\ln S_i^2}{1+\frac{1}{3(k-1)}(\sum_{i=1}^{k}\frac{1}{n_i-1}-\frac{1}{N-k})} \tag{4}$$

where

$$N = \sum_{i=1}^{k} n_i \tag{5}$$

$$S_p^2 = \frac{1}{N-k}\sum_{i}(n_i-1)S_i^2 \tag{6}$$

The null hypothesis is then rejected if $X^2 > X^2_{k-1,\alpha}$, where $X^2_{k-1,\alpha}$ is the upper tail critical value for the X^2_{k-1}.

5.2.3 INTERQUARTILE RANGES

It is a measure of statistical dispersion. Equal to the difference between 75th and 25th percentiles, defined as

$$IQR = Q_3 - Q_1 \tag{7}$$

We use IQR to decide if incoming price X is the outlier, if either.

$$X < Q_1 - c * IQR or X > Q_3 + c * IQR \tag{8}$$

Where c is empirically chosen constant, based on observations.

5.2.4 DIXON'S Q TEST

Alternatively, a Q test is used to test if the given value is an outlier. To test this, data is sorted in increasing order, and Q is calculated as

$$Q = \frac{gap}{range} \tag{9}$$

Where the gap is the absolute difference between the outlier in question and the closest number to it. Range if the difference between the maximal and minimal value in data. Incoming price is rejected if

$$Q > Q_{table} \tag{10}$$

Where Q_{table} is a reference value corresponding to the sample size and confidence level. Dixon's Q table is available at [60].

5.2.5 RATIO THRESHOLD

All tests above assume that we already have enough data, e.g., matched offers with prices, that we can test our hypothesis against. In the case of the lonely offer, we do a simple threshold ratio test defined as

$$V = \frac{\max(old, incoming)}{\min(old, incoming)} \tag{11}$$

and reject the offer if

$$V > Constant \tag{12}$$

Where constant is empirically chosen value based on observations.

5.2.6 TESTS

Every test gives slightly different outcomes, and there are even situations when some of them can not be used, e.g., because of an insufficient amount of data points. The results of different tests on different datasets are shown in the 1 table below.

Table 1: Price outlier detection

Data	New value	Grubbs	Barlett	Inter quartile	Dixon	Ratio
1299, 1349, 1259, 1289	1229	S	S	S	S	S
1480, 1349, 1590, 1450	1129	O	S	O	S	S
1299	1800	U	U	U	U	O

Where O is denoting Outlier, S standard, and U unknown because the test can not be applied. The selection of tests is a pure empirical task. Test using the ratio threshold described in section 5.2.5 is required in cases where the product has only one or two offers, e.g., data points or prices. Others are used to provide more signals, and the final decision is made on the majority among them, or they can be processed by xgboost model, as will be described in section 15.

5.3 ATTRIBUTES EXTRACTED FROM TITLE

Attributes are a strong signal for (dis)matching of offers. For example, if we know that the brand of product is Royal Canin, but the brand of the incoming offer is DM, then we can dismiss them with almost 100% certainty. However, shops usually don't provide by computer easily interpreted list of attributes. They are often misspelled, shortened, or missing at all.

We thus keep the list of all known attributes and check against them in the titles. The attribute is the property of the product, like its brand or flavor. The definition of attributes consists of two steps. First, extract them automatically from e-shops that provide them. The second one, manually check them, remove duplicates, or add new ones based on today's interest.

5.3.1 FIRST VERSION OF ATTRIBUTE CHECKING, TOKENIZED EXACT MATCH

In the first version, we created two sets of n-grams. One for the product title and second for the offer titles. Then, there is an n-gram to value to attribute names mapping table, which maps n-grams to the all possible attributes, if any. Lastly, if two titles contain contradictive attributes, they are mismatched. Contradictive attribute means that if two products contain them, they naturally can not be the same. Such an attribute is, for example, the brand of a product. The visualization of this process is shown in figure 6.

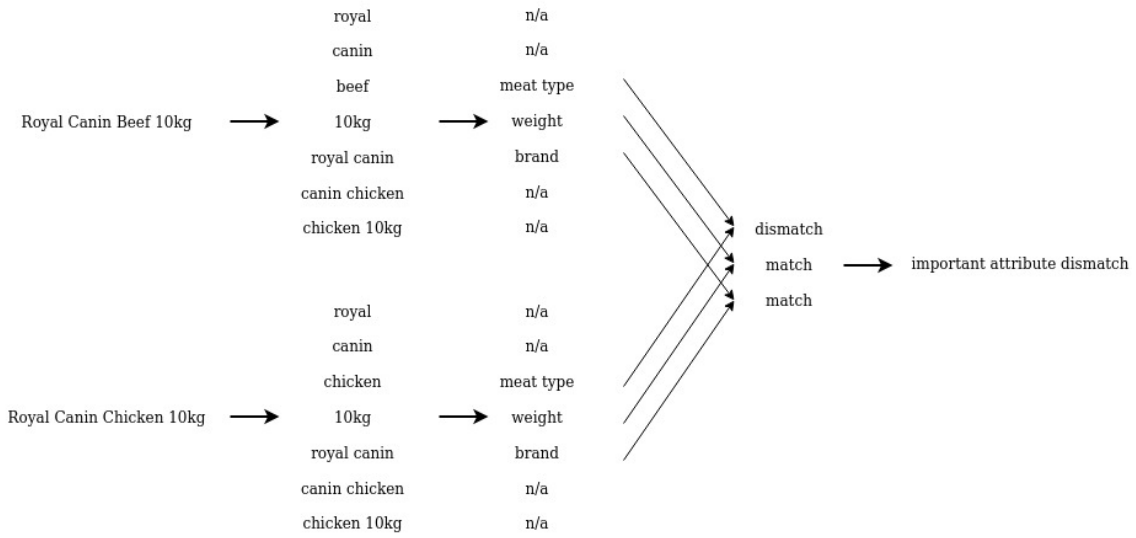

Figure 6: Title attribute matching

5.3.2 SECOND VERSION OF ATTRIBUTE CHECKING, DAMERAU–LEVENSHTEIN VARIANCES WITH THRESHOLD

Damerau-Levenshtein is a metric to measure the edit distance between two words. More precisely, the Damerau-Levenshtein edit distance is a modification of Levesthtein distance, where the distance between two words is the minimum number of edit actions needed to transform the first word into second. The allowed actions are insertions, deletions, substitutions of a single character, or transposition of two adjacent characters. The transposition of two adjacent characters is on top of actions allowed in the classical Levenshtein distance consisting of three classical single-character edit operations [55].

To enrich the previous technique, we utilize this to successfully recognize misspelled attributes. For example, "beef" can now be matched with "bef" or "salmon" with "sallmon." However, attention needs to be put for maximum allowed edit distance. As "small" can be mismatched with "tall" with $DL = 2$.

5.4 EUROPEAN ARTICLE NUMBERS

The EAN number (European Article Number) is a standard that should uniquely identify a retail product by a thirteen-digit code. [56].

These codes should identify products uniquely. However, they are easily mistyped (e.g., if the shop is creating their catalog manually), not provided, or for a completely different product. We observed that the quality of EANs is highly dependent on the target category of offers. For example, while books usually provide good matching results, electronics do not. We thus use them as heuristics for title similarity as

MATCH = True if (EAN matched and TitleSimilarity > X or TitleSimilarity > Y) else False

Here X and Y are empirically chosen values, most of the time $X \in [0.5, 0.7]$ and $Y \in [0.8, 1]$. And the value of TitleSimilarity will be defined in a later section.

5.5 PROPOSED API

As an outcome of this work should be a usable prototype for product matching, one needs to consider options of later deployment. We have considered two options for inference with trained

models. Message queues like Kafka or NATS Streaming allow to continuous processing of data that comes in batches. The second option is to expose it via JSON API for everyone. We utilize the second one because it allows for more natural interaction during development.
An API stands for application programming interface, and it provides convenient access to the underlying application features. Web API is a type of API where the API is accessible via HTTP calls. The service developed in this thesis is available as JSON API, and its specification is shown in the following figure.

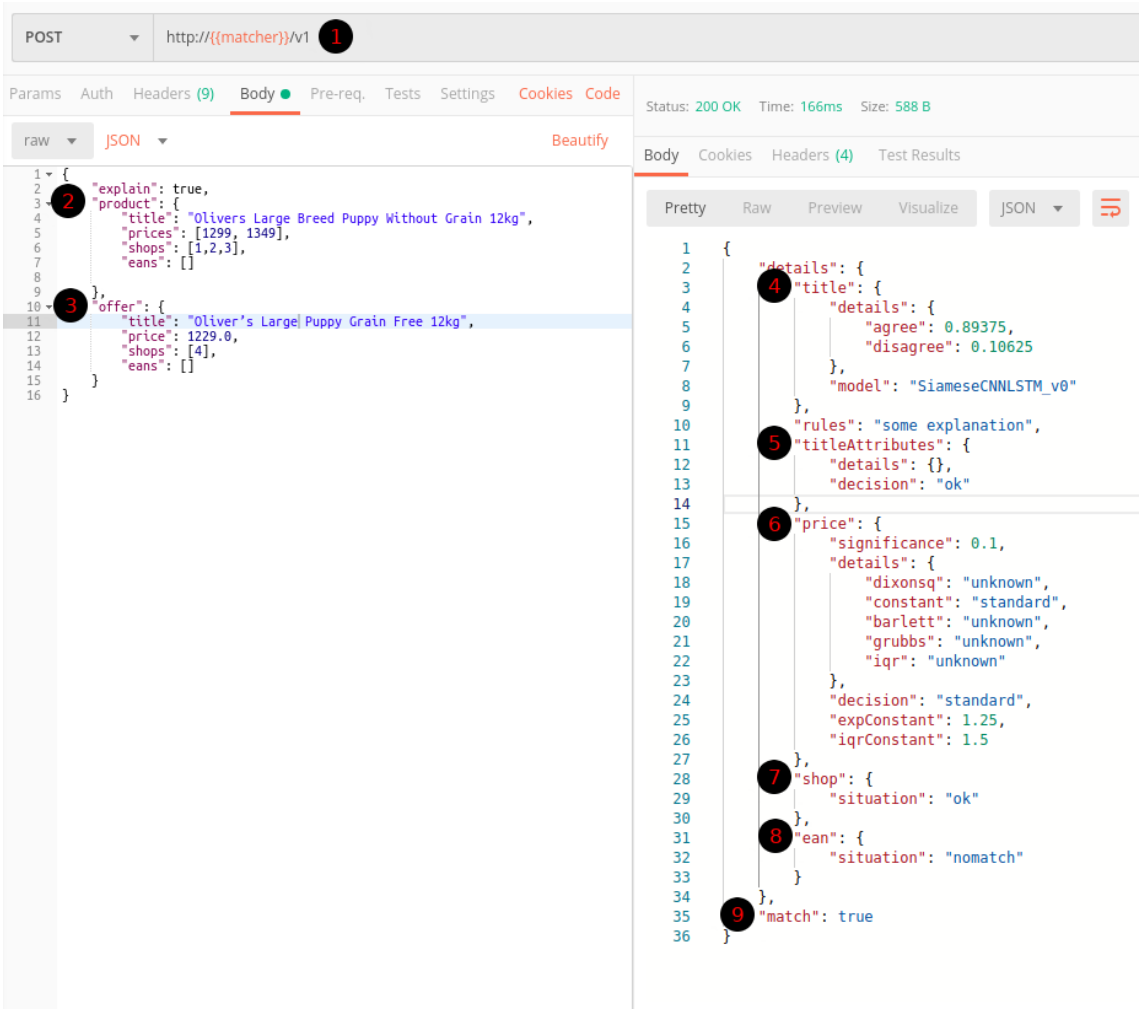

Figure 7: Match API specification

The numbers on the figure 7 represent the following components.

1. Versioning - in case of compatibility breaking changes, API version increases
2. Product to match for - data about product candidate in JSON format
3. Offer that should be tested - data about the incoming offer in JSON format
4. Similarity of titles based on the deep learning model - similarity measurement between product and offer title
5. Decision based on attributes extracted from titles - check of attributes extracted from titles
6. Price outlier detection - check between the distribution of product prices and incoming offer price

7. Check if product and offer does not have the same seller

8. Check if some EAN is matched

Here, the requester puts two objects into the body: the base product and the offer for which we want to decide if it is the same as the product.
The requester can insert several signals. Here, only the product title is currently required. Based on which signals the API receives, the decision about the match is made. For further tuning and debugging, except for the final $match = true||false$ decision, the API also provides per-signal outcomes. This is used for logging responses and fixing models. Thanks to the API interface, we can also send the same request to several models and compare their differences.

6 DEEP LEARNING MODEL

The main part of this work is the research of deep learning architectures for evaluating the similarity between pairs of product titles. In this section, we describe and discuss the most promising architectures that we tried during development.

6.1 OBTAINING DATASET FOR TRAINING AND EVALUATION

With big e-commerce websites, there is a huge amount of products. Products from different categories can usually be distinguished by different signals. For example, in categories such as books or electronics, EAN codes are reliable indicators, but in the category consisting of baths, they are, in most cases, incorrect. We focus our development efforts on the dog food category, which is in our datasets large enough to allow training deep neural networks but not so big that its size would limit us due to lack of computing resources. The second reason to limit research to a few categories is that in the production environment, the company does not want to trigger changes in the entire catalog at once. Testing and monitoring are needed to make sure that the new system does not do more damage than benefits.

We observed that there is a large number of products in the database available to us that are unmatched or are matched incorrectly. We also evaluate our solution by eliminating those cases. The goal here is to iteratively improve our training datasets, as we discuss later.
Since the available database contains only a relatively small number of matched pairs of product offers (from the dog food category) that can serve as positive examples during training, we needed to find a way to generate additional examples from the existing ones to allow training neural networks. To this end, we implemented several production rules that we describe in this section. In the end, with relatively few rules, we were able to produce 2000000+ of quality training pairs from 20 000 products. We believe that by having a lot more correct samples than incorrect ones, we will be able to produce usable and production-ready results. We validate this hypothesis experimentally in section 8.

6.2 PRODUCT AND OFFERS DOWNLOADING IN GO LANG

Deep learning models are trained on thousands to millions of training examples. Obtaining those at the start of each development iteration from the database would be very ineffective. More importantly, the database is always evolving, as new offers come or old ones go. This would make it difficult to compare the results of different models during development.

For this reason, we first downloaded the data for offline usage. We utilized Go lang, which is a fast server-side programming language [57], to communicate with products API. Results are stored in plain text files, one result per line in JSON format, in the same format as from product catalog API. This, along with the second script called Loader, allows us to simulate results from the database for future kind-of online training in automatic production pipelines.

6.3 TOKENIZATION

Given a product title and defined set of rules, tokenization of titles is the process of applying those rules to the titles, for example, by splitting them into pieces by spaces, throwing away non-related characters like punctuation, lowercasing, and many other. The result of tokenization is a set of tokens. The primary question is, what the correct tokens to use are? The starting point for our problem is easy, lowercase, split by spaces, perhaps remove punctuation or stop words.

In product titles, we do not care about most of the information that the usual NLP tasks need to process. We want a maximally clean text to distinguish between two titles. We have created our tokenization algorithm to achieve this, and its results are shown in table 2.

Table 2: TOKENIZATION SAMPLES

Original	Tokenized	Reasoning
Barking Heads Tender Loving Care 6 kg	barking heads tender loving care 6kg	lowercased and units joined with numbers
Hill's PD Canine D/d Venison (zvěřina) 370 g	hills pd canine d/d venison zverina 370g	lowercased, joined units and removed irrelevant characters
CD Healthy Line Adult MAXI, 15kg action 50 %	cd healthy line adult maxi 15kg	lowercased, removed advertision
Hill's Canine Senior 12 x 370 g big breedexp. 02/2020	hills canine senior 12x370g big breed	lowercased, joined units and removed expiration date

This is possible via a combination of character replacements, stop word removal, and regex replacements. They are shown in tables 3 and 4.

Table 3: Sample of stopwords

and, ., „ /, -, +, –, &, *, nbsp, html, body, div, span, table, title, exp, novinka, cashback, osobni, odber, doprava, zdarma, dozivotni, zaruka

Table 4: Sample of regexes

Name	Pattern	Will match
NUM	\d+[.,]?\d*	Numbers, eg. 10, 10.50, etc.
UNIT	x | mm | cm | kg | %	Units
JOIN NU	NUM \s* UNIT \b	50 kg to 50kg

6.4 GENERATING PAIRS OF TITLES

Obtaining enough real data to train deep learning models is often a hard task. Our options were either to summon human forces or come up with a smart solution to synthesize more data from existing. Although human-labeled pairs would likely lead to the best results in production, their acquirement is time-consuming and expensive, as there are hundreds of different categories with millions of products. We embrace the second option and generate more training pairs based on existing ones, with the hope that true positive matches will outcome false positive matches by a large margin and thus provide good enough data. In the following subsections, we describe the production rules that we used in this work. We describe rules for generating both positive and negative examples – this is always indicated in the heading of the subsection.

6.4.1 RANDOM MERGE - POSITIVE PAIR

Given several titles of the same product, create a new title by randomly selecting words in order from these titles. An idea of this method is shown in figure 8.

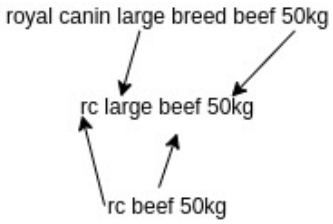

Figure 8: Random merge

6.4.2 SHUFFLE - POSITIVE PAIR

Given one title, shuffle tokens, but pay attention to meaningful n-grams, like producers or flavors. Thanks to the known list of producers, attributes, and their values, we can meaningfully shuffle tokens in such a way that "large breed" stays "large breed." However, due to incompleteness of those lists, it can happen that shuffling will corrupt names, like "royal canin" to "canin 50kg royal" as is shown in figure 9. Depending on the target category, this can but does not have to be a problem.

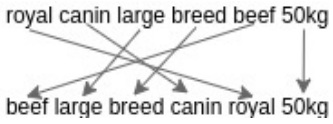

Figure 9: Random shuffle

6.4.3 JOIN TOKENS - POSITIVE PAIR

It is common to accidentally forget space between words, or that two different people are used to write some words together. Given title, randomly join two or three tokens into the one. This process is shown in figure 10.

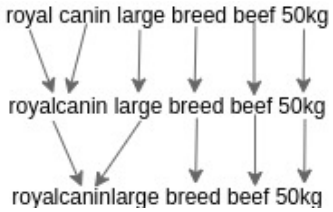

Figure 10: Join tokens

6.4.4 DROP OR SWITCH CHARACTERS - POSITIVE PAIR

Another common situation is misspelling by missing characters and transposition between two adjacent characters. Both are easily modeled by iterating over the title and randomly at some position either drop or switch characters. This is not done on numeric characters because the "16GB" model of the iPhone is very different from the "64GB" one. But "64BG" can be considered as misspelling "64GB".

6.4.5 DROP OR DUPLICATE TOKEN - POSITIVE PAIR

It happens that one title contains more information than another, for example, "iPhone 6S 64GB" and "iPhone 6S" if this happens, we can not decide only on the base of title, but from the title perspective, they are the same.

6.4.6 CREATE ACRONYMS - POSITIVE PAIR

A lot of n-grams can be expressed by acronyms. For example, the brand "Hewlett Packard" is widely known only by "HP." In another situation, shops want to shorten titles to put more attention on significant tokens. "Large Breed" is often shortened to "LB."

6.4.7 CHANGE NUMERIC VALUES - NEGATIVE PAIR

A lot of products come in different sizes and options. Smartphones are distinguishing in the size of their memories while dog food in size of the package. False pairs can be synthesized by changing only numeric characters in the title.

6.4.8 CHANGE ATTRIBUTES - NEGATIVE PAIR

Having a set of known attributes like "beef, chicken, lamb," we can easily replace them in the title and use this newly created as a false pair.

6.5 GENERATED TITLES

Several results of title generation and their labels are shown in table 5.

Table 5: Generated titles

Title 1	Title 2	Label	Reasoning
champion petfoods roijen ppupy 2kg	champion po puppy 2kg	1	Tokens dropped and character misplaced
nutrilove kapsicka filetky kureci 85g	nutrilove kureci 85g	1	Tokens dropped
bravery puppy imni chicken 2gk	bravery puppy mini chicken 2kg	1	Characters misplaced
deyd adult lal brede 6kg	eddy adult all breed 8kg	0	Different weight
delikan junior 10kg	delikan optimal hovezi 10kg	0	Important keywords mismatch
champion petfoods roijen ppupy 2kg	royal canin po puppy 2kg	1	Different brand

6.6 REAL-LIFE TEST SET

Our main goal is to create a method that will work well on real-life databases. As we will see later, accuracy on the synthetic train and test pairs come out very well. But those are all generated pairs using the same algorithm. To better mimic real-world usage, we created a real-life test set consisting of 200+ title pairs, manually created and labeled by us. Those pairs contain title pairs not shown either in train or test split and contains only real-world titles. The production model will then be chosen based on classification accuracy on the generated test set as well as on the real-life test set.

6.7 PADDED DATA-SET STRUCTURE

Each product comes with various names. These names range from three words, e.g., "Apple iPhone 6S", to fourteen words, e.g., "Diamond Naturals Skin Coat Real Meat Recipe Dry Dog Food with Wild Caught Salmon". Given two titles for comparison, the shorter one needs to be padded to the

longer one. This comes from the design of our architectures. However, for efficient training, titles need to be compared in mini-batches.

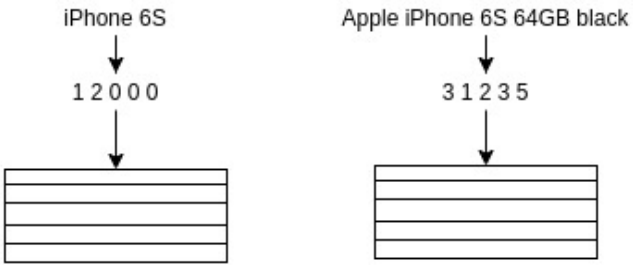

Figure 11: Padding of titles of different length

Padding every title to the longest one in each batch comes out to be a very inefficient technique. With only one long title, there is a huge amount of wasted memory for padding of others, as in figure 11. Also, pairs of titles where both are padded are unnecessary hard for the network. This is illustrated in the following figure, where red backgrounds stand for zero paddings.

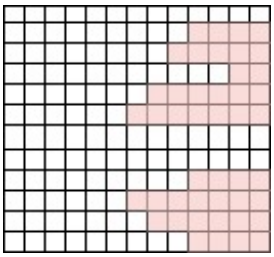

Figure 12: Inefficiency of zero padding in batches

To tackle this, our batching structure automatically groups titles with the same lengths. Because of the big variance in training data, there is no group with less than 300 titles, which is more than enough considering the usual batch size from 32 to 200 samples. Inefficient structure from figure 12 vs. batched one in figure 13 shows this difference.

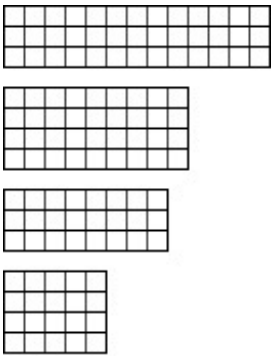

Figure 13: Batches grouped according to title lengths

6.8 TRAINING

6.8.1 TRAINING HARDWARE

Training neural networks on the hardware of personal computers are often an unrealistic task. We used Google Cloud Platform, which allows us to easily switch between machines with 32-core CPU, for tasks optimized for CPU, like FastText and machines with less CPU power but attached GPU

for the training of big neural networks.

Because online on-demand servers are billed hourly, fast set-up and synchronization are needed. We created a bash script to install all necessary libraries like TensorFlow, Python, etc. and rsync for synchronization of development files between machines.

6.8.2 TRAINING SOFTWARE

The main player in the research of machine learning is currently Python. Many libraries provide binding to their C-implementations for fast prototyping and good user experience. However, many alternatives exist, like Julia or Swift. Python is a great language but with two disadvantages that we considered problematic.

Firstly, its speed is often several times slower than that of other languages. [1] In this work, a lot of computations that don't have existing Python bindings to C libraries were needed. So most of the work is written in Swift and especially Swift 4 TensorFlow (S4TF). Python was still used for many tasks where implementation was straightforward and efficient, like reading huge text files without the need to store them in RAM as a whole.

We experimented with two deep learning libraries, PyTorch and S4TF, and chose the latter one, although we still experiment with PyTorch in some cases.

6.8.3 EARLY STOPPING

An epoch in machine learning training is a step when the model sees all the training data, and it is starting again. In each epoch, the training procedure uses gradient descent to iteratively update the model in order to better fit the training data. Early stopping is a technique that stops learning in a point where higher training accuracy would come from the increased testing error, more specifically generalization error that would negatively affect results in production.

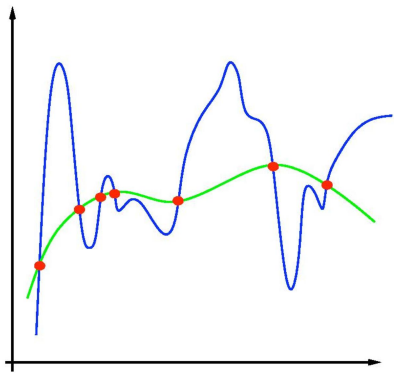

Figure 14: Over fitting

That is the situation when the network sees training data so many times that it starts to learn their exact patterns, instead of generalizing, like in figure 14. In our experiments, when we did not use early stopping, accuracy on the test and validation datasets was sometimes worse than random ($< 50\%$). By keeping track of best-achieved accuracy on the validation dataset, we could avoid this.

Early stopping techniques are employed in many machine learning methods with both theoretical and experimental foundations such as [64] or [36].

[1] During the study at CTU FEE, one needs to implement various fast performing algorithms. And by experience, the same code written in Python timeouts after 5+ minutes of running while the version in C++ executed in below several seconds.

6.8.4 LOSS FUNCTION AND OPTIMIZER

Optimization algorithms help us to minimize a loss function, which is a mathematical function dependent on the model's internal learnable parameters, which are used in computing the target values from the set of predictors used in the model.

For classification tasks, sigmoid cross-entropy and softmax cross-entropy are the two dominant ones. Where the first one can compute losses for the sigmoid output function of the model, deciding between two classes 0 and 1, softmax is a more general version used in multi-classification tasks. Although softmax primary intention is the enhancement of sigmoid for more than 2 classes, it works for binary classification problems too. The later advantage is the re-usability of codebases across more projects. Our neural network uses a dense layer with softmax activation function, so softmax cross-entropy is selected as a loss function, and it is optimized by an Adam optimizer.

6.8.5 ACCURACY MEASUREMENT

In later sections, we evaluate individual architectures based on their achieved accuracy. Accuracy is measured as a portion of the amount of correctly labeled samples vs. the total amount of all samples.

$$accuracy = \frac{|correctdecisions|}{|samples|} \tag{13}$$

6.9 PROPOSED ARCHITECTURES FOR TITLE SIMILARITY MEASUREMENTS

We tested multiple architectures, some providing better results than others. Some were completely unusable. The most promising ones are discussed in this chapter. An interesting observation that was consistent in all architectures we tested is the following. Given a title with M words, where each word is represented in tokenized form as an N-dimensional vector, there are two possible ways to process them in convolutions.

The first option is to stack the vectors on the top of each other, creating an M x N grid, which we apply 2D convolutions to (figure 15). The second option is to consider the title as a 1D vector with N channels (figure 16), where each channel corresponds to one embedding dimension. Surprisingly, in all tests representing titles with the second option leads to faster training and more accurate results.

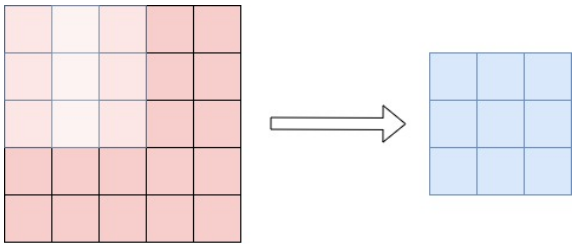

Figure 15: 2D convolution with one channel

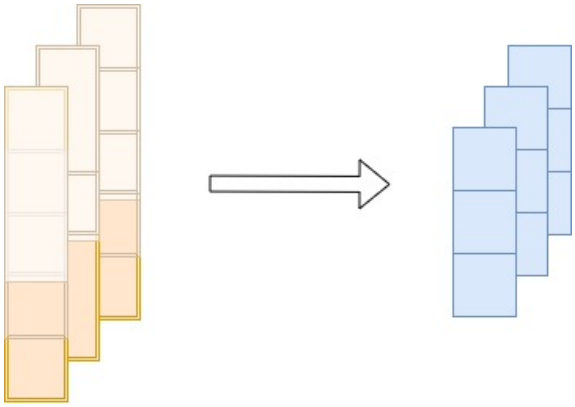

Figure 16: 1D convolution three channels

6.9.1 CONCAT CNN

The first option to learn the similarity between two inputs is to simply concatenate them and let the network derive all required properties. This architecture comes from Walmart's article and is shown in the following figure 17 [1].

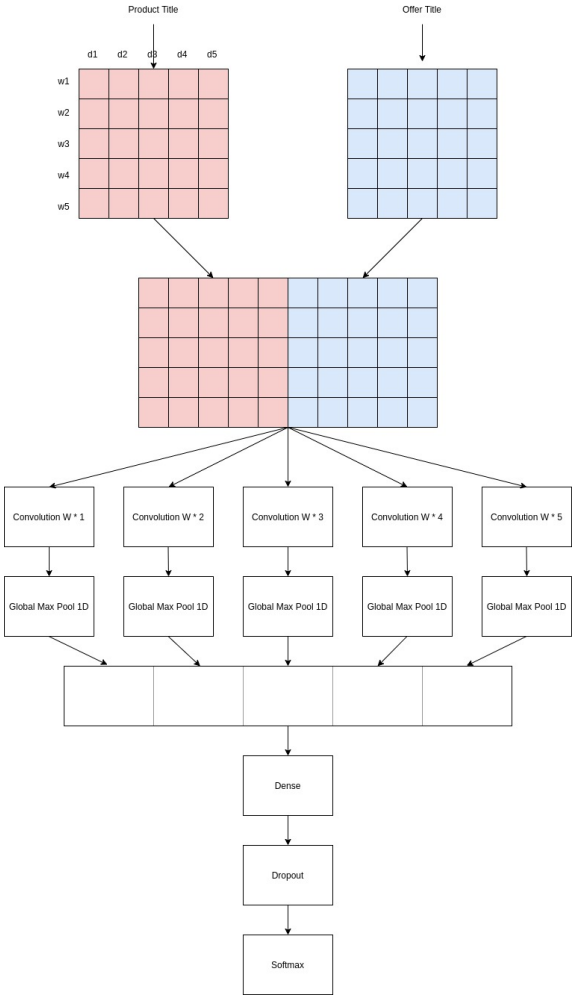

Figure 17: Concat CNN architecture

For an unknown reason, this does not lead to very good results, as shown in the comparison table in the results section 8. One possible explanation is that our dataset was too different from Walmart's one, or training times were too low (although the latter seems rather unlikely based on our observations). In any case, it was a good starting point for baseline results and provided ideas for the following architectures.

6.9.2 SIAMESE CNN, FIRST VERSION

The main idea of Siamese Networks is to forward two inputs in networks with shared weights and originates in 1994 [50]. This leads to a slightly modified architecture, where we process both sentences through convolutions followed by global max-pooling to deal with titles of different lengths. Then the squared difference of vectors representing the titles is made and forwarded through a dense layer with tanh activation, dropout, and final dense layer with softmax activation. This architecture is shown in figure 18.

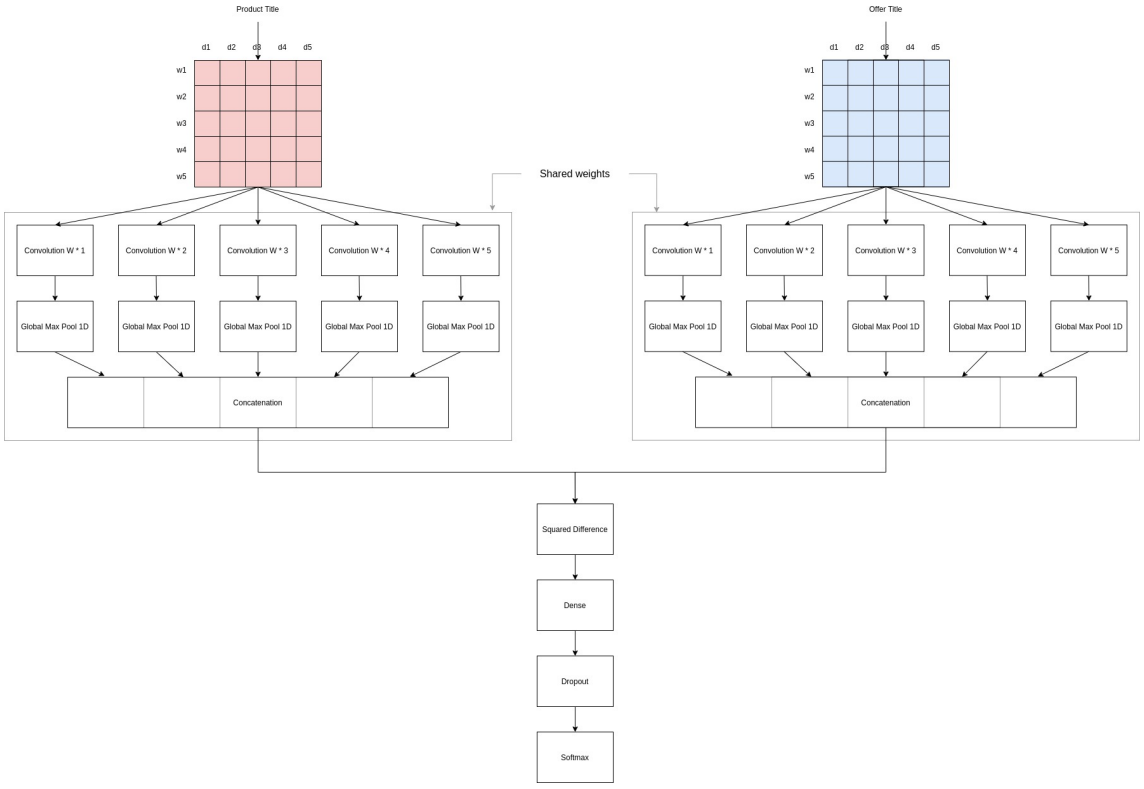

Figure 18: Siamese CNN architecture

6.9.3 SIAMESE CNN, SECOND VERSION

In the second version of this architecture, max pooling is done outside of the network with shared weights. As the difference is calculated as $(a - b)^2$, doing the max pool outside of the shared network results in bigger tensor. However, the final accuracy is not much higher, as will be shown in the results section. Resulting architecture is slight modification of figure 18 in figure 19.

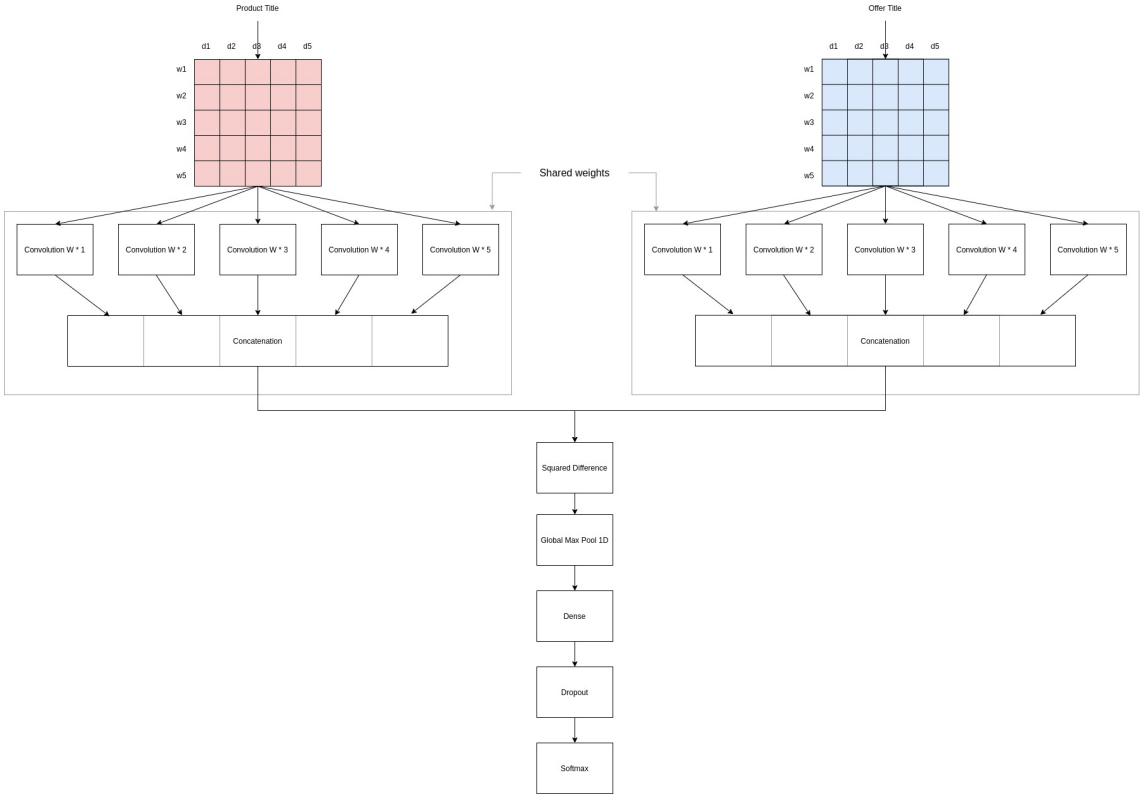

Figure 19: Siamese CNN v2 architecture

6.9.4 SIAMESE CNN-LSTM, FIRST VERSION

One big drawback of the previous architectures is the usage of max-pooling to handle titles of different lengths. This can quickly lead to loss of important information. The addition of min or avg pooling layers, which we also tried, did not help to achieve better accuracy, probably because titles have to be zero-padded, and those zeros devalued outcomes of those pooling layers.

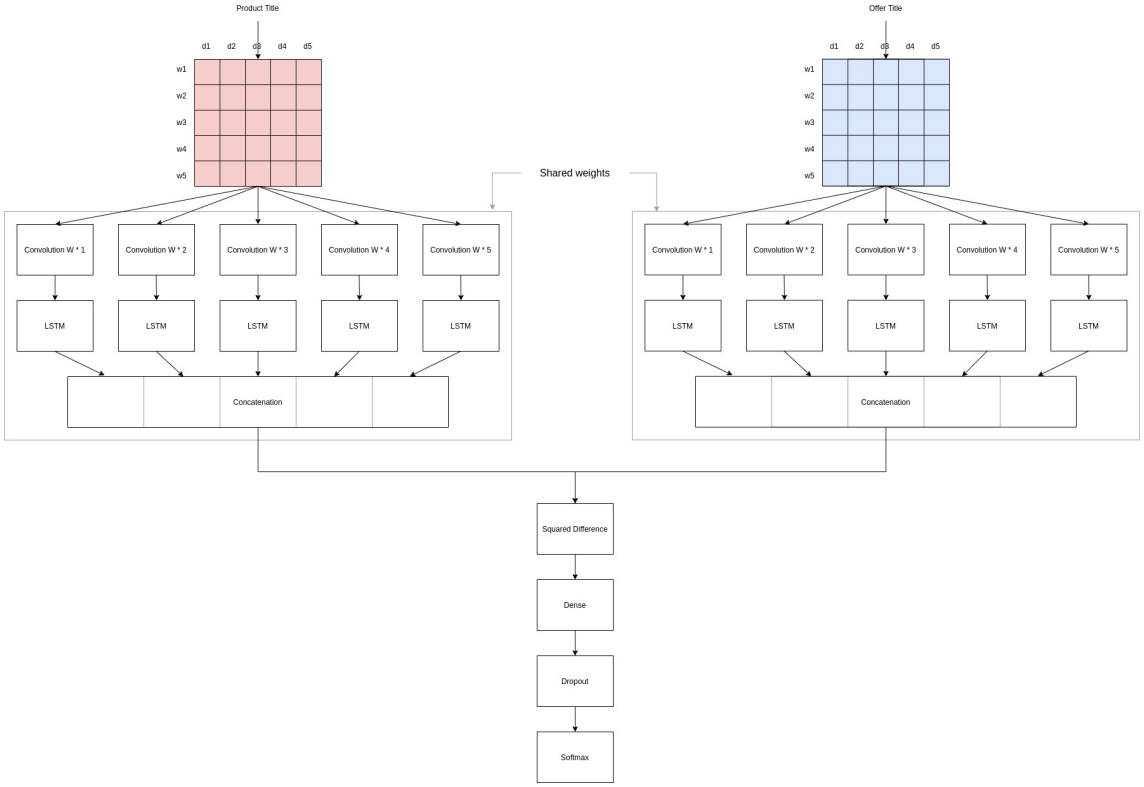

Figure 20: Siamese CNN-LSTM architecture

By replacing max-pooling with LSTM units and using their last hidden state as a comparison vector for squared difference function, the network is able to learn important segments of title extracted by previous convolution layers.

6.9.5 SIAMESE CNN-LSTM, SECOND VERSION

Vanilla LSTMs [47] are one layer deep with forwarding pass only. Previous works [8] have shown that stacking several LSTM, creating Deep LSTM architecture can lead to better results. Another famous LSTM architecture consists based on passing of a sequence in a forward direction into one LSTM while in another direction to second LSTM and thus creating BiDirectional LSTM [8]. However, neither of those adaptations provided better results in our experiments. Training time was also highly increased, and comparisons were harder to make because the same number of epochs took a lot more time, and perhaps with the same training time (and so more epochs), the accuracy of previous networks would increase too.

6.9.6 TRIPLET SIAMESE CNN

Triplet loss [53] is a special form of loss used to learn a representation of the input in space, where similar input are clustered together and dissimilar drawn from each other. The decision about similarity is then made directly on the distance between two vectors.

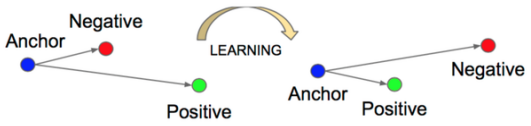

Figure 21: Visualization of triplet learning

An anchor (baseline) input is compared to a positive (in our case matching offer) input and a negative (different product) input, as shown in figure 21. The L2 distance from the anchor input to the positive input is minimized, and the L2 distance from the anchor input to the negative input is maximized.

$$L(A, P, N) = \max\left(||f(A) - f(P)||^2 - |f(A) - f(N)||^2 + \alpha, 0\right) \tag{14}$$

The problem is then minimization of

$$J = \sum_{i=1}^{M} L(A^i, P^i, L^i) \tag{15}$$

Facebook [49] has achieved SOTA results at that time using this method together with a smart sampling of triples. We tried to mimic this architecture. However, the results were not promising. This can be because of several reasons. First, product title similarity is a very different problem from face recognition. Secondly, their learning times were huge compared to ours. Perhaps if we had more computing power and time, the results would outperform our Siamese CNN-LSTM architecture with softmax outputs. More exploration of distance minimizing losses are planned for future work.

7 GRADIENT BOOSTING TO UNIFY ALL SIGNALS

Every component produces an important signal for the final matching decision, but in order to get high accuracy, we also need to learn how important each of them is. For example, 95% match in the name means that offer is very likely belonging to the product, but if many other signals say that the offer should not match, it can mean that eshop provided incorrect title or that model for title evaluation is faulty. Figure 22 shows part of one of the decision trees from the ensemble trained using XGBoost, where f0 stands for the similarity between offer and product name and f7-10 are various price outlier tests mentioned in section 5.2.

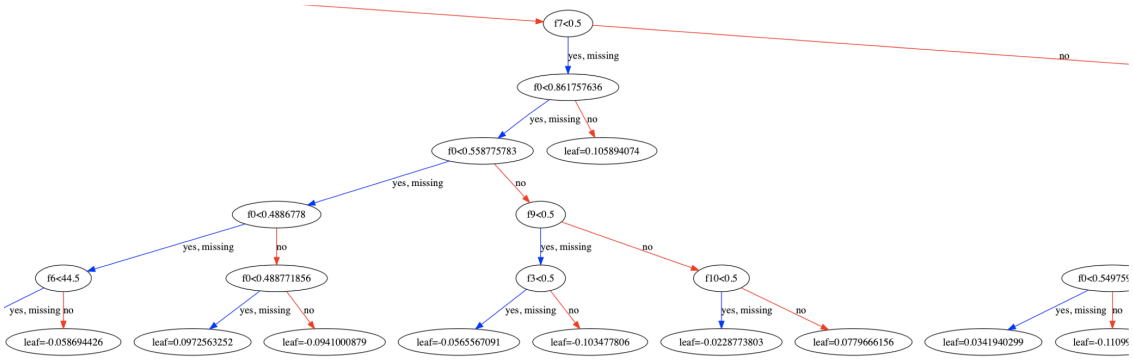

Figure 22: Match decision visualization using trees

As we can see from the figure, the main part of the matching decision is indeed product title, as it is the most used node in the tree with various rules, and other nodes provide hints in cases of uncertainty. It will be interesting to see how this tree changes with the addition of other deep learning models.

8 RESULTS

8.1 DIFFERENT MODEL ARCHITECTURES

The best accuracy on the validation dataset was achieved by the Siamese CNN-LSTM. The comparison of individual architectures is shown in table 6 below. For computational time reasons,

real-world test cases were made only on most promising architectures and are reported in section 8.4.

Table 6: Model architectures results

Model / Accuracy	Train.	Valid.	Duplicates	Bad match
SiameseCNN v1	0.971	0.811	-	-
SiameseCNN v2	0.982	0.827	0.929	0.832
SiameseCNN-LSTM v1	0.970	0.890	0.958	0.879
SiameseCNN-LSTM v2	0.967	0.833	-	-
Triplet	0.746	0.631	-	-

8.2 DIFFERENT EMBEDDING TYPES

In this section, we describe the effect on the accuracy and speed of different embedding methods. A standard method used in many tasks is word-level embedding [39], which consists of learning vector representations on per word basic. In section 8.2.1, we describe achieved results using this methodology with the method called Word2Vec [42]. Second, more robust methods are sub-word or char-level methods. In section 8.2.2 we describe results achieved using library called FastText [21].

All measurements in the following sections were evaluated using SIAMESE CNN-LSTM, FIRST VERSION architecture. The reason is that this architecture has shown the best results for our use cases and training all possible combinations of embedding type - architecture - hyperparameters would be unachievable due to computational and time limitations.

8.2.1 WORD LEVEL

In table 7 are shown results of SIAMESE CNN-LSTM, FIRST VERSION model accepting titles as tensors of dimension MxN, where M is a number of words in the title and N is embedding dimension. We tried the most frequently used dimensions in the literature [51]. Choosing the right embedding dimension is an empirical task, too low, and the vector will not be able to obtain enough information. Too much and vector will be more random then informative because of the lack of enough training data.

Table 7: Results of word level embeddings

Dimension	E. Model	Corpus	Training acc. (at stop)	Real-life acc. (best)
100	Word2Vec	Products	0.968	0.793
150	Word2Vec	Products	0.971	0.798
200	Word2Vec	Products	0.959	0.658
300	Word2Vec	Products	0.969	0.724

Based on results from table 7, dimension 150 seems to be the best fit. The main issue with higher dimensions (200, 300) is training difficulty, as it takes much more time to achieve an accuracy as with 100 or 150 dimensions. However, there is a chance that with even longer training times, accuracy would go up, but it does not seem that it would be worth the increased computing complexity. We can see that training accuracy is very good because, at the training phase, all words are known. However, the accuracy of real-life dataset suffers a lot. Our hypothesis was that this is due to non-informative vectors in the titles with the out vocabulary words.

8.2.2 SUB-WORD LEVEL

To improve accuracy and test our hypothesis about OOV words from section 8.2.1, we will train the same model on the same dataset and evaluate it on the same real-life set of titles as in the previous section. In table 8 we show results of SIAMESE CNN-LSTM, FIRST VERSION architecture fed with the FastText [21] embeddings.

Table 8: Results of sub-word embeddings

Dimension	E. Model	Corpus	Training acc. (at stop)	Real-life acc. (best)
100	FastText	Products	0.970	0.890
150	FastText	Products	0.967	0.881
200	FastText	Products	0.971	0.874
300	FastText	Products	0.853	0.769

As we can see, the training accuracy is similar to the accuracy in the table with the word-level embeddings 7 and, in some cases, even lower. But the difference is negligible and could be easily caused by factors like random initial weights or training time. The primary focus is accuracy on the real-life test because, in the production environment, titles will not be generated by our algorithm or known in advance. They will be provided by eshops and will not always follow the same patterns. As shown in table 8, real-life accuracy is significantly higher than from the model based on word-level embedding.

8.2.3 PRE-TRAINED

There are many available pre-trained word vectors for various embedding approaches. For example, FastText has pre-trained word vectors for 157 different languages available from its website [22]. Those are trained on big corpora like Common Crawl and Wikipedia. As their dimension 300 was unnecessary high for our problem, we needed to reduce the dimensionality. There is a lot of ways to reduce dimensions of vectors while preserving their main features, like Principal component analysis [23], Non-negative matrix factorization [33], or Kernel PCA. We used the official FastText method to reduce dimensions from 300 into smaller ones. The source code for this operation is available at [13] and uses simple PCA. Accuracies achieved using pre-trained embeddings are shown in table 9.

Table 9: Results of pre-trained FastText embedding

Dimension	E. Model	Corpus	Training acc. (at stop)	Validation acc. (best)
100	FastText	Wikipedia	0.954	0.674
150	FastText	Wikipedia	0.952	0.632
200	FastText	Wikipedia	0.963	0.713
300	FastText	Wikipedia	0.941	0.589

8.2.4 TRAINED PER-CATEGORY

The second option usually done in NLP tasks is to train embedding directly on our data. There is one big advantage and disadvantage of this. Firstly, our dataset is very limited in vocabulary in comparison with big dumps like Wikipedia, so the handling of OOV words can be worse. But secondly, by training embedding directly on our corpus, even embeddings alone can be a good base of product similarity by placing tokens of similar products to a similar place in space. Accuracies achieved using per-category trained embeddings are shown in table 10.

Table 10: Results of per category trained FastText embeddings

Dimension	E. Model	Corpus	Training acc. (at stop)	Validation acc. (best)
100	FastText	Products	0.970	0.890
150	FastText	Products	0.967	0.881
200	FastText	Products	0.971	0.874
300	FastText	Products	0.853	0.769

8.2.5 CONCLUSION

In this section, we evaluated different methods for learning word representations in the vector space. We had shown that dimension of 150 best suits our datasets based on final accuracy as well

as training and inference times. And we had also shown that word-level representations provide good results on training data but perform worse on unseen data, this should not be a big surprise. Usually, in the literature, word-level embedding is used on the whole documents or chunk of texts like product review [32], but product titles tend to be short and descriptive, so few missing words due to lack of training data can result in the missing information like product brand or color.

8.3 IMPACT OF THE AMOUNT OF TRAINING DATA

The amount of data one needs depends both on the complexity of the problem and on the complexity of the chosen model. But this does not help if we are at the pointy end of a machine learning project. In machine learning, we are trying to learn a function that maps input data to the outputs as accurately as possible. And thus, all models can be only as good as the data we provide to them.

With generation, we can make a very large amount of training data. But after some time, patterns in the samples will be the same and will not contribute to training very much. The training was tested on several training set sizes, and results are evaluated with the best-chosen architecture and shown in table 11 below.

Table 11: Results of training on different sizes of training samples

Training pairs number	Training acc. (at stop)	Validation acc. (best)
100 000	0.995	0.748
1 000 000	0.987	0.845
5 000 000	0.97	0.890
10 000 000	0.976	0.887
30 000 000	0.968	0.889

8.4 EVALUATION ON REAL-WORLD CASES

To showcase the final solution, we try its evaluation on two real-world problems that often concern big online product catalogs. First, finding products that should be matched but are not, and the second one, finding offers that are matched into one product but should be not. This experiment is interesting because we did not know those in advance, and they might have been included in our training data as incorrectly labeled examples. The identified pairs can be used in practice for recommendations to human annotators to check and possibly update these records in the databases. The network can then be retrained with data of better quality.

8.4.1 FINDING DUPLICATES IN EXISTING DATABASE

Given all products in a specific category, we want to find those that should be matched but are not. Below is table 12 with a few detected samples.

Table 12: Based on similarity between two titles

Title 1	Title 2	Sim	Reasoning
Happy Dog Natur-Croq Geflügel Pur Rice 15 kg	Happy Dog Natur-croq 23/10 Geflugel pur reis 15 kg	0.97474	Almost certain match
Happy Dog Lamm Rice Ecken 10 kg	Happy Dog Cano Lamm Reis Ecke 10 kg	0.97428	Misspeled and missing token, but almost certainly match
Nature's Protection SC White Dog Adult Large Breeds 10kg	Nature's Protection Dog Superior Adult White Large breed 10kg	0.95807	All important tokens match
Doxneo Duck 2,5 kg	Doxneo 1 Duck 2,5 kg	0.91282	Almost certain match
Happy Dog Natur-Croq Geflügel Pur Rice 15 kg	Happy Dog Natur-Croq Geflügel Pur Reis 15 kg	0.89091	"Rice" and "Reis" are same attributes, different language
PURINA PRO PLAN Dog Duo Delice Adult Chicken Rice 2 x 10 kg	Purina Pro Plan Duo Delice Adult chicken 2 x 10 kg	0.87942	"Rice" is missing, but usually it is match
Sportmix Adult Mini 20 kg	Sportmix Adult Maitenance Mini 20 kg	0.83581	Usually not important attribute missing, almost certainly match
Eukanuba Mature Large Breed 2 x 15 kg	Eukanuba Mature Large 2 x 15 kg	0.81184	Almost certain match
Nutram T28 Total Grain Free Small Breed Salmon 6,8 kg	Nutram T28 Total Grain Free Salmon Trout Small Breed 6,8 kg	0.73944	This should ideally have bigger probability of match
Hill's Canine i/d Digestive Care Turkey 12x360g	Hills Prescription Diet i/d Digestive Care s krůtím - 12 x 360 g	0.73839	"Turkey" and "krůtím" are same attributes in different languages
Nature's Protection Adult Light 12 kg	Nature's Protection Light 12 kg	0.6723	"Adult" is missing, we can not be sure, but more probably match
Belcando Jehněčí 800 g	Belcando jehněčí rýže rajče 800 g	0.58598	Two attributes are missing in first title, but probably match

8.4.2 FINDING BADLY MATCHED OFFERS IN EXISTING DATABASE

The opposite problem is when the offer is matched to the incorrect product. This can result in even bigger financial losses than the previous case. Because if the user orders a product, another one will be delivered. Sample of such cases shown in table 13.

Table 13: Based on similarity between two titles

Title 1	Title 2	Sim	Reasoning
Purizon Single Meat Adult jehněčí a hrách 1 kg	Purizon Single Meat Adult losos a špenát - bez obilovin - 4 kg	0.02755	Different flavour
PIPER Senior s jehněčím 400 g	PIPER SENIOR konzerva pro psy jehně pro seniory 400g	0.03239	This should not been detected as bad match
Nutram SOUND Senior 3 x 13,6 kg S10	Nutram Sound Senior Dog - pro psí seniory všech plemen 13,6 kg	0.03255	Different weight
Dolina Noteci Premium malé rasy s husou, brambory a jablkem 185 g	DOLINA NOTECI PREMIUM kure, brambory a jablka 185g pro dospělé psy malých plemen	0.03258	Different flavour
Solo Vitello 100% telecí vanička 300 g	SOLO Buffalo 100% (bůvol) vanička 300g	0.03386	Different flavour
Happy Dog konzerva Strauß Pur - pštrosí 400 g	Happy Dog Premium konzerva 100% Strauss Pur 400 g	0.0346	"Strauss" was not recognized as different language of "pštrosí"
Profidog kapsička filety krůtí a zvěřinové ve šťávě 12 x 85 g	PROFIDOG kapsička filety hovězí a kuřecí ve šťávě 12x85g	0.03491	Different flavour

8.5 ITERATIVE IMPROVEMENT OF THE TRAINING DATASET

After results are checked and labeled by humans, they are corrected in the catalog database. This is desired because of several reasons. Firstly, by removing duplicates and bad matches in the catalog, we immediately improve user experience and thus business results. This is very helpful in the company as we can demonstrate partial results right away, allowing continuation in development with so said clean shield. Secondly, by improving the catalog, we can re-download and re-create more accurate training datasets, which will allow us to be more confident in the generation of samples, allowing us to create more worthy samples.

8.6 CLASSIFICATION USING GRADIENT BOOSTING

In the sections above, we show matching accuracy using title alone. As we can see, this leads to good results as the title is the main describer of the product, but in many cases, it can be misleading, incorrect, or just not specific enough to correctly distinguish between two products even for humans. In section 5, we described several additional signals that could potentially identify two different products, mainly thanks to the price and existing database of attributes. In the first iteration, we tried to hard-code decisions about when offer should be rejected even when the title shows strong similarity. Figure 23 shows one of those rules in the code.

```
if product.prices.count > 0 {
    let price = matchPrice(
        response: &response, product: product, offer: offer)

    if !price {
        if similarityName < 0.8 {
            return .no
        }
```

Figure 23: Hard-coded price decision rule

However, doing this for all available signals and perhaps more of them in the near future would be very inefficient, and we would probably miss more complicated patters. Thanks to the numerical nature of individual signals we have been able to create a second training dataset, where each column corresponds to one of the signals, this is shown at table 14, where columns stand for a label, name similarity, keywords match, name attributes, match, shops match, number of matched attributes, number of unmatched attributes, ean code match, and price outlier tests.

Table 14: Sample of dataset used for XGBoost

0.0	0.54122615	1.0	1.0	0.0	0.0	0.0	-1.0	0.0	2.0	0.0	0.0	0.0
1.0	0.66885996	1.0	1.0	0.0	0.0	0.0	-1.0	2.0	2.0	2.0	2.0	2.0
1.0	0.8687079	1.0	1.0	1.0	0.0	0.0	25.0	1.0	1.0	1.0	1.0	1.0

Results from the prediction based on XGBoost models with various parameters are shown in table 15.

Table 15: XGBoost results

Parameters	Category	Training acc.	Testing acc.
booster=gbtree $\max_d epth = 6 subsample = 0.5$	Dog food	0.975	0.946
booster=gbtree $\max_d epth = 6 subsample = 1.0$	Dog food	0.958	0.935
booster=gbtree $\max_d epth = 12 subsample = 1.0$	Dog food	0.948	0.943
booster=gbtree $\max_d epth = 6 subsample = 0.5$	Keyboards	0.972	0.926
booster=gbtree $\max_d epth = 12 subsample = 1.0$	Keyboards	0.955	0.933

As we can see, final accuracy is much higher than the accuracy provided solely by similarity based on the titles, and this extra training step is worth taking.

9 TESTING GENERALIZATION OF THIS SOLUTION ON OTHER CATEGORIES

All initial tests were made on category Dog food. Because this category tends to be somehow problematic with big variations in names of the same product, to test whether this solution can be scaled to other categories, we test the whole pipeline on several different categories. That means download products with collector, generate samples, train embeddings, and neural network and then evaluate results.

9.1 COMPUTER KEYBOARDS

Computer Keyboard is a category with roughly 1500 products and more than 10 000 offers. We evaluate both real-life tests, namely product duplicate detection and mispaired offers elimination. In table 16 we show a comparison of two best appealing architectures.

Table 16: Models accuracy on different category

Model	Train. acc.	Valid. acc.	Duplicates acc.	Bad match acc.
SiameseCNN v2	0.947	0.832	0.911	0.828
SiameseCNN-LSTM v1	0.991	0.878	0.942	0.841

9.1.1 DUPLICATES

Keyboard titles are usually straightforward, consisting of brand name, model name, and mostly model number. This lead to a minimal amount of found duplicates. Nevertheless, we show the results of this test in table 17.

Table 17: Title similarities on different category

Title 1	Title 2	Sim.	Reasoning
Cherry TouchBoard G80-11900LTMEU-2	Cherry G80-11900LTMEU-2	0.94795	First title has one more describing token, but model type is the same.
E-Blue EKM752 YCEBUB72XU00	E-Blue EKM752 EKM752MGUS-IU	0.71814	Second title excplicitly says that it is US variant of keyboard, but model type match. Most of the time this means same product, but we can not be sure just from title alone.
E-Blue Cobra II EKM705BK	E-Blue II EKM705BKUS-IU	0.67104	Second title has explicit declaration of language variant and model name is missing. Similarity is thus even lower.

9.1.2 BAD MATCHES

Finding of mispaired offers in some categories is harder than in others. Especially in computer keyboards, eshops tends to add a lot of redundant text that can easily mislead machine learning models. In table 18, we show results with the lowest title similarity, which should lead to mispaired offers.

Table 18: Bad matches on different category

Title 1	Title 2	Sim.	Reasoning
Canyon CNE-CKEY01-CZ	BARBONE CI-58B klávesnice HI1CNECKEY01SPCZ	0.00964	Different brand and model number
A4Tech KB-720	A4Tech KB-720, PS/2, černá (KB-720 BLACK)	0.01544	Almost same keyboard but different cable type.
Dell KB522 580-17667	DELL KB522 - Klávesnice - USB - USA/evropská (QWERTY) - černá - pro Inspiron 11 3179, 15 5 (DELL-580-17667)	0.01737	This is same keyboard, but model is confused from notebook description.
Esperanza Titanium TK101UA	Klávesnice standard černá TK100UA, UA, USB připojení	0.02083	Different model type.
Cherry CYMOTION PRO CORDED G85-20050TSAABA černá	CHERRY CYMOTION PRO CORDED Czech, USB + PS/2, černostříbrná; G85-20050TSAABA	0.02097	Different color.
Cherry CYMOTION PRO CORDED G85-20050TSAABA	CHERRY CYMOTION PRO CORDED Czech (G85-20050TSAABA) Silver/Black USB + PS/2 adapt	0.02482	Second comes with adapter and thus has bigger price.
Keytools Big Keys Plus verze 4b K-BK-QC-P USB	KEYTOOLS Big Keys Plus (verze 4b) PS/2, QWERTY, černé písmo, barevné klávesy; K-BK-QC-P	0.02624	Different cable type.
Thermaltake Soprano Aluminum A2478CZ	THERMALTAKE A2478CZ Soprano Aluminu / černá / CZ / + Cyber Clean Teaser Pack 40g za 1,- (A2478)	0.02669	Different color and more items in pack.
Keytools Big Keys Plus verze 3b K-BK-AC-P	KEYTOOLS Big Keys Plus (verze 3b) PS/2, ABC, černé písmo, barevné klávesy; K-BK-AC-P	0.02934	This should be same product, but model is consufed from description in second title.

9.2 BATHTUBS

The second category to test real-life cases is bathtubs. With about 8000 products and 30 000 offers is this category a few times bigger than the previous one. Table 19 shows a comparison of the most promising architectures.

Table 19: Models accuracies on different category

Model	Train. acc.	Valid. acc.	Duplicates acc.	Bad match acc.
SiameseCNN v2	0.931	0.832	0.911	0.828
SiameseCNN-LSTM v1	0.963	0.827	0.931	0.883

9.2.1 DUPLICATES

In contrast with keyboards, the names of bathtubs are much harder to analyze. Bathtubs come in various sizes, and even a 10cm difference in length is considered a different product, although, for NLP, it is just one character. Table 20 shows product duplicates found in this category.

Table 20: Title similarity on different category

Title 1	Title 2	Sim.	Reasoning
Laufen Ilbagno alessi 183 x 87 cm H2459720006251	Laufen Il Bagno Alessi One 183 x 87 cm H2459720006251	0.88665	Same brand, model number and dimensions.
Laufen Ilbagno alessi 178 x 82 cm H2459710006051	Laufen Il Bagno Alessi One 178 x 82 cm H2459710006051	0.88472	Same brand, model number and dimensions.
Laufen Ilbagno alessi 178 x 82 cm H2459710000001	Laufen Il Bagno Alessi One 178 x 82 cm H2459710000001	0.88114	Same brand, model number and dimensions.
Laufen Alessi One 204 x 102 cm H2439700000001	Laufen Il Bagno Alessi One 204 x 102 cm H2439700000001	0.84616	Same brand, model number and dimensions.
RIHO LUGO 180 x 80 BT0200500000000	Riho LUGO 180 x 80 cm BT0200500000000	0.84195	Same brand, model number and dimensions.
Laufen Ilbagno alessi 183 x 87 cm H2459720006151	Laufen Il Bagno Alessi One H2459720006151	0.82356	Same brand and model number.

9.2.2 BAD MATCHES

As bathtub names are hard to match, various forms of regexes are used to match as many offers as possible. This can lead to various mismatches, mainly because of different sizes or orientations. Table 21 shows top eight detected cases.

Table 21: Bad matches on different category

Title 1	Title 2	Sim.	Reasoning
Polysan VIVA B SLIM 175 x 100 cm 88119S	Sapho VIVA B SLIM obdélníková vana 175x80x47cm, bílá 88119S	0.02514	Different dimensions of otherwise same bath.
Polysan VERSYS 160 x 85 cm 15611	Sapho VERSYS R asymetrická vana 160x84x70x47cm, bílá 15611	0.02528	Different dimension, although it is probably only small mistake in the title.
SAPHO OBLO 165 x 48 cm 72840	Polysan OBLO 165x165 vana s konstrukcí - 72840	0.02581	Different dimensions.
Polysan VERSYS 160 x 85 cm 14611	Sapho VERSYS L asymetrická vana 160x84x70x47cm, bílá 14611	0.02594	Different dimension, although it is probably only small mistake in the title.
Sapho SAIMA 156 x 95 cm 72360	Polysan Saima Volně stojící vana 176x95x60cm, bílá, 72360	0.02661	Different brand and dimensions.
Sapho SAIMA 176 x 95 cm 72360	POLYSAN SAIMA volně stojící vana 176x75x60cm, bílá (72360)	0.02661	Different brand and dimensions.
Roth Stone Amore 160 x 85 cm 9930000	Roltechnik STONE AMORE Oválná vana z litého mramoru integrovaný sifon 1600x850 ROL-9930000	0.02741	Dimensions of second title are probably in mm and model does not understand that.
Sapho ZASU 180 x 81 x 58 69611	ZASU volně stojící vana 150x81x58cm, bílá	0.02847	Different brand and dimensions.

10 OCCURRED PROBLEMS

As with all software engineering tasks, the experience is needed for clear development. Many bugs or mistakes were made. For example, during training with word embedding, there was a special character inside one title that some applications threatened as a new line. This caused all embedding after this title was shifted by one index. So embedding at index N was for title at index N - 1. It also becomes clear that it is needed to check data after every operation. If we know that there should be no duplicates or that array should have no negative values, it is helpful to check this. This extra little computation time involved is negligible in comparison with saved developer time.

10.1 CALIBRATION OF NEURAL NETWORK

Overconfidence is a common problem in neural networks [4] [18] [19], where the model always outputs 0 or 1 as probability, even when it is wrong. In [19], they limit this by introducing temperature hyperparameter, activated during inference, while standard softmax has the form.

$$\sigma(z)_i = \frac{\exp z_i}{\sum_{j=1}^{K} \exp z_j}, \tag{16}$$

after applying temperature T, softmax will take the form of

$$\sigma(z)_i = \frac{\exp \frac{z_i}{T}}{\sum_{j=1}^{K} \exp \frac{z_j}{T}} \tag{17}$$

This allows us to scale probabilities, in the ideal case, they would then correspond to the probability of being right. However, in our experiments, we were not able to find such value of T that would accomplish this, but using temperature allowed us to sort results by their probabilities. With $P \in \{0, 1\}$ ordering was not possible, and all results seemed equally probable. With the adapted softmax and $P \in [0, 1]$ we can order them and take the most promising one.

11 OPEN-SOURCE CONTRIBUTIONS

During the work, several opportunities to contribute to the world of open-source emerged. Because we believe that by helping others, we can drive research progress, we happily embraced them.

11.1 SWIFT 4 TENSORFLOW

As S4TF is still evolving, few things are missing. Fortunately, as a fully open-source project with the helpful community and their google mailing list making new functionality is pretty straightforward.

11.1.1 BIDIRECTIONAL RECURRENT LAYERS

As discussed in 3.4, recurrent layers take an array of tensors as input, performing various non-linear transformations to generate a hidden vector state. In vanilla recurrent layers, the input is taken one by one from the beginning of array to end. However, literature [11] shows that passing input to two individual RNNs, one in a forward and second in a backward direction, can provide better results. Unfortunately, S4TF did not has a universal bidirectional rnn structure. This pull request on Github [24] introduces a generic structure for all available recurrent layers (Basic, LSTM, GRU).

[24]

11.1.2 TENSOR PRODUCT DIFFERENTIATION

Product is the result of pointwise multiplying. With tensors, a product of two tensors denoted $V \otimes W$ can be visualized as in figure 24.

$$\mathbf{v} \otimes \mathbf{w} = \begin{bmatrix} v_1 w_1 & v_1 w_2 & \cdots & v_1 w_m \\ v_2 w_1 & v_2 w_2 & \cdots & v_2 w_m \\ \vdots & \vdots & \ddots & \vdots \\ v_n w_1 & v_n w_2 & \cdots & v_n w_m \end{bmatrix}$$

Figure 24: Product of vectors

This computation was required to compute cosine similarity with batch data for efficient word2vec training.

$$similarity = cos(\phi) = \frac{A \cdot B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{18}$$

In simple terms, the gradient of the product can be expressed by dividing the product by each entry of the input tensor, but this approach can't deal with zeros in the input. So, we avoid this problem by composing the output as a product of two cumprod operations. Source code with implementation can be seen at [28].

11.1.3 MODELS BUG FIXES

With active development, it is common to accept pull requests with possible bugs. When we noticed such situations, we tried our best to fix them. One example can be found at [26], fixing missing executable for model training.

11.2 TESTS

With big frameworks like TensorFlow or PyTorch, people usually take for granted that underlying, hidden, implementations are correct. However, this does not has to be always true [29] [31]. If in doubt, it is a good idea to test outcomes of layers against outcomes in different frameworks, if they do not match, we either do not understand the implementation details correctly, or one of them has a bug. Both cases can lead to a loss in accuracy or abandoning of a theoretically good model. This was a case with the implementation of GRUCell in the Recurrent layers [30], where we created a pull request with more detailed tests and fixes of incorrect equations.

11.3 PYPIKA QUERY BUILDER

SQL (Structured Query Language) is a DSL language (domain-specific language) mostly used in a database system to manage data held in RDBMS (relational database management system). Because of its popularity, recent usages were also in stream processing applications like Kafka (KSQL) to provide a convenient way of manipulating big data. It is useful mainly in the handling of structured data and querying data across many tables based on some common variables.

The purpose of PyPika is to be an open-source SQL builder written in Python and expose the full richness of the SQL language via object-oriented programming. The goal of PyPika is to provide a simple interface for building SQL queries without limiting the flexibility of handwritten SQL. Because companies usually do not provide you with clean, training-and-evaluation-ready datasets, strong SQL skills are needed to extract them from their databases. Usage of query builders allows us to re-use existing codebases and extract information with only little modifications.

11.3.1 BITWISE AND SUPPORT

Bitwise operation manipulates numerals at the level of their bits. A bitwise AND operation takes two binary representations of equal length and performs the logical AND operation on each pair of their corresponding bits. This is equivalent to multiplying them and visualized in table 22.

In the database system, this can be used for efficient filtering, where instead of N individual columns, we have just one, with N bits.

Table 22: Bitwise AND visualization

	0	1	1
AND	0	0	1
RESULT	0	0	1

This functionality was, however, not present in the PyPika, so in the spirit of open source, instead of hardcoded SQL, we developed it in [25].

11.4 SFASTTEXT

FastText is an open-source library for fast text representation and classification. It was created by Facebook's AI Research (FAIR) lab. The model allows to create unsupervised learning or supervised learning algorithm for obtaining vector representations. They are written in C++ and with official bindings for usage in Python. While Swift allows Python incompatibility simply by writing Python.import("fasttext"), this results in performance drop and bad pointer exceptions in a multi-threaded environment, because of GIL implementation in Python.

SFastText is an open-source wrapper around FastText hosted at Github, developed by us. Providing all basic functionality needed to train, save, load, and evaluate the embedding. [27].

12 PRODUCTION

In the production environment, the machine learning model that returns the probability between a pair of product and offer is an only small part of the whole ecosystem. The whole process from acquiring data up to the inferencing probabilities consists of several components exchanging data between each other. It creates a DAG (Directed Acyclic Graph) where some steps can be even executed in parallel. There exist many utilities that aim to orchestrate machine learning workflow, where MLflow [43] and KubeFlow [35] seem to be the most promising ones.

In the near future, we aim to investigate them more closely and create an automatic end-to-end system that could do most of the required steps like obtaining training data, training, hyperparameter search, testing accuracy, and finally serving best model.

13 CONCLUSION

Even with all signals discussed in this work, it is still unrealistic to decide between two products in some cases. For example, two products that distinguish in token "adult" are the same, where another two products that distinguish in token "puppy" are different, and there is no rule about when it happens. However, the system has very good results so far.

In the future, observation of product descriptions could provide very good results in such a situation as well as the similarity between product images. The product that has only small differences in titles are often very different in visual, for example, by entirely different packaging or at least color.
Also, with continuous improvement of the product catalog, there will be more quality training data that we can train and optimize our system on. With the usage of tools like KubeFlow, we will be able to create semi-automatic that could be easily tweaked around and perhaps gives us even more accurate results.

14 APPENDICES

14.1 TRAINING TRACKING

14.1.1 TELEGRAM

While training is being done on the cloud servers, one needs to keep track of results. Executing SSH into several machines one by one is unnecessary time consuming and could lead to mistakes, like accidentally quitting training instead of logging out. We developed a simple library for notifications via Telegram, providing a summary of current training results as well as elapsed time or best-achieved validation accuracy.

14.1.2 MLFLOW

MLFlow is an open source platform for the machine learning life cycle. In section 12, we introduced an idea of using it for creating a pipeline of the entire model life cycle. Currently, we use its metrics UI to log and compare runs with different parameters, as shown in figure 25.

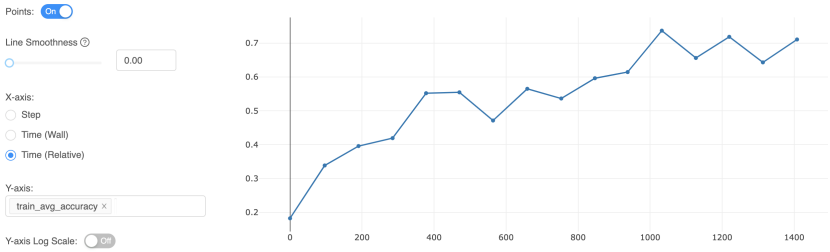

Figure 25: MLFlow tracking UI

References

[1] Walmart Ajinkya More. *Product Matching in eCommerce using deep learning*. 2017. URL: https://medium.com/walmartlabs/product-matching-in-ecommerce-4f19b6aebaca.

[2] J urgen Schmidhuber Alex Graves; Marcus Liwicki; Santiago Fern andezRoman Bertolam; Horst Bunk; *A Novel Connectionist System for UnconstrainedHandwriting Recognition*. 2008. URL: http://people.idsia.ch/~juergen/tpami_2008.pdf.

[3] Ben Athiwaratkun; Andrew Gordon Wilson; Anima Anandkumar. *Probabilistic FastText for Multi-Sense Word Embeddings*. 2014. URL: https://www.aclweb.org/anthology/P18-1001.pdf.

[4] Konstantin B Bulatov et al. "Reducing overconfidence in neural networks by dynamic variation of recognizer relevance." In: *ECMS*. 2015, pp. 488–491.

[5] Kris Cao and Marek Rei. "A joint model for word embedding and word morphology". In: *arXiv preprint arXiv:1606.02601* (2016).

[6] Cimri Cenk Çorapcı. *Product Matching with Deep Learning*. 2017. URL: https://medium.com/@cenk.corapci/product-matching-with-deep-learning-49d868c54fdb.

[7] R.; LeCun Y. Chopra S.; Hadsell. *Learning a similarity metric discriminatively, with application to face verification*. 2005. URL: https://en.wikipedia.org/wiki/Special:BookSources/0-7695-2372-2.

[8] Zhiyong Cui et al. "Deep bidirectional and unidirectional LSTM recurrent neural network for network-wide traffic speed prediction". In: *arXiv preprint arXiv:1801.02143* (2018).

[9] Sounak Dey et al. "Signet: Convolutional siamese network for writer independent offline signature verification". In: *arXiv preprint arXiv:1707.02131* (2017).

[10] Sounak Dey et al. "Signet: Convolutional siamese network for writer independent offline signature verification". In: *arXiv preprint arXiv:1707.02131* (2017).

[11] Marco Dinarelli and Loıc Grobol. "Seq2biseq: Bidirectional output-wise recurrent neural networks for sequence modelling". In: *arXiv preprint arXiv:1904.04733* (2019).

[12] Ceska Ecommerce. *Velikost e-commerce trhu*. 2020. URL: https://www.ceska-ecommerce.cz/.

[13] Inc. Facebook. *FastText dimension reduction*. URL: https://github.com/facebookresearch/fastText/blob/master/python/fasttext_module/fasttext/util/util.py#L94.

[14] Kunihiko Fukushima. *Cognitron: A self-organizing multilayered neural network*. 1975. URL: https://link.springer.com/article/10.1007/BF00342633.

[15] Google. *FaceNet: A Unified Embedding for Face Recognition and Clustering*. 2016. URL: https://research.google/pubs/pub45610/.

[16] Sorin Grigorescu et al. "A survey of deep learning techniques for autonomous driving". In: *Journal of Field Robotics* (2019).

[17] Jiuxiang Gu et al. "Recent advances in convolutional neural networks". In: *Pattern Recognition* 77 (2018), pp. 354–377.

[18] Chuan Guo et al. "On calibration of modern neural networks". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1321–1330.

[19] Chuan Guo et al. "On calibration of modern neural networks". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1321–1330.

[20] Rosie Hood. *https://towardsdatascience.com/unravelling-product-matching-with-ai-1a6ef7bd8614*. 2019. URL: https://towardsdatascience.com/unravelling-product-matching-with-ai-1a6ef7bd8614.

[21] Facebook Inc. *Library for efficient text classification and representation learning*. URL: https://fasttext.cc/.

[22] Facebook Inc. *Pretrained FastText vectors*. 2018. URL: https://fasttext.cc/docs/en/crawl-vectors.html.

[23] Ian T. Jolliffe and Jorge Cadima. *Principal component analysis: a review and recent developments*. 2016. URL: https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202.

[24] Peter Jung. *Bidirectional Recurrent Layers in S4TF*. 2020. URL: https://github.com/tensorflow/swift-apis/pull/946.

[25] Peter Jung. *Bitwise and support*. 2019. URL: https://github.com/kayak/pypika/pull/292.

[26] Peter Jung. *Executable for VGG-Imagewoof*. 2020. URL: https://github.com/tensorflow/swift-models/pull/301.

[27] Peter Jung. *Facebook FastText support in Swift*. 2020. URL: https://github.com/kongzii/SFastText.

[28] Peter Jung. *Make 'Tensor.product(squeezingAxes:)' differentiable*. 2019. URL: https://github.com/tensorflow/swift-apis/pull/550.

[29] Peter Jung. *S4TF bug in GRU*. 2020. URL: https://github.com/tensorflow/swift-apis/issues/936.

[30] Peter Jung. *S4TF fix GRU implementation*. 2020. URL: https://github.com/tensorflow/swift-apis/pull/960.

[31] Peter Jung. *S4TF test RNN by reference*. 2020. URL: https://github.com/tensorflow/swift-apis/pull/950.

[32] Kaggle. *Kaggle Product Review Classification*. 2013. URL: https://www.kaggle.com/c/product-review-classification.

[33] *Kernel principal component analysis*. URL: https://en.wikipedia.org/wiki/Kernel_principal_component_analysis.

[34] Yoon Kim et al. "Character-aware neural language models". In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[35] KubeFlow. *KubeFlow*. 2020. URL: https://kubeflow.org/.

[36] Mingchen Li, Mahdi Soltanolkotabi, and Samet Oymak. "Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks". In: *arXiv preprint arXiv:1903.11680* (2019).

[37] Xiangang Li and Xihong Wu. "Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2015, pp. 4520–4524.

[38] Yitan Li et al. "Word embedding revisited: A new representation learning and explicit matrix factorization perspective". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

[39] Amit Mandelbaum and Adi Shalev. "Word embeddings and their use in sentence classification tasks". In: *arXiv preprint arXiv:1610.08229* (2016).

[40] Jeffrey Pennington; Richard Socher; Christopher D. Manning. *GloVe: Global Vectors for Word Representation*. URL: https://nlp.stanford.edu/pubs/glove.pdf.

[41] Tomas Mikolov et al. “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[42] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[43] MLFlow. *MLFlow*. 2020. URL: https://mlflow.org/.

[44] Ajinkya More. “Attribute extraction from product titles in ecommerce”. In: *arXiv preprint arXiv:1608.04670* (2016).

[45] Ecommerce News. *The Czech Republic is fastest-growing ecommerce market in Europe*. 2019. URL: https://ecommercenews.eu/the-czech-republic-is-fastest-growing-ecommerce-market-in-europe/.

[46] Bromley Jane; Guyon Isabelle; LeCun Yann; Säckinger Eduard; Shah Roopak. *Signature verification using a "Siamese" time delay neural network*. 1994. URL: https://papers.nips.cc/paper/769-signature-verification-using-a-siamese-time-delay-neural-network.pdf.

[47] Sepp Hochreiter; Jurgen Schmidhuber. *LONG-SHORT TERM MEMORY*. 1997. URL: https://www.bioinf.jku.at/publications/older/2604.pdf.

[48] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “Facenet: A unified embedding for face recognition and clustering”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[49] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “Facenet: A unified embedding for face recognition and clustering”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[50] Jane Bromley; Isabelle Guyon; Yann LeCun; Eduard Sickinger; Roopak Shah. *Signature Verification using a "Siamese" Time Delay Neural Network*. 1994. URL: https://papers.nips.cc/paper/769-signature-verification-using-a-siamese-time-delay-neural-network.pdf.

[51] Zi Yin; Yuanyuan Shen. *On the Dimensionality of Word Embedding*. 2018. URL: https://papers.nips.cc/paper/7368-on-the-dimensionality-of-word-embedding.pdf.

[52] Yaniv Taigman et al. “Deepface: Closing the gap to human-level performance in face verification”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1701–1708.

[53] *Triplet Loss*. URL: https://en.wikipedia.org/wiki/Triplet_loss.

[54] Kelly L Wiggers et al. “Image retrieval and pattern spotting using siamese neural network”. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–8.

[55] Wikipedia. *Damerau Levenshtein distance*. 2020. URL: https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance.

[56] Wikipedia. *EAN*. 2020. URL: https://en.wikipedia.org/wiki/International_Article_Number.

[57] Wikipedia. *Go (programming language)*. 2020. URL: https://en.wikipedia.org/wiki/Go_(programming_language).

[58] Wikipedia. *Gradient boosting*. URL: https://en.wikipedia.org/wiki/Gradient_boosting.

[59] Wikipedia. *The Barlett test*. 2020. URL: https://en.wikipedia.org/wiki/Bartlett%27s_test.

[60] Wikipedia. *The Dixons Q test*. 2020. URL: https://en.wikipedia.org/wiki/Dixon%27s_Q_test.

[61] Wikipedia. *The Grubbs test*. 2020. URL: https://en.wikipedia.org/wiki/Grubbs%27s_test_for_outliers.

[62] Petar Ristoskia; Petar Petrovskia; Peter Mikab; Heiko Paulheim; Yahoo. *A Machine Learning Approach for ProductMatching and Categorization*. 2016. URL: http://www.heikopaulheim.com/docs/swj2018_product_data.pdf.

[63] Gregory Koch; Ruslan Salakhutdinov; Richard Zemel. *Siamese Neural Networks for One-shot Image Recognitio*. URL: https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf.

[64] Jinshan Zeng; Min Zhang and Shao-Bo Lin. *Fully-Corrective Gradient Boosting with Squared-Hinge: Fast Learning Rates and Early Stopping*. 2020. URL: https://arxiv.org/pdf/2004.00179.pdf.

[65] Guineng Zheng et al. "OpenTag: Open Attribute Value Extraction from Product Profiles [Deep Learning, Active Learning, Named Entity Recognition]". In: *arXiv preprint arXiv:1806.01264* (2018).