A HYBRID APPROACH FOR LARGE-SCALE PRODUCT CATEGORIZATION BASED ON WEIGHTED KNN AND LSTM-BPV

HAOHAO HU

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

MASTER OF ARTS

GRADUATE PROGRAM IN INFORMATION SYSTEMS AND TECHNOLOGY
YORK UNIVERSITY
TORONTO, ONTARIO
AUGUST 2019

Abstract

In modern e-commerce systems, large volumes of new items are being added to the product list everyday, which calls for automatic product categorization. In this thesis we propose a weighted K-Nearest Neighbour (KNN) based classification system for solving large-scale e-commerce product taxonomy classification problem. We use information retrieval (IR) model as similarity function in our weighted KNN algorithm. Among all IR models used in this study, we achieved highest classification performance through using information-based (IB) model as similarity function in the KNN algorithm. Moreover, our proposed method can improve the overall performance when combining prediction results with those from advanced neural network based method, namely Long Short-Term Memory with Balanced Pooling Views (LSTM-BPV). The hybrid system could achieve results comparable to the state of the art (SotA). We also get good results by fine-tuning pre-trained Bidirectional Encoder Representations from Transformers (BERT) model.

Acknowledgements

I would like to thank the supervisor of this thesis Professor Jimmy Xiangji Huang for his guidance and continuous support. Without his help, this thesis would not be completed and I would not have the opportunities to participate in various learning activities related to information technology. I would also like to thank Dr. Xing Tan for giving me helpful advice. I would also like to thank Runjie Zhu, Yuqi Wang and Professor Wenying Feng (from Trent University) for collaborating with me in our preliminary work in 2018 SIGIR eCom Data Challenge. I would also like to thank Professor George Georgopoulos and Professor Xiaohui Yu for being a member of my supervisory committee. Additionally, I would like to thank my colleagues in Information Retrieval and Knowledge Management (IRKM) Laboratory for their help and support.

In addition, I gratefully acknowledge the support by NSERC (Natural Sciences and Engineering Research Council of Canada) CREATE (Collaborative Research and Training Experience Program) award in ADERSIM (Advanced Disaster, Emergency and Rapid-response Simulation), ORF-RE (Ontario Research Fund - Research Excellence) award in BRAIN (Big Data Research, Analytics, and Information Network) Alliance, and the York Research Chairs Program.

Additionally, I would like to thank Michael Skinner and Google AI Language Team for making the implementation codes of LSTM-BPV and BERT publicly available. I would also like to thank Rakuten Institute of Technology Boston (RIT-Boston) for organizing 2018 SIGIR eCom Data Challenge and providing me with the SIGIR eCom Data Challenge Dataset and evaluation script.

Finally, I would like to thank my family for continuous love and support.

Table of Contents

List of Tables

List of Figures

Abbreviations

Here is a list of abbreviations used in this thesis:

AbLSTM: Attention-based LSTM

AI: Artificial Intelligence

BERT: Bidirectional Encoder Representations from Transformers

BLSTM: Bidirectional LSTM

B-LSTM-BPV: Bidirectional ensemble of a forward LSTM-BPV and a backward LSTM-BPV

BM25: Best Match 25

BoW: Bag-of-Words

CAN: Collaborative and Adversarial Network

CA-RNN: Context-aligned RNN

CLSM: Convolutional Latent Semantic Model

CNN: Convolutional Neural Network

CoLA: Corpus of Linguistic Acceptability

CRTER: CRoss TERm

CV: Computer Vision

DeepCN: Deep Categorization Network

DF: Document Frequency

DNN: Deep Neural Network

DSSM: Deep Structured Semantic Models

ELMo: Embeddings from Language Models

F1: F1 score

GloVe: Global Vectors

GPT: Generative Pre-Training

HBM: High Bandwidth Memory

IB model: Information-based model

IDF: Inverse Document Frequency

IR: Information Retrieval

KNN: K-Nearest Neighbour

K-NRM: Kernel based Neural Ranking Model

LL: Log-Logistic

LLSF: Linear Least-Square Fit

LM: Language Model

LSA: Latent Semantic Analysis

LSTM: Long Short-Term Memory

LSTM-BPV: Long Short-Term Memory with Balanced Pooling Views

LtR: Left-to-Right

L2R: Learning-to-Rank

ML: Machine Learning

MLE: Maximum Likelihood Estimator

MLM: Masked Language Model

MLP: MultiLayer Perceptron

MT: Machine Translation

NIST: National Institute of Standards and Technology

NLM: Neural Language Model

NLP: Natural Language Processing

NSP: Next Sentence Prediction

1NN: 1-Nearest Neighbour

OOV: Out-of-Vocabulary

P: Precision

POS: Part-of-Speech

PRF: Pseudo-Relevance Feedback

PV-DBOW: Paragraph Vector Distributed Bag of Words

PV-DM: Paragraph Vector Distributed Memory

R: Recall

RNN: Recurrent Neural Network

RSV: Retrieval Status Value (or relevance score)

R2L: Right-to-Left

SGD: Stochastic Gradient Descent

SGDM: SGD with (Nesterov's) Momentum

SGDR: SGD with warm Restarts

SIGIR:Special Interest Group on Information Retrieval

SotA: State of the Art

SPL: Smoothed Power-Law

SST-2: Stanford Sentiment Treebank (with 2 target classes)

SVD: Singular-Value Decomposition

SVM: Support Vector Machine

TextCNN: Text Convolutional Neural Network

TF: Term Frequency

TPU: Tensor Processing Unit

TREC: Text REtrieval Conference

TTF: Total Term Frequency (or collection frequency)

ULMFiT: Universal Language Model Fine-Tuning

UNSPSC: United Nations Standard Product and Service Code

VM: Virtual Machine

VSM: Vector Space Model

Chapter 1

Introduction

In this chapter, we will introduce the background, motivation, problem definition and main contributions of our research. We will also present an outline of the remaining part of this thesis.

1.1 Background

As the fast-paced development of the internet, there has been a huge rise of the e-commerce market. Online shopping platforms, such as Alibaba, Amazon and Taobao, provide not only goods meeting consumers' specific needs, but also products that are basically everyone's daily consumption in life. Almost all the e-commerce platforms aim at updating their shopping lists and inventories at their fastest speed to target certain consumers in order to win a bigger proportion of the market. Also, e-commerce websites usually classify products into a taxonomy tree consisting of multiple levels. A product taxonomy tree typically contains more than 1000 leaf nodes, i.e. classes. So, the technologies adopted to efficiently and effectively categorize a product's category within taxonomy tree become more important. This would help the system operators or merchants to add in or delete certain items from their product lists. It would also be easier for system operators or managers to deal with data analysis and data management in future.

1.2 Motivation and Research Problems

The motivation of this research is to combine recently developed information retrieval (IR) models and conventional KNN approach for efficient and effective e-commerce product taxonomy classification. Compared to other methods like Support Vector Machine (SVM), it would be relatively easy to incorporate our approach into modern e-commerce websites, as they already have product search systems. Also, we would like to compare the effectiveness of different IR models through using them as similarity function in our KNN algorithm.

Problem Definition: The problem to solve in this thesis is to predict a product's category id path c_i in a product taxonomy tree, which consists of N possible category id paths $\{c_1, \ldots, c_N\}$, given the product's title pt_i.

Large-scale product taxonomy classification has the following challenges:

The first challenge is that there are many classes (leaf nodes) in the taxonomy. For example, the SIGIR eCom Data Challenge Dataset have 3,008 target classes (distinct category id paths). This makes computational cost high, especially for more complex algorithms (e.g. Support Vector Machine (SVM)). In contrast, our proposed method generally involves less computational cost, because it is instance-based and thus its computational cost is independent of the number of classes.

The second challenge is that the product titles have various lengths. As an example, according to [50], the product titles in the SIGIR eCom Data Challenge Dataset have minimum and maximum word-level length of 1 and 58 respectively. This makes it less efficient to be tackled with neural network approaches, since excessive padding (i.e. "0") has to be used so that examples within a mini-batch or even training set are of equal length. In contrast, our proposed method requires no padding and thus is more efficient.

The third challenge is that there is possibility of error during manual labelling of a product's class. This is partly because, according to [50], labelling is usually done

by merchants in an e-commerce setting. Also, there is no strict rule/gold standard in labelling. This problem lies in the dataset itself, and we could address it by providing merchants with more product labelling examples deemed correct by experts.

The fourth challenge is that the size of product catalog is usually quite large. For example, according to [50], the size of Rakuten's product catalog is much greater than 1 million. Because of such heavy workload, we need relatively good and fast product classification algorithms to help merchants label products more efficiently and effectively. Our proposed KNN method is fast, relatively good and scalable, where new training instances could be easily added into the weighted-KNN system to improve classification performance.

The fifth challenge is that the distribution of products over category is skewed/unbalanced. For example, top categories/classes may have over 10,000 product titles, while many other classes may only have less than 100 product titles. Such unbalanced data distribution could be tackled with oversampling or undersampling. In our proposed method, we address this issue with weighted-KNN algorithm, where the category with highest category score instead of highest number of instances within top k training products ($\{pt_{i(1)}, \ldots, pt_{i(k)}\}$) most similar to a given test product title t_i is deemed its predicted category c_i.

1.3 Main Contributions

The main contributions of our thesis are listed as follows:

Firstly, we introduced an end-to-end weighted KNN-based system for large-scale e-commerce product taxonomy classification. The weighted KNN algorithm in the system uses IR model as similarity function. The system could serve as a fast and relatively good baseline. Also, the system would probably achieve better results with more recently developed neural IR models like that in [8] as similarity function in KNN.

Secondly, we compared and analysed performance of classification on the SIGIR eCom Data Challenge Dataset using different IR models (i.e. Best Match (BM) 25 [78], Language Models (LM) [104], Vector Space Model (VSM) with TF-IDF Weight, Information-based (IB) Models [10] and cosine similarity of Doc2vec embeddings [46]) as similarity function in our weighted-KNN system. In a sense, this serves as a benchmark of the effectiveness of these IR models. Among these IR models, we found that IB Model and cosine similarity of Doc2vec embeddings obtained the highest and lowest classification performance respectively.

Thirdly, we used ensemble of our approach and SotA LSTM-BPV(s) [84], and we could improve the overall classification performance of constituent LSTM-BPV(s) by around 0.5% in terms of weighted-F1 score (weighted-F1). This suggests that word-level matching in our KNN system is helpful for character-level neural network. In a sense, this also shows the usefulness of our weighted-KNN system. We also applied fine-tuning technique to boost the classification performance of LSTM-BPV(s) and ensembles.

Lastly, we conducted experiments to get better performance on product classification by fine-tuning the pre-trained BERT-large uncased model [15]. To the best of our knowledge, this is the first attempt at applying the technique of fine-tuning pre-trained neural language model (NLM) on large-scale product classification task. The model is able to obtain good performance after being fine-tuned for 40 epochs. However, although it could achieve similar results with the ensembles above, we also found this method requires high computational cost and generalizes slowly on this product categorization task. Furthermore, we examined the effect of using larger batch size to fine-tune the BERT model and confirmed that using smaller batch size can improve training performance.

1.4 Outline

The remaining part of this thesis is organized as follows: we will first introduce technical preliminaries related to our study and compare related work with our work in Chapter 2. After that, we will describe our proposed method in Chapter 3. This is followed by experiments in Chapter 4, result analyses and discussions in Chapter 5 and conclusion in Chapter 6. A part of the SIGIR eCom Data Challenge Training Dataset is shown in Appendix A. The instructions for running our KNN classification system based on Lucene are included in Appendix B. We also put the instructions of our KNN classifier based on Gensim Doc2vec in Appendix C. The instructions for doing experiments with the hybrid classification system based on our KNN algorithm and LSTM-BPV(s) are included in Appendix D. We also included the instructions of the classification system based on pre-trained BERT-large uncased model in Appendix E.

Chapter 2

Preliminaries

In this chapter, we will introduce the technical preliminaries relating to our work and compare our work with other existing work most related to our work. We will also present an overview of top participants' approaches to solving the e-commerce product taxonomy classification problem in 2018 SIGIR eCom Data Challenge.

2.1 Text Classification

Product taxonomy categorization is a subtask of text classification, which is a classic task in machine learning (ML) and artificial intelligence (AI). The task has many real-world applications, such as spam email detection, sentiment analysis (or opinion mining) and news categorization. Generally, text classification is to automatically classify a text document $D_i \in C$ (C is the set of documents or document collection, $C = \{D_1, \ldots, D_{|C|}\}$) as one predefined category (or class) $cls_i \in Y$ (Y is the set of all predefined classes $Y = \{cls_1, \ldots, cls_{|Y|}\}$). For multi-label text classification, such as twitter classification, a text document is classified as one or more classes. Generally, text documents need to be transformed into useful numerical vector representation for classification. A wide variety of methods have been proposed and adopted so far for text classification, including classic methods such as KNN [96, 24], SVM [38], Linear Least-Square Fit (LLSF) [97],

decision tree [11] and Bayesian Classifier [58, 76]. More specifically, proposed in 1994, LLSF [97] tries to find an optimal transformation between bag-of-words (BoW) document representations $\mathbf{D}_i$ and their corresponding category representations $\mathbf{y}_i$. In 1998, [38] proposed to use SVM for classifying text. They found that SVMs could outperform all existing methods, including KNN. Their experiments were conducted on 2 datasets and the higher number of target classes within these datasets is 90. Also, in recent years, neural network based methods, such as fastText [41], Text Convolutional Neural Network (TextCNN) [42] and Recurrent Neural Network (RNN) [92, 12], have been used in text classification and set new records in this task. A recent trend for text classification is to pre-train a NLM on free text (unsupervised learning) and then fine-tune it for classification (supervised learning) [15, 30, 99]. As a brief introduction, a NLM learns to predict a word based on its context. For example, a right-to-left (R2L) NLM learns to predict a word x_i based on words after it within a text sequence $[x_{i+1}, \ldots, x_n]$. As language modelling only requires readily available unlabelled text as training data, it helps to reduce human labour.

2.2 Neural Network Approaches for Text Classification

Here we give a brief overview of popular neural network methods for text classification:

LSTM classifier is based on Long Short-Term Memory (LSTM) [29], a refined RNN [40]. RNN has been shown to obtain good performance on sequential input, such as time series. LSTM is better at tackling gradient vanishing problem than traditional RNN. In recent years, LSTM [29] gained its popularity in text classification because of its relative simplicity and ability to better capture long term dependency in text sequence than traditional RNN. Generally, in LSTM classifier, each word or character in a sentence is mapped to a word/character embedding as input to LSTM layer(s) sequentially. The last layer of the classifier is a classification layer. Bidirectional LSTM (BLSTM) [21] is a

variant of LSTM. It performs better than LSTM, because it is trained on 2 directions, i.e. the forward and reverse direction, whereas LSTM is trained on 1 direction. The output of BLSTM layer is the concatenation of final hidden states in both directions. Later, various modifications were applied to LSTM for better classification performance: In 2015, word-level LSTM was applied to 5 text classification benchmarks for the first time and it obtained relatively good results [12]. In addition, [12] experimented with character-level LSTM for classifying Wikipedia pages. In 2016, [39] proposed "region embedding + pooling" based on LSTM with one-hot word vector as input (i.e. no word embedding layer). The LSTM emits a hidden state at each time step of a document, and max-pooling over these hidden states was used to get the final document representation. [39] also proposed other simplifications of the model to accelerate training, such as removing input/output gates of LSTM. In 2018, [30] proposed concat pooling, which involves augmenting the last hidden state of LSTM layer with max-pooling and mean-pooling representations of intermediate hidden states. According to [30], concat pooling's purpose is to provide more useful information for classification, as the key words for classification can occur in any place of a document and just the final hidden state is not enough because of possible information loss. Later, in [84], balanced pooling views (BPV) was proposed. Compared to concat pooling [30], BPV [84] has additional concatenation of min-pooling representations in the output of LSTM layer, which results in better performance of classification.

TextCNN [42] is based on convolutional neural network (CNN) [47]. CNN is good at capturing local dependencies. The TextCNN network consists of embedding layer that maps input words to word embeddings, 1-Dimensional convolutional layer with multiple filters (which are similar to word n-grams), max pooling layer, fully connected layer and softmax (or classification) layer.

FastText classifier [41] is a 2-layer shallow neural network and its input is a sentence.

Multiple features within a sentence, i.e. character n-grams and word n-grams, are turned into dense vectors and averaged to produce the sentence representation in hidden layer. As a brief introduction, a character n-gram is formed by n continuous characters within a word. For example, "nufa" is a character 4-gram in "manufacture". Similarly, a word n-gram is formed by n continuous words within a sentence. For example, "sleeping" and "cat is" are word unigram and bigram in "A cat is sleeping" respectively. The use of character n-gram helps to capture morphological information, so the classifier would become more tolerant to typos like "enviroment". Additionally, using word n-gram can help to capture local dependencies. The sentence representation is then fed into a softmax classifier. The network can be trained asynchronously using multiple CPUs.

Bidirectional Encoder Representations from Transformers (BERT) model [15], based on Transformer [91] architecture, is a pre-trained NLM. Transformer model, originally proposed for tackling machine translation (MT) problem, is a recently developed neural network approach that uses self-attention mechanism instead of conventional convolution/recurrent mechanism to capture local/long-term dependencies. Self-attention models inner dependencies of a sequence of text, i.e. queries $\mathbf{q_i} \in \mathbb{R}^{d_k}$, keys $\mathbf{k_i} \in \mathbb{R}^{d_k}$ and values $\mathbf{v_i} \in \mathbb{R}^{d_v}$ (specifically for Transformer, $\mathbf{k_i} = \mathbf{v_i}$) all come from the embedding representation of same text sequence $X = [\mathbf{x_1}, \ldots, \mathbf{x_n}]$ (but this sequence can be 2 sentences for sentence pair classification). Specifically, Transformer uses multi-head attention consisting of several scaled dot-product attention in parallel. The raw attention weight for $\mathbf{v_i}$ given $\mathbf{q_j}$ is calculated as follows:

$$rw(\mathbf{v_i}, \mathbf{q_j}) = \frac{\mathbf{q_j}.\mathbf{k_i}}{\sqrt{d_k}}, \tag{2.1}$$

where . is dot product. BERT can be modified to do text classification by adding a classification layer on the top of it. The input embedding is the sum of WordPiece embedding, segment embedding and position embedding. Different from OpenAI GPT [73], a pre-trained left-to-right (LtR) NLM based on Transformer architecture, where the LtR NLM

is trained to only predict a word x_i based on its previous words $[x_1, \ldots, x_{i-1}]$ within a text sequence, the BERT [15] is pre-trained to predict a word based on other words within a sentence, which is called masked language model (MLM). The BERT model is also pre-trained to predict whether two sentences drawn from the training text are adjacent sentences within a document or not, which is called next sentence prediction (NSP).

2.3 Document Representation

Both text classification and IR requires representing a document as useful numerical features such as a vector or a matrix. Different methods may use different document representations. They are listed as follows:

2.3.1 Traditional Bag-of-words Representation

Bag-of-words (BoW) representation is widely adopted as inverted index in search systems and utilized in many classic text classification methods, such as KNN, LLSF [97], Naive Bayes Classifier and SVM. Each document D_j is mapped to a document vector $\mathbf{D_j}$ of dimension $|V_C|$ (size of vocabulary or number of distinct terms in document collection C)($\mathbf{D_j} \in \mathbb{R}^{|V_C|}$). In this thesis, we use "term" and "word" interchangeably. A term/word, containing 1 or more characters, is the basic building block of a text document. The TF-IDF weight [79] is a common feature used in BoW document representation. So, a document D_j can be represented as:

$$\mathbf{D_j} = (tfidf_{1j}, \ldots, tfidf_{|V_C|j}), \tag{2.2}$$

where $tfidf_{ij} = TF(w_i, D_j) \times IDF(w_i) = TF(w_i, D_j) \times \log\left(\frac{N}{DF(w_i)}\right)$ ($TF(w_i, D_j)$ is the term frequency (TF) (or number of occurrences) of term w_i in D_j, $IDF(w_i)$ is the inverse document frequency (IDF) of w_i, N is the number of documents in document collection C, $DF(w_i)$ (document frequency (DF)) is the number of documents containing

w_i in C):

$$DF(w_i) = |\{D_j \in C | w_i \in D_j\}| \tag{2.3}$$

So, a term-document matrix M would look like this:

$$M = \left[\mathbf{D'_1} \quad | \quad \cdots \quad | \quad \mathbf{D'_j} \quad | \quad \cdots \quad | \quad \mathbf{D'_N}\right], \tag{2.4}$$

where $\mathbf{D'_j}$ is the transpose of $\mathbf{D_j}$. Apart from TF-IDF weight, other IR models, such as BM25, can also be used to represent a document as bag of words. $|V_C|$ is usually very large in modern search systems, typically over 1 million. An example of query and documents in VSM is shown in Figure 2.1. Although feature selection can be used to reduce the dimension, the computational cost is still high due to large feature size. Also, the limitation of such representation is that the positional information of words in a document is ignored. In the following two kinds of representations, such positional information can be retained if character/word vectors are concatenated.

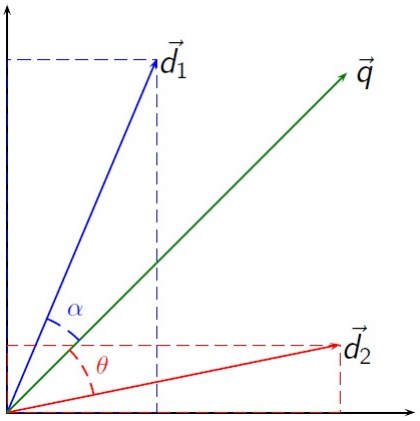


Figure 2.1: Example of documents and query on VSM, by Riclas - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=9076846

2.3.2 Sparse One-hot Character/Word Vectors

A one-hot vector is a vector that has only one 1 and all others are 0, e.g. $(0, 1, 0, 0, 0)$. For sparse one-hot character vector, each unique character c_i is represented as a unique one-hot vector $\mathbf{c_i} \in \mathbb{R}^{|V_{char}|}$ ($|V_{char}|$ is the size of character-level vocabulary or number of distinct characters in document collection C). Similarly, for sparse one-hot word vector, each unique word w_i is represented as a unique one-hot vector $\mathbf{w_i} \in \mathbb{R}^{|V_C|}$ ($|V_C|$ is the size of word-level vocabulary or number of distinct words in document collection C). As such character/word vectors in a document come in sequence, they can be concatenated to become the document's representation. The downside of sparse word vector is that the dimension $|V_C|$ can be very large, which leads to high computational cost and generally large memory requirement. In face of this problem, the following approach has been developed:

2.3.3 Dense/Distributed Character/Word/Paragraph/Document Vector

Distributed representations include dense character vector, dense word vector [65, 70], dense paragraph vector [46] and distributed document vector [14, 46]. A traditional approach to getting dense document vector/embedding is Latent Semantic Analysis (LSA) [14], where distributed document embeddings are derived from singular-value decomposition (SVD) on term-document matrix. Dimension of dense word vectors (i.e. word embeddings) is typically set within the range $[100, 500]$. Historically, distributed word embeddings were trained using a multiclass classifier like softmax. Generally, distributed word embedding had not gained popularity due to the high computational cost involved during training until in recent years researchers proposed fast and efficient methods for training it, such as hierarchical softmax [67, 65] and negative sampling [65]. In [65], word embeddings are trained using shallow neural networks in two ways, as shown in

Figure 2.2:

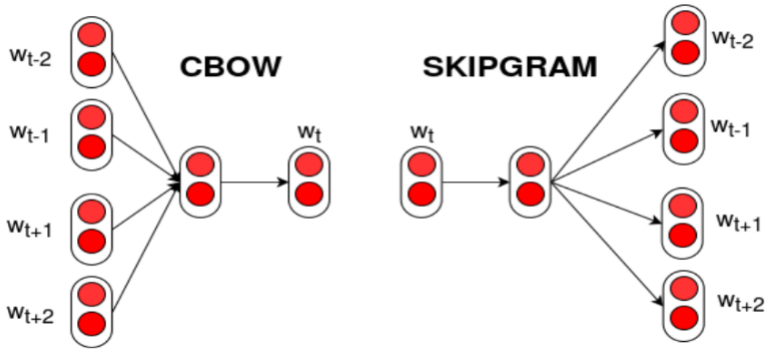


Figure 2.2: CBOW and Skipgram for training Word2vec word embedding, by Aelu-013 [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)], from Wikimedia Commons

The first way is Continuous Bag-of-Words (CBOW), where the surrounding words within the context (a user-defined window of size c around a word) of a word w_t, i.e. $[w_{t-c}, ..., w_{t-1}, w_{t+1}, ..., w_{t+c}]$, are trained to predict that word. The training objective is to maximize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_t|w_{t+j}), \tag{2.5}$$

where T is the number of words in the training corpus, c is the window size.

The second way is Skipgram, where a word w_t is trained to predict its surrounding words within its context, i.e. $[w_{t-c}, ..., w_{t-1}, w_{t+1}, ..., w_{t+c}]$. The training objective is to maximize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t), \tag{2.6}$$

where T is the number of words in the training corpus, c is the window size.

Since [65], more enhanced versions of word embedding have been proposed. For example, WordPiece embedding [93] is a good balance between word embedding and

character embedding. Similar to character n-gram features, rare words are split into sub-word units, i.e. word pieces. This helps to fix out-of-vocabulary (OOV) problem. Another example is fastText embedding [5], which is similar to the Skipgram word embedding in [65], with additional character n-gram features within a word that is helpful for capturing morphological information.

As for document embedding, recently [22] proposed a new document embedding approach: firstly, word embeddings are trained using Skipgram with negative sampling [65], then IDF values are calculated for all words. After that, they use K-means algorithm to cluster all words with their word embeddings. Each cluster's vector is the sum of word embeddings of words within the cluster, and its cluster frequency is the sum of IDF values of words within the cluster. After calculating cluster vectors and cluster frequencies, a document's vector is made by concatenating all its cluster vectors and cluster frequencies.

Distributed character/word vector is the backbone of many NLP tasks, including text classification. They are usually better than sparse one-hot character/word vectors, as they can capture semantics. Word embeddings of words of similar meaning usually have high cosine similarity. Distributed word/character embeddings are used in many text classification approaches, such as fastText [41] and TextCNN [42]. Distributed character/word/document embeddings can be learned in an unsupervised manner, which can benefit from large amounts of text data available on the Internet.

2.3.4 Contextualized Embeddings

More recently, contextualized embeddings have been developed to better capture contextual information within a sentence. They can be produced from LSTM encoder trained for MT [61], with GloVe vectors [70] as input. However, this is still supervised pre-training. As for unsupervised pre-training of contextualized embeddings, they can also be generated from NLMs based on LSTM [71] or Transformer [15] with character or

WordPiece embeddings as input, respectively. For example, Embeddings from Language Models (ELMo) [71] are derived from character-convolution-based bidirectional NLMs with two layers. Pre-trained contextualized embeddings like ELMo [71] can be integrated into existing approaches to improve performance.

2.4 Large-scale E-commerce Product Taxonomy Categorization

Large-scale product taxonomy classification has been studied since the rise of e-commerce websites. Properly categorizing a new product as a dynamically updated category within a taxonomy tree is of critical importance for e-commerce. Algorithms in support automated process for classification need to be straightforward, scalable and flexible to allow labelling errors and noises. The product taxonomy can actually be flattened for categorizing. Generally, there are two approaches to tackling such product taxonomy classification problem:

The first one is hierarchical classification, also referred to as "gate-and-expert" approach. The idea is to "divide and conquer". For example, [82] proposed a two-level classification: they first discover latent groups consisting of similar target classes through finding dense subgraphs within the confusion graph of all target classes, and then they train a coarse-level classifier (i.e. KNN) to classify items into those coarse groups. In each coarse group, a fine-level classifier (i.e. SVM) is used to classify items into target classes within the group. The approach calls for additional parameter tuning, i.e. the threshold value that controls the size of latent groups. Their method could obtain 0.754 in terms of accuracy in eBay Dataset (20,000+ classes). [22] also proposed a two-level approach utilizing an ensemble of classifiers at the coarse level.

The second approach is flat classification, also referred to as "end-to-end" classification, where classification is done in one system which uses raw text as input and generates predicted class as output. For example, in [44], a binary linear classifier model was

trained for each target category. The predicted probability of each class are compared and the one with highest probability is deemed the predicted class. Their best result was obtained (0.88 in terms of F1 in Yahoo Dataset (319 classes)) using the average of all dense word vectors [65] in a product title as feature for model training. [7] proposed to use multi-class SVM with margin re-scaling and loss normalization. Their experiments were conducted on United Nations Standard Product and Service Code (UNSPSC) dataset with 1,073 target classes. [7] also proposed a new evaluation metric for practical product taxonomy classification, namely average revenue loss. This metric takes both product revenue and distance of the true category and predicted category within product taxonomy into account. In [23], deep categorization network (DeepCN) was proposed. DeepCN consists of several word-level RNNs, one for each item metadata attribute (e.g. item name). The output of these RNNs is concatenated and then fed into 2 fully connected layers, followed by a classification layer. [23] found their approach was better than single RNN and BoW based Bayesian networks in terms of accuracy on a large dataset with 4,116 target classes.

2.5 KNN Classification

We chose the weighted KNN, a classic classification algorithm, as our major classification approach. KNN for classification can be traced back to as early as 1950s [86]. It is also referred to as lazy learning or instance-based learning. KNN is traditionally a simple algorithm that stores all the available candidates for classification, and it classifies each new candidate based on the similarity function. KNN algorithm is based on feature similarity measurement. Specifically, let Y be the set of all predefined classes:

$$Y = \{cls_1, \ldots, cls_{|Y|}\},$$

where $|Y|$ is the number of predefined classes. Let a be a data point we want to classify. a's ith nearest neighbour is $a_{(i)}$ whose class and weight is $y(a_{(i)})$ and $w_{a_{(i)}}$ respectively. The most intuitive KNN classifier is to set the $k = 1$, i.e. the 1-Nearest Neighbour (1NN) classifier which assigns point a to the class of its closest neighbour $a_{(1)}$ in the feature space,

$$\hat{y}(a) = y(a_{(1)}), \tag{2.7}$$

where $\hat{y}(a)$ is the predicted class of a, $y(a_{(1)})$ is a's first nearest neighbour's class.

Generally, for weighted KNN classification, the data point a is classified as the class with the largest added weight:

$$\hat{y}(a) = cls_j, \tag{2.8}$$

where

$$\sum_{i=1}^{k}(w_{a_{(i)}} \times \mathbf{1}_{\{y(a_{(i)})=cls_j\}}) = \max_{u\in\{1,\ldots,|Y|\}} \sum_{i=1}^{k}(w_{a_{(i)}} \times \mathbf{1}_{\{y(a_{(i)})=cls_u\}}), \tag{2.9}$$

where $\mathbf{1}_{\{y(a_{(i)})=cls_j\}}$ is an indicator function, which equals to 1 if $y(a_{(i)}) = cls_j$ and to 0 otherwise.

A wide range of methods have been proposed to weigh a given data point a's ith nearest neighbour $a_{(i)}$, e.g. based on their rankings i, the distances d between $a_{(i)}$ and a or their similarity scores Sim with a, etc.:

$$w_{a_{(i)}} = f(i, d(a, a_{(i)}), Sim(a, a_{(i)})), \tag{2.10}$$

Three variants of weighted KNN were used in our experiments, and they are listed as follows:

1. Weighted KNN: The weight for $a_{(i)}$ in Equation 2.9 is simply its similarity score with a:

$$w_{a_{(i)}} = Sim(a, a_{(i)}) \tag{2.11}$$

2. KNN with simple voting: KNN classifier (simple voting) can be considered as a special case of weighted KNN classifier where each of a's k nearest neighbours has a weight of $\frac{1}{k}$ and all others weigh zero. So, the weight for $a_{(i)}$ in Equation 2.9 is as follows:

$$w_{a_{(i)}} = \frac{1}{k} \tag{2.12}$$

3. Biweight kernel weighted KNN [28]: The weight for $a_{(i)}$ in Equation 2.9 is calculated using its similarity score with a and also the $(k+1)$th neighbour's similarity score with a:

$$w_{a_{(i)}} = \frac{15}{16}(1 - (\frac{Sim(a, a_{(k+1)})}{Sim(a, a_{(i)})})^2)^2 \tag{2.13}$$

This method considers the relative similarity so that the weights fall into the range $[0, \frac{15}{16})$.

Since 90s, KNN has been used in many text classification applications [59, 45, 98]. [98] conducted controlled experiments on a multi-label news story classification task to compare 5 classification approaches, namely SVM, KNN, neural network, LLSF [97] and Naive Bayes (NB) classifier. [98] found that although KNN is simple, it can perform as well as LLSF and SVM in news story classification. To tackle class imbalance, [52] used SMOTE method, where KNN search is used to generate new minority instances.

A wide range of weighting schemes of KNN have been proposed and applied so far. We adopted weighted KNN instead of KNN based on simple majority voting, as according to [80], weighting by similarity often outperforms simple voting. This was also confirmed in our experiments (we will compare different weighting schemes of KNN in Section 5.6).

2.6 IR

We use IR model as similarity function in our weighted KNN algorithm. IR is a general term used to describe the process of getting relevant documents in descending order of relevance from document collection based on a user's query. In terms of text retrieval, an IR model is a ranking function that assigns similarity (or relevance) score (retrieval status value) $RSV(D_i, Q_j)$ of a text document D_i and given text query Q_j.

There are 3 strands of IR model nowadays:

2.6.1 Hand-crafted IR Model

Hand-crafted IR models, such as BM25 [78], LM [104], VSM with TF-IDF [79] and IB Model [10], are ranking formulae designed by computer scientists. These models usually rank documents based on statistical features, such as TF, DF and document length. However, the common problem of the above models is that they use BoW representation that cannot take word positional information into account. Another problem is the keyword mismatch due to typo or synonyms. The above problems can be partially solved using character/word n-grams. Although using character n-grams can alleviate keyword mismatch due to typo, it is usually computationally expensive. Furthermore, although using word n-grams can partially capture word positional information and local dependencies, it is generally computationally expensive.

2.6.2 Traditional Learning-to-rank (L2R) IR Model

Traditional L2R IR models utilize ML methods to learn a ranking function based on training data. For example, [17] proposed Rank-SVM, similar to SVM, for multi-label text classification. The training data can be a set of tuples, each tuple consists of a query, a relevant text document and an irrelevant document. A L2R model can use multiple hand-

crafted IR models as query-document features. A L2R model can also use document features like document length and query features like query length.

2.6.3 Neural IR Model

Compared to traditional L2R IR models, neural IR models, based on neural networks, can usually capture the semantics and word positional information better. Such models usually use word/character/document embeddings for representing documents. For example, cosine similarity of Doc2vec document embeddings [46] was used for IR in [46], which we also use as similarity function in our KNN classification system. A neural IR model can use pre-trained word embeddings for document representation, which requires less computational cost compared to training them from scratch. For example, [81] proposed a L2R model for re-ranking sentences, and CNN was used for generating intermediate sentence representations based on pre-trained word embeddings.

Recent years have seen many studies of neural L2R IR models, partly because of popularity of neural network. According to [6], they can be categorized into 2 strands, i.e. distributed [32, 83, 66] or local-interaction [66, 95]. Generally, distributed IR models use cosine similarity of query vector and document vector for document ranking, while local-interaction IR models take word-by-word interaction of query and document into account.

In 2013, [32] proposed Deep Structured Semantic Models (DSSM). Different from previous latent semantic models, DSSM utilizes clickstream data for supervised learning and it consists of multiple nonlinear layers. A new technique to reduce the dimensionality of a BoW document vector, namely character n-gram based word hashing, was also proposed.

In 2014, [83] proposed convolutional latent semantic model (CLSM), which captures local contextual information of word n-grams and global contextual information of sen-

tence through convolution and max pooling respectively. In addition, the model uses clickstream data for training.

In 2016, [66] proposed Duet model that consists of both local-interaction model and distributed model. Both the 2 constituent models are based on deep neural network (DNN). The local-interaction model takes exact term match and proximity into account, while the distributed one captures semantic properties like synonyms. Duet model is able to perform better than its constituent models.

Later, in [95], kernel based neural ranking model (K-NRM) was proposed. Instead of exact matching as in traditional IR models, in K-NRM, semantic matching ("soft match") is achieved through the use of kernel pooling. This technique uses kernels to count word matches at different similarity levels and provides soft TF as features for ranking. Their experiments were also based on a query log.

A subtask of IR is to model similarity of sentences. In recent years, various neural L2R IR models have been proposed and adopted to model similarity of sentence pairs. [8] proposed a context-aligned RNN (CA-RNN) model. The model uses a novel context alignment gating mechanism for capturing contextual information of the aligned words in 2 sentences. Later, [9] proposed a Collaborative and Adversarial Network (CAN) based on LSTM. The model contains a novel common feature extractor, consisting of a generator that generates features from a sentence within a sentence pair and discriminator that learns to predict whether generated features come from the first sentence or the second sentence in the sentence pair.

2.7 IR Models Used in This Study

Here is a list of IR models we use as similarity measure in our weighted KNN algorithm:

2.7.1 BM25

BM25 is a well-known weighting function employed by the Okapi system [77]. As shown in previous TREC experiments, BM25 and its extensions provide very effective retrieval performance on the TREC collections [25, 78, 35, 33, 107]. As our work uses BM25 to calculate the similarity between two specific product titles, we present a brief overview on BM25 and its successors here. The supervisor of this thesis Professor Huang was instrumental in the research reported in [25], and has been working and contributing consistently on the subject for two decades to follow, in theory and in application. More specifically, as recorded in [36], BM25 with 2 new query expansion methods and Okapi system won Huang and his team the first place in the Genomics/biomedical track among all 135 entrants from around the world in the TREC (Text REtrieval Conference) conference organized by National Institute of Standards & Technology (NIST). In [34], single text classifier or co-training text classifier was applied to Pseudo-Relevance Feedback (PRF) to improve the performance of IR. Term proximity for enhancement of BM25 was proposed in [27], with solidly verified improvement on effectiveness. [106] proposed pseudo term (or Cross Term) to model term proximity for boosting retrieval performance and thus the bigram CRoss TERm Retrieval (CRTER) model based on BM25. Meanwhile, [56] proposed an integrated sampling technique incorporating both oversampling and undersampling, with an ensemble of SVMs to improve the prediction performance. In [18], an ensemble method of SVM and Clustering based on Self-Organized Ant Colony Network (CSOACN), i.e. Combining Support Vectors with Ant Colony (CSVAC), was proposed for network intrusion detection.

To get a document D_j's BM25 score given a query Q, a weighting function for each query term $q_i \in Q$ and the document D_j is first calculated as follows:

$$w(q_i, D_j) = \frac{(k_1 + 1) \times TF(q_i, D_j)}{K + TF(q_i, D_j)} \times \frac{(k_3 + 1) \times QTF(q_i)}{k_3 + QTF(q_i)} \times IDF(q_i), \qquad (2.14)$$

where $IDF(q_i) = \log(1 + \frac{N - DF(q_i) + 0.5}{DF(q_i) + 0.5})$, N is the number of documents in the document collection C:

$$N = |C| = |\{D_1, \ldots, D_{|C|}\}|,$$

$DF(q_i)$ is the DF of q_i. $K = k_1 \times [(1 - b) + b \times dl_{D_j}/avdl]$. dl_{D_j} is the word-level document length of D_j:

$$dl_{D_j} = \sum_{i=1}^{|V_{D_j}|} TF(w_i, D_j), \tag{2.15}$$

where V_{D_j} is the set of distinct terms occurring in D_j:

$$V_{D_j} = \{w_1, \ldots, w_{|V_{D_j}|}\},$$

$TF(w_i, D_j)$ is the TF of w_i in D_j, $avdl$ is the average document length in C:

$$avdl = \frac{\sum_{i=1}^{N} dl_{D_i}}{N}, \tag{2.16}$$

k_1, k_3 and b are parameters. $TF(q_i, D_j)$ is the TF of q_i in D_j, and $QTF(q_i)$ is the TF of q_i in Q. The document D_j's BM25 similarity score given a query Q is then calculated as the sum of D_j's weight for each Q's term:

$$BM25(Q, D_j) = \sum_{i=1}^{|Q|} w(q_i, D_j), \tag{2.17}$$

where w is the term weight obtained from the above Equation 2.14, $|Q|$ is the number of terms in Q.

2.7.2 LM

A probabilistic LM approach [72] constructs a LM M_{D_j} for a document D_j and tries to estimate the probability of a query Q being generated from the LM of the document $P(Q|M_{D_j})$. Then it ranks documents according to their corresponding probabilities.

Smoothing is a technique for LM that tunes maximum likelihood estimator (MLE) to tackle the problem of data sparseness. A MLE for calculating the probability of term w_i being generated from document LM M_{D_j} is:

$$P_{MLE}(w_i|M_{D_j}) = \frac{TF(w_i, D_j)}{dl_{D_j}}, \tag{2.18}$$

where $TF(w_i, D_j)$ is the TF of w_i in D_j, dl_{D_j} is the word-level document length of D_j.

There are two smoothing methods we used in our experiments, i.e. Bayesian smoothing using Dirichlet priors [104] and the Jelinek-Mercer method [104]. In this thesis, we refer to the LM using the above 2 smoothing methods as **Dirichlet LM** and **Jelinek-Mercer LM** respectively. To get a document D_j's LM score given a query Q, a probability $P(q_i|M_{D_j})$ for each query term $q_i \in Q$ and the document D_j is first calculated as follows:

For **Dirichlet LM**:

$$P(q_i|M_{D_j}) = \frac{TF(q_i, D_j) + \mu P_{MLE}(q_i|M_C)}{dl_{D_j} + \mu}, \tag{2.19}$$

where $TF(q_i, D_j)$ is the TF of q_i in D_j, μ is a smoothing parameter, dl_{D_j} is the word-level document length of D_j, $P(q_i|M_C)$ is the MLE of q_i being generated from the collection LM M_C:

$$P_{MLE}(q_i|M_C) = \frac{\sum_{k=1}^{N} TF(q_i, D_k)}{\sum_{k=1}^{N} dl_{D_k}}, \tag{2.20}$$

where N is the number of documents in C.

For **Jelinek-Mercer LM**:

$$P(q_i|M_{D_j}) = (1-\lambda) P_{MLE}(q_i|M_{D_j}) + \lambda P_{MLE}(q_i|M_C), \tag{2.21}$$

where $P_{MLE}(q_i|M_{D_j})$ is the MLE of q_i being generated by M_{D_j}:

$$P_{MLE}(q_i|M_{D_j}) = \frac{TF(q_i, D_j)}{dl_{D_j}}, \tag{2.22}$$

$TF(q_i, D_j)$ is the TF of q_i in D_j, λ is a smoothing parameter, dl_{D_j} is the word-level document length of D_j, $P_{MLE}(q_i|M_C)$ is the MLE of q_i being generated from the collection LM, as in Equation 2.20.

The probability of document LM M_{D_j} generating a query Q is then calculated as the sum of probability of document LM M_{D_j} generating each Q's term:

$$P(Q|M_{D_j}) = \sum_{i=1}^{|Q|} P(q_i|M_{D_j}), \tag{2.23}$$

where $P(q_i|M_{D_j})$ is obtained from the above Equation 2.19 or Equation 2.21, $|Q|$ is the number of terms in Q (word-level length of Q).

2.7.3 VSM with TF-IDF Weight

VSM for IR is a general approach that uses cosine similarity of query vector $\mathbf{Q}$ and document vector $\mathbf{D_j}$ for ranking:

$$CosineSimilarity(Q, D_j) = \frac{\mathbf{Q}.\mathbf{D_j}}{|\mathbf{Q}||\mathbf{D_j}|}, \tag{2.24}$$

where $\mathbf{Q}.\mathbf{D_j}$ is the dot product of query vector and document vector, $|\mathbf{Q}|$ and $|\mathbf{D_j}|$ are their Euclidean norms.

Lucene's VSM with TF-IDF weight was used in our experiments. BM25 also belongs to VSM. Lucene's implementation of VSM with TF-IDF weight refined the traditional VSM, taking document length into account. Matches with longer documents are less precise, so they should contribute less to VSM score.

So, to get a document D_j's VSM-TFIDF score given a query Q, a weighting function for each query term $q_i \in Q$ and the document D_j is first calculated as follows:

$$\begin{aligned} w(q_i, D_j) =& \sqrt{TF(q_i, D_j)} \times IDF(q_i)^2 \times \frac{1}{\sqrt{dl_{D_j}}} \\ =& \sqrt{TF(q_i, D_j)} \times (1 + \log \frac{N+1}{DF(q_i)+1})^2 \times \frac{1}{\sqrt{dl_{D_j}}} \end{aligned}, \tag{2.25}$$

where $TF(q_i, D_j)$ is the TF of q_i in D_j, dl_{D_j} is the word-level document length of D_j, $DF(q_i)$ is the DF of q_i in document collection C, N is the number of documents in C.

The document D_j's VSM-TFIDF similarity score given a query Q is then calculated as the sum of D_j's weight for each Q's term:

$$VSM\text{-}TFIDF(Q, D_j) = \sum_{i=1}^{|Q|} w(q_i, D_j), \tag{2.26}$$

where w is the term weight obtained from the above Equation 2.25, $|Q|$ is the number of terms in Q.

2.7.4 IB Model

IB (Information-based) models [10] are based on information theory. They are based on the assumption that a term having high probability of occurring in the document collection gives little information for a document. The general ranking function of IB models is as follows:

$$RSV(Q, D_j) = \sum_{q_i \in Q} -QTF(q_i) \log P(X \geq TFnorm(q_i, D_j) | \lambda_{q_i}), \tag{2.27}$$

where $QTF(q_i)$ is the TF of q_i in Q, $TFnorm(q_i, D_j)$ is the normalized TF of q_i in D_j and λ_{q_i} is the parameter of q_i's probability distribution in document collection C.

[10] proposed 2 power law distributions for calculating
$P(X \geq TFnorm(q_i, D_j) | \lambda_{q_i})$ in Equation 2.27:

1. Log-Logistic (LL) Distribution:

   The LL distribution's definition:

$$P_{LL}(X < x | r, \beta) = \frac{x^\beta}{x^\beta + r^\beta} (X \geq 0) \tag{2.28}$$

The authors set $\beta = 1$, the Equation 2.27 becomes:

$$\begin{aligned} RSV(Q, D_j) &= \sum_{q_i \in Q} -QTF(q_i) \log P_{LL}(X \geq TFnorm(q_i, D_j) | \lambda_{q_i}) \\ &= \sum_{q_i \in Q} -QTF(q_i) \log \frac{\lambda_{q_i}}{TFnorm(q_i, D_j) + \lambda_{q_i}} \end{aligned} \tag{2.29}$$

2. Smoothed Power-Law (SPL) Distribution:

   The authors defined the following SPL distribution:

$$\begin{aligned} f(x; \lambda) =& \frac{-\log \lambda}{1 - \lambda} \frac{\lambda^{\frac{x}{x+1}}}{(x+1)^2} (0 < \lambda < 1) \\ P(X > x | \lambda) =& \int_x^\infty f(x; \lambda) dx = \frac{\lambda^{\frac{x}{x+1}} - \lambda}{1 - \lambda} \end{aligned}, \tag{2.30}$$

   where f refers to the probability density function. With the SPL distribution, the Equation 2.27 becomes:

$$\begin{aligned} RSV(Q, D_j) &= \sum_{q_i \in Q} -QTF(q_i) \log P_{SPL}(X \geq TFnorm(q_i, D_j) | \lambda_{q_i}) \\ &= \sum_{q_i \in Q} -QTF(q_i) \log \frac{\lambda_{q_i}^{\frac{TFnorm(q_i, D_j)}{TFnorm(q_i, D_j)+1}} - \lambda_{q_i}}{1 - \lambda_{q_i}} \end{aligned} \tag{2.31}$$

The probability distribution parameter λ_{q_i} in Equation 2.27 can be set as follows:

1. ATF (average term frequency): the average number of occurrences (or TF) of q_i in the collection C:

$$\lambda_{q_i} = \frac{TTF(q_i, C)}{N} = \frac{\sum_{k=1}^{N} TF(q_i, D_k)}{N}, \tag{2.32}$$

   where N is the number of documents in C, $TTF(q_i, C)$ is the total term frequency (TTF) of q_i in C, $TF(q_i, D_k)$ is the TF of q_i in D_k.

2. ADF (average document frequency): the average number of documents containing q_i in the collection C:

$$\lambda_{q_i} = \frac{DF(q_i)}{N}, \tag{2.33}$$

where N is the number of documents in C, $DF(q_i)$ is the DF of q_i in C.

The normalized TF $TFnorm(q_i, D_j)$ in Equation 2.27 can be calculated as follows:

1. Normalization H1 (NormH1) [2]: This model assumes uniform distribution of TF:

$$TFnorm(q_i, D_j) = TF(q_i, D_j) \times c \times \frac{avdl}{dl_{D_j}}, \tag{2.34}$$

where $TF(q_i, D_j)$ is the TF of q_i in D_j, $avdl$ is the average document length in C, dl_{D_j} is the word-level document length of D_j, c is a parameter (default to 1 in Lucene).

2. Normalization H2 (NormH2) [2]: This model assumes that TF is inversely related to document length:

$$TFnorm(q_i, D_j) = TF(q_i, D_j) \times \log\left(1 + c \times \frac{avdl}{dl_{D_j}}\right), \tag{2.35}$$

where $TF(q_i, D_j)$ is the TF of q_i in D_j, $avdl$ is the average document length in C, dl_{D_j} is the word-level document length of D_j, c is a parameter (default to 1 in Lucene).

3. Normalization H3 (NormH3) [2]: This is TF normalization with Dirichlet Priors, similar to Equation 2.19:

$$TFnorm(q_i, D_j) = \frac{TF(q_i, D_j) + \mu \frac{\sum_{k=1}^{N} TF(q_i, D_k)}{\sum_{k=1}^{N} dl_{D_k}}}{dl_{D_j} + \mu}, \tag{2.36}$$

where $TF(q_i, D_j)$ is the TF of q_i in D_j, μ is a smoothing parameter (default to 800 in Lucene), dl_{D_j} is the word-level document length of D_j, N is the number of documents in C.

4. Normalization Z (NormZ) [3]: This is TF normalization based on Pareto-Zipf distributions:

$$TFnorm(q_i, D_j) = TF(q_i, D_j) \times (\frac{avdl}{dl_{D_j}})^z, \tag{2.37}$$

where $TF(q_i, D_j)$ is the TF of q_i in D_j, $avdl$ is the average document length in C, z is a parameter (default to 0.3 in Lucene), dl_{D_j} is the word-level document length of D_j.

5. No Normalization: This is to use the original TF:

$$TFnorm(q_i, D_j) = TF(q_i, D_j) \tag{2.38}$$

2.7.5 Cosine Similarity of Doc2vec Embeddings

Similar to word embedding [65], in [46], sentence/paragraph/document embeddings could be trained in two ways:

The first one is Paragraph Vector Distributed Memory (PV-DM), where paragraph id and the context words (words within a sliding window of user-defined constant size c before a word), i.e. $[w_{t-c}, ..., w_{t-1}]$ are trained to predict that word w_t.

The second way is Paragraph Vector Distributed Bag of Words (PV-DBOW), where a paragraph id is trained to predict randomly sampled words within a randomly sampled context of size c in the paragraph.

As mentioned in the definition of VSM, the cosine similarity of a query's document embedding $\mathbf{Q}$ and a document's document embedding $\mathbf{D_j}$ can be calculated with Equation 2.24.

2.8 KNN Text Classification Using IR Model

We chose to use IR model for KNN searching, because it can quickly find nearest neighbours of a query with inverted index. The initial attempt of using IR model for nearest neighbour searching was presented in [59], where, similar to VSM with TF-IDF Weight, traditional BoW representation was used and each searchable term w_i (i.e. a single word or a capitalized word pair) was given a weight

$$\frac{1}{TTF(w_i, C)}, \tag{2.39}$$

where $TTF(w_i, C)$ is the TTF of w_i in the training corpus C (i.e. collection frequency). Our work used the same weighting scheme of KNN as that in [59], i.e. a document is weighted by similarity score between it and the given query (as shown in Equation 2.11). However, our work differs from [59] in the following aspects: we used different IR models; their classification task (i.e. multi-label news story classification) is different from ours; while they removed stop words and 72 common words from the document collection, we used whole document as query (we will analyse the effect of removing stop words in Section 5.2).

IR model based KNN text classification is also considered in [87]. [87] focused on short Web snippet classification (with 8 target classes) and used a few important words in a document as query. In contrast, we used the whole document as query. We think preserving the whole product title as query can yield better prediction results, as product titles are usually short and the number of classes is large (e.g. 3008 target classes in the SIGIR eCom Data Challenge Dataset), which was confirmed in our experiments (as in Section 5.2). Also, [87] only used default IR model of Lucene (i.e. VSM with TF-IDF weight) for searching, while we compared classification performance of a range of IR models. In addition, their classification task is different from ours. Furthermore, in their task, they found KNN with simple majority voting performs better than weighted KNN,

which is different from our findings. This suggests that the best KNN weighting scheme is probably dependent on the task-specific dataset.

2.9 Neural IR Models for Modelling E-commerce Product Similarity

In recent years, due to the need of effective product searching in E-commerce websites, neural network based approaches, such as K-NRM [95], have also been used in modelling product similarity. [6] proposed task modelling techniques to construct a large-scale e-commerce dataset for product similarity modelling. They evaluated several supervised neural IR models' (i.e. CLSM [83], DSSM [32], Duet [66], K-NRM [95]) performance on the dataset. They found K-NRM could outperform the baseline (i.e. VSM with TF-IDF weight) significantly, reducing 33% error rate of the baseline, while distributed models could not be as good as the baseline.

Generally, using supervised neural IR model requires training dataset that is manually labelled or derived from clickstream data (supervised learning), whereas our approach do not (unsupervised learning). Most of IR models in this study belong to unsupervised hand-crafted IR model, except for cosine similarity of Doc2vec document embedding [46], which is an unsupervised neural IR model.

2.10 Fine-tuning Pre-trained Model for Text Classification

In the field of AI, in particular computer vision (CV) and natural language processing (NLP), *transfer learning* has been a popular research topic. By definition, *transfer learning* means to apply what we have learned in one situation to another learning situation. Hence, usually in the new learning situation, we could learn more easily and quickly. Similar to human, in this way, we can train a model with less examples, which saves

human labour cost in manual labelling.

In recent years, a new research trend of transfer learning of NLP is to build a universal model capable of adapting to various downstream NLP tasks like part-of-speech (POS) tagging at a little cost of architecture modification. Language modelling has been widely adopted as a pre-training task, because it only requires readily available text corpus for unsupervised pre-training.

In [12], 2 unsupervised pre-training tasks for RNN (e.g. LSTM), namely language modelling and sequence autoencoding, were proposed. [12] also found that pre-training LSTM leads to better stability of training.

In [30], Universal Language Model Fine-tuning (ULMFiT) was proposed. Their proposed method is as follows: they first pre-trained a NLM AWD-LSTM [62] on a general-domain corpus, namely Wikitext-103 (103 million words) [63]. Then, they fine-tuned the NLM on a specific NLP task. After that, 2 linear blocks were added to turn the model into a task-specific classifier. Finally, the classifier was fine-tuned on the specific task. Apart from this method, [30] also proposed several fine-tuning techniques, namely discriminative fine-tuning and gradual unfreezing. Their method could obtain high performance on 6 text classification benchmarks. Among these benchmarks, the largest number of target classes is 14 and the least number of examples is 5,500. [30] also found that fine-tuning is better than training from scratch, especially on a small dataset.

Different from [30], [73] proposed to pre-train LtR NLM based on Transformer [91] architecture. In [73], the OpenAI GPT model was pre-trained on BooksCorpus (800 million words) dataset and then fine-tuned for a specific supervised task. The model has been evaluated in 12 NLP tasks including 2 text classification problems, namely the Stanford Sentiment Treebank (SST-2) (with 2 target classes) and the Corpus of Linguistic Acceptability (CoLA) (with 2 target classes).

In [15], the BERT model, also based on Transformer [91], was pre-trained on 2 large text corpuses (i.e. BooksCorpus and Wikipedia (2,500 million words)) for 2 novel pre-training tasks, namely MLM and NSP task.

Later, [74] proposed to pre-train a much larger Transformer-based NLM, namely GPT-2 (1.5 billion parameters), than BERT-large model in [15] (340 million parameters) on a huge web text corpus, namely WebText (40Gb in terms of file size). This model could reach SotA performance in zero-shot setting (i.e. without fine-tuning on task-specific dataset) on 7 out of 8 language modelling benchmarks. The model has not been applied to text classification yet. [74] also mentioned that because the model is so powerful that they would not make it publicly available to prevent malicious use of it.

Later in 2019, [101] proposed a new evaluation metric, i.e. codelength, to calculate how fast a model learns in a new task by using knowledge gained from previous training. [101] also found that although recently developed general NLP models like BERT [15] and ELMo [71] could obtain new SotA performance, fine-tuning them still requires large amounts of supervised/unsupervised training examples and they could easily forget acquired knowledge from previous training. In this thesis, we also got similar conclusions from our experiments with BERT-large uncased model [15].

Later in June 2019, XLNet [99], extending Transformer-XL [13], was proposed. It uses a permutation language modeling objective to enable better capturing of bidirectional contextual information. XLNet, pretrained on 126Gb text corpuses, achieved new SotA on 7 text classification benchmarks.

Later in July 2019, Robustly optimized BERT approach (RoBERTa)[55] was proposed based on BERT [15]. It was pretrained longer on longer sequences, with larger batch size and larger text corpuses. They also removed NSP task. Additionally, dynamic masking pattern was applied to the training data. RoBERTa could achieve new SotA single model performance on SST and CoLA.

2.11 Fine-tuning Pre-trained BERT Model for Large-scale Product Taxonomy Classification

Motivated by the recent success of BERT [15] in various NLP tasks including text classification and text pair classification, we conducted experiments based on it to see whether it can effectively use pre-trained knowledge to tackle this product taxonomy classification task. Although it has been adopted to tackle several text classification problems like SST-2 and CoLA, to the best of our knowledge, it has not been applied to large-scale product taxonomy classification task like this task (with 3008 target classes) yet.

2.12 Overview of 2018 SIGIR eCom Data Challenge

We [31] participated in 2018 SIGIR eCom Data Challenge[1], and the task is to classify a product into the taxonomy tree only based on its title. The final results of all participants are shown in Table 2.1. The participants were ranked according to weighted-F1. Here, we give an overview of top participants' approaches to solving the product classification problem:

2.12.1 Winner's Solution

The winner of the Data Challenge [84] used an ensemble of 3 pairs of Bidirectional Long Short-Term Memory with Balanced Pooling Views (B-LSTM-BPV) networks (i.e. 6 LSTM-BPV networks in total) for end-to-end classification. Each LSTM-BPV network takes character embeddings as input and contains 2 LSTM layers, with random dropout after the embedding layer, between LSTM layers and after the LSTM output. The second LSTM layer's output is a concatenation of the final hidden state $\mathbf{h}_T$, mean-pooling, max-

[1] https://sigir-ecom.github.io/data-task.html

Ranking	Team	Weighted-Precision	Weighted-Recall	Weighted-F1 Score
1	mcskinner	0.8697	0.8418	0.8513
2	MKANEMAS	0.8425	0.8427	0.8399
3	tiger	0.8397	0.8428	0.8379
4	Uplab	0.8368	0.8419	0.8366
5	JCWRY	0.8528	0.8172	0.8295
6	neko	0.8267	0.8305	0.8256
7	Ravenclaw	0.8289	0.8114	0.8175
8	Uplab-2	0.8186	0.8243	0.8173
9	ssdragon	0.8226	0.8163	0.8172
10	RITB-Baseline	0.8276	0.8077	0.8142
11	inception	0.8259	0.8077	0.8139
12	Tyche	0.8599	0.7644	0.8004
13	minimono	0.8019	0.8023	0.7994
14	Topsig	0.7921	0.8014	0.7941
15	VanGuard	0.7899	0.7917	0.7884
16	**HSJX-ITEC-YU**	0.7809	0.7821	0.7790
17	Waterloo	0.7802	0.7857	0.7781
18	CorUmBc	0.7745	0.7712	0.7690
19	Sam-chan	0.7718	0.7745	0.7666
20	Tyken2018	0.7654	0.7603	0.7509
21	Or	0.7419	0.7250	0.7245
22	Coumodo	0.7275	0.7140	0.7107
23	Uplab-3	0.6698	0.6588	0.6509
24	the1owl	0.5947	0.6277	0.5682
25	sherlock	0.5855	0.5091	0.5025
26	B4_toku	0.4340	0.4751	0.4144
27	Hawk	0.2679	0.0561	0.0642
28	Fractal AIML	0.0148	0.0152	0.0150

Table 2.1: **Test Dataset**, performance comparison of different teams in the Stage 2 of SIGIR eCom Data Challenge. Our team is printed in bold.

pooling and min-pooling representations of all hidden states $\mathbf{H} = \{\mathbf{h}_1, \ldots, \mathbf{h}_T\}$, which is called balanced pooling views (BPV)(traditional LSTM [12] layer's output is only the final state $\mathbf{h}_T$):

$$\mathbf{h}_c = [\mathbf{h}_T, meanpool(\mathbf{H}), maxpool(\mathbf{H}), minpool(\mathbf{H})], \tag{2.40}$$

where [] means concatenation. After random dropout, this representation is fed to a linear layer for classification. The network training method used is Stochastic Gradient Descent with Momentum (SGDM) [88], with a 1cycle learning schedule [85] that allows fast convergence. [84] also optimized the weighted-F1 by setting a probability threshold for each target class. Such optimizing technique can boost weighted-precision (weighted-P) substantially at the expense of weighted-recall (weighted-R). Let t be a test product title. Specifically, the final prediction is made with two approaches listed as follows:

The first one is **Best-wins (no F1 optimizing)**: The category id path dc_q with highest raw probability is deemed the predicted category id path:

$$P_{ensemble}(t \in dc_q) = \max_{j \in \{1,\ldots,3008\}} P_{ensemble}(t \in dc_j) \tag{2.41}$$

The second one is **F1 tuning/optimizing** [84]: In this approach, softmax probability instead of raw probability is used:

$$P_{ensemble,smax}(t \in dc_i) = \frac{e^{P_{ensemble}(t \in dc_i) - \max_{q \in \{1,\ldots,3008\}} P_{ensemble}(t \in dc_q)}}{\sum_{j=1}^{3008} e^{P_{ensemble}(t \in dc_j) - \max_{q \in \{1,\ldots,3008\}} P_{ensemble}(t \in dc_q)}} \tag{2.42}$$

To optimize F1, [84] set a probability threshold τ_{dc_i} for each category id path dc_i, below which the probability is not counted:

$$\text{if } P_{ensemble,smax}(t \in dc_i) < \tau_{dc_i} \text{ , } P_{ensemble,smax}(t \in dc_i) := 0 \tag{2.43}$$

The threshold τ_{dc_i} is estimated as follows:

1. The number of true instances of dc_i is estimated as the total of probability of a test instance t_j belonging to dc_i in test dataset TE:

$$n_{true}(dc_i) = \sum_{j=1}^{|TE|} P_{ensemble,smax}(t_j \in dc_i), \tag{2.44}$$

where $|TE|$ is the number of test instances in TE.

2. Then $\{P_{ensemble,smax}(t_j \in dc_i)|j \in \{1,\ldots,|TE|\}\}$ is sorted in descending order, which is denoted as

$$\{P_{ensemble,smax,sorted}(t_q \in dc_i)|q \in \{1,\ldots,|TE|\}\}$$

3. After that, [84] calculates the cumulative sum set $\{\sum_{k=1}^{q} P_{ensemble,smax,sorted}(t_k \in dc_i)|q \in \{1,\ldots,|TE|\}\}$ from

$$\{P_{ensemble,smax,sorted}(t_q \in dc_i)|q \in \{1,\ldots,|TE|\}\}$$

4. After that, P, R and F1 for the top k instances with highest probabilities $\{P_{ensemble,smax,sorted}(t_q \in dc_i)|q \in \{1,\ldots,k\}\}$ are estimated as follows:

$$Precision(dc_i@k) = \frac{\sum_{j=1}^{k} P_{ensemble,smax,sorted}(t_j \in dc_i)}{k}, \tag{2.45}$$

$$Recall(dc_i@k) = \frac{\sum_{j=1}^{k} P_{ensemble,smax,sorted}(t_j \in dc_i)}{n_{true}(dc_i)}, \tag{2.46}$$

$$F1(dc_i@k) = \frac{2 \times Precision(dc_i@k) \times Recall(dc_i@k)}{Precision(dc_i@k) + Recall(dc_i@k)} \tag{2.47}$$

5. [84] then finds the maximum estimated F1 and uses the corresponding probability as the threshold:

$$\tau_{dc_i} = P_{ensemble,smax,sorted}(t_j \in dc_i), \tag{2.48}$$

where $j = argmax_{k \in \{1,\ldots,|TE|\}} F1(dc_i@k)$.

After thresholding, the category id path dc_q with highest softmax probability is seen as the predicted category id path:

$$P_{ensemble,smax}(t \in dc_q) = \max_{j \in \{1,...,3008\}} P_{ensemble,smax}(t \in dc_j) \tag{2.49}$$

2.12.2 Other Top Participants' Solutions

The second winner of the competition [89] used an ensemble network consisting of a multi-kernel TextCNN based on word embeddings, a Bidirectional LSTM (BLSTM) with soft attention based on word embeddings and a multilayer perceptron (MLP) based on ad-hoc features generated from words. An external dataset, i.e. Amazon Product Data, was also used to generate ad-hoc features. The output of the 3 networks is flattened, concatenated and passed into a linear layer.

The third winner of the competition [102] also used ensemble strategy. They first pre-trained fastText word embeddings [5] on the merged set of the Training and the Test Dataset. End-to-end models, namely fastText and Attention-based LSTM (AbLSTM), were trained. Also, 2 types of hierarchical tree classification models were adopted: 1) The first type is an ensemble of 8 fastText or AbLSTM models, each predicts a category id and searches the tree in top-down order. 2) The second one is also an ensemble of 8 fastText/AbLSTM models, but it differs from the first one, where each model predicts a combination of category ids from top to the current level. Their final submission is based on the weighted voting of 6 models.

The fourth winner of the contest [19] tried using 3 approaches, namely an end-to-end classifier (i.e. SVMs with TF-IDF unigram and bigram features), a hierarchical classifier and a CNN with Glove [70] pre-trained word embedding, to solve this classification task. They obtain best classification results using SVMs with TF-IDF unigram and bigram features.

In [37], the fifth winner of the contest, an ensemble of 5 models based on CNN was

used. They utilized multiple features as input to the ensemble, i.e. named entity types, POS tags, document embeddings [46], pre-trained or supervised word embeddings [65] and supervised character one-hot embeddings. To overcome unbalanced data distribution, they also used oversampling and threshold moving (i.e. raw prediction probabilities are divided by class size (number of instances belonging to a given class in training dataset)) techniques.

The sixth winner of the contest [48] proposed to tackle the product taxonomy classification problem through sequence generation, which makes it possible to generate category id paths that do not occur in training dataset. Specifically, they used ensemble of attentional sequence-to-sequence (seq2seq) models based on RNN.

2.12.3 Conclusion from Top Participants' Solutions

From the methods of top participants, we can see that using ensemble strategy can effectively boost the product classification performance. Also, we can see that neural network based approaches are popular and can yield good results.

Chapter 3

Methodology

In this chapter, we will explain product categorization through weighted KNN algorithm with IR model as similarity function (pseudo code in Table 3.3), with an example provided. The framework in support of the classifier will also be presented (illustration in Figure 3.1 and pseudo code in Table 3.4). We will also discuss the advantages of our KNN approach. After that, we will present our hybrid method where we combine prediction results of our KNN algorithm with LSTM-BPV(s) [84]. Finally, we will examine the advantages of the hybrid system.

3.1 A Weighted KNN Classifier for Product Categorization

In our program, k of our KNN classifier is a parameter that controls the number of nearest neighbours contributing to prediction. The input of our KNN algorithm is a query (test title), a specific IR model, k and the training document collection. And the output of our KNN algorithm is the category id path with highest category score (i.e. the most relevant category id path) among category id paths of training titles most similar to the query. We chose to use a flat (end-to-end) classification approach to solving the given problem instead of a hierarchical one for its simplicity and effectiveness.

An IR model relevance score (retrieval status value (RSV)) is calculated for each

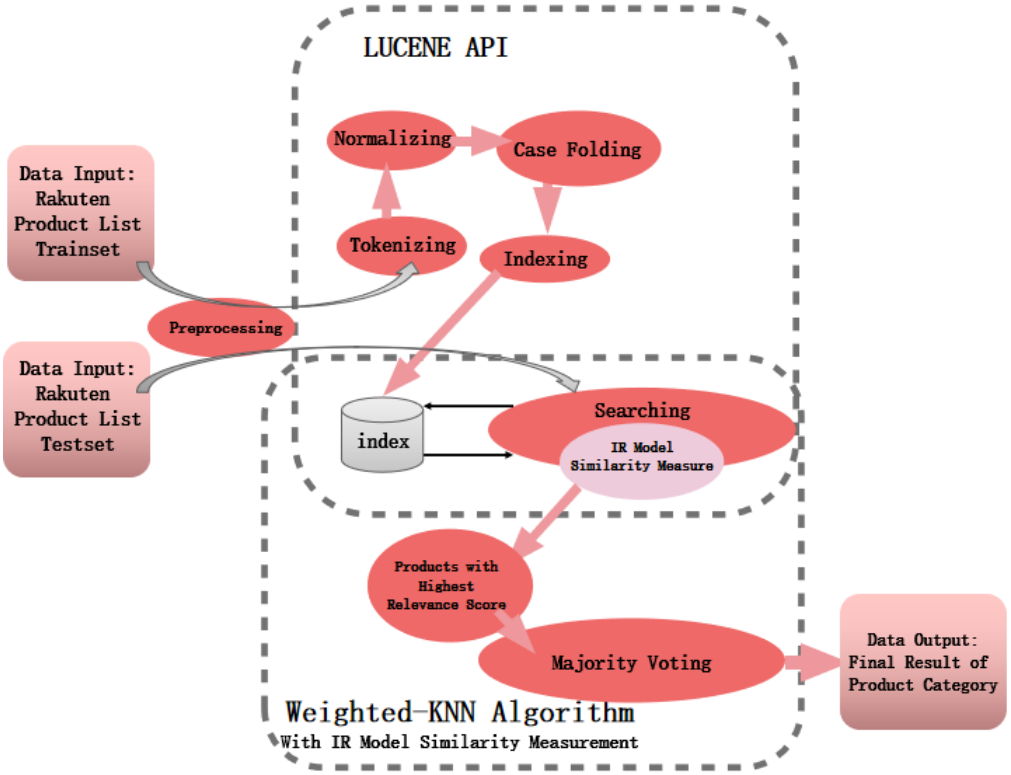


Figure 3.1: A Weighted KNN Product Classification System

title pt_i in the training dataset and a title t_j in the test dataset. In KNN paradigm, the similarity score of our approach is the IR model relevance score between an item title in the test dataset and an item title in the training dataset. The higher the relevance score, the more similar a training title and a test title. We tried setting different values of k in KNN to see whether or not predicting based on individual match (i.e. 1NN) is better than on multiple matches, as the individual match may be an outlier (tuning of k will be examined in Section 4.5). The pseudo code of our weighted KNN algorithm is shown in Table 3.3. Specifically, the algorithm works as follows:

Suppose p_i is an instance/example (i.e. product) in the training dataset TR. p_i contains product title pt_i and product category id path c_i. t_j is a product title in the test

dataset TE. N is the number of products in TR. When $k = 1$, we assign the category id path c_1 of the top 1 matched training title (document), i.e. the title pt_1 with highest IR model relevance score given that test title (query) t_j, as the predicted category id path $\hat{c}_j$:

$$\hat{c}_j = c_1, \tag{3.1}$$

where $RSV(t_j, pt_1) = \max_{m \in \{1,2,...,N\}} RSV(t_j, pt_m)$.

Generally, when $k > 1$, we assign the category id path with highest category score among returned top n ($n \leq k$, since it is possible that the number of matches is less than k) products' category id paths $\{c_1, c_2, ..., c_n\}$. Specifically, the algorithm finds the distinct category id paths $\{dc_1, dc_2, ..., dc_i\} \subset \{c_1, c_2, ..., c_n\}$ ($i \leq n$) and calculates their category scores $\{Cat(dc_1), Cat(dc_2), ..., Cat(dc_i)\}$, where

$$Cat(dc_m) = \sum_{i=1}^{n} RSV(t_j, pt_i) \times \mathbf{1}_{\{pt_i \in dc_m\}}, \tag{3.2}$$

where $\mathbf{1}_{\{pt_i \in dc_m\}}$ is an indicator function, which equals to 1 if $pt_i \in dc_m$ and to 0 otherwise. The distinct category id path dc_m with the highest category score among the category id paths of the top n matched training titles given a test title is deemed the predicted category id path $\hat{c}_j$:

$$\hat{c}_j = dc_m, \tag{3.3}$$

where $Cat(dc_m) = \max_{q \in \{1,2,...,i\}} Cat(dc_q)$.

In some rare occasions, when the distinct category id paths have the same category scores within the top matched training titles, we assign the category id path of the higher ranked matched training title(s) as predicted category id path. For example, if

$$Cat(dc_1) = Cat(dc_2) = \max_{q \in \{1,2,...,i\}} Cat(dc_q), \tag{3.4}$$

then dc_1 is deemed the predicted category id path $\hat{c}_j$.

If no match is found ($n = 0$), we assign "2296>3597>689" as predicted category id path, which corresponds to "Media>Music >Pop" (we manually deduced this from

training data and Rakuten website[2]). This is because no match means the title is short and distinct, which generally belongs to "Media>Music" big category. Additionally, according to [48], this big category contains 25 distinct categories, among which "Pop" has the largest number of examples. Generally, without prior knowledge, it would be difficult even for human to predict the genre of a music product based on its short title.

Figure 3.2: An earring item need to be categorized. Picture source: https://www.rakuten.com/shop/sabrina-silver/product/TE3873/

Example: Here is an example to show how a product is categorized in our KNN system. We choose to use BM25 model as similarity function of KNN here for illustration purpose. We set $k_1 = 1.2$, $b = 0.92$ in BM25. As shown in Fig.3.2, given this item (query) in test dataset:

```
"Sterling Silver Dangle Ball Earrings w/ Brilliant Cut CZ
 Stones & Yellow Topaz-colored Crystal Balls, 1"" (26 mm)
tall"
```

[2] https://www.rakuten.com

If we set $k = 10$, then the searcher will return the item's top 10 matches in training set according to BM25 similarity score of a document and the query in descending order as shown in Table 3.1 below.

As an illustration, the term weight for the matched term "sterling" (q_1) in the top 1 document (D_1) is calculated as follows:

$$\begin{aligned} w(q_1, D_1) &= \frac{(k_1+1) \times TF(q_1, D_1)}{k_1 \times [(1-b) + b \times dl/avdl] + TF(q_1, D_1)} \times \frac{(k_3+1) \times QTF(q_1)}{k_3 + QTF(q_1)} \times \\ &\log\left(1 + \frac{N - DF(q_1) + 0.5}{DF(q_1) + 0.5}\right) \\ &= \frac{(1.2+1) \times 1}{1.2 \times [(1-0.92) + 0.92 \times 18/11.566] + 1} \times \frac{(8+1) \times 1}{8+1} \times \\ &\log\left(1 + \frac{800000 - 16528 + 0.5}{16528 + 0.5}\right) \\ &= 3.0329628 \end{aligned} \tag{3.5}$$

The weighted KNN algorithm then calculates the category scores of the categories within these matches. As shown in Table 3.2 below, "1608>2320>2173>2878" (corresponding to "Clothing, Shoes & Accessories>Jewelry & Watches>Earrings>Stud Earrings") has the highest category score among the top 1/3/5/7/10 matches' categories. Thus, the category "1608>2320>2173>2878" is assigned as the predicted category id path when our KNN algorithm's k is set to 1/3/5/7/10. We have manually verified on the Rakuten website[3] that this prediction is correct.

3.2 The Classifier in Action

Based on the classifier above, we actually implemented a system for the classifier in action. Figure 3.1 is a pictorial description of the KNN system, where ovals are functional

[3] https://www.rakuten.com

Ranking	Product Title	Category Id Path	BM25 score
1	“Sterling Silver Marquise Shape Dangle Earrings with Brilliant Cut CZ Stones, 1 1/16 in. (26 mm) tall”	1608>2320>2173>2878	56.89168
2	“Sterling Silver Floral Dangle Chandelier Earrings with Brilliant Cut CZ Stones, 1 1/4 in. (31 mm) tall”	1608>2320>2173>2878	51.909283
3	“Sterling Silver Curvy Hoop Earrings with Brilliant Cut CZ Stones, 13/16 in. (21 mm)”	1608>2320>2173>2878	42.515083
4	“Sterling Silver French Clip Black Onyx Bar Earrings with Brilliant Cut CZ Stones, 5/8 in. (16 mm) tall”	1608>2320>2173>2878	41.787933
5	“Sterling Silver Square-shaped Stud Earrings (7 mm) & Pendant (12mm tall) Set, with Princess Cut Blue Sapphire-colored CZ Stones”	1608>2320>2173>3881	40.36754
6	“Sterling Silver Double Wire Knot Lace Post Earrings with Brilliant Cut CZ Stones, 1 7/16 in. (36 mm)”	1608>2320>2173>2878	39.01934
7	“High Polished Sterling Silver 3/4”” (18 mm) tall Heart Cut Out Pendant, with Brilliant Cut CZ Stones, with 18”” Thin Box Chain”	1608>2320>498>1546	37.43821
8	“High Polished Sterling Silver 7/16”” (11 mm) tall Bead Charm, with Brilliant Cut CZ Stones, with 18”” Thin Box Chain”	1608>2320>2495>3682	37.43036
9	“Sterling Silver Black Onyx Ring with Brilliant Cut CZ Stones, 1/4 in. (6 mm) wide, size 7”	1608>2320>3648	36.71535
10	“High Polished Sterling Silver 11/16”” (17 mm) tall Flower Cut Out Pendant, 1.5mm Brilliant Cut CZ Stones, with 18”” Thin Box Chain”	1608>2320>498>1546	36.443573

Table 3.1: Top 10 matches obtained given the query

Candidate Category Id Path	Candidate Category	k Value				
		1	3	5	7	10
1608>-2320 > 2173> 2878	Clothing, Shoes & Accessories > Jewelry & Watches > Earrings > Stud Earrings	56.89	151.32	193.10	232.12	232.12
1608>-2320 > 498> 1546	Clothing, Shoes & Accessories > Jewelry & Watches > Pendants & Necklaces > Pendants	0	0	0	37.44	73.88
1608>-2320 > 2173> 3881	Clothing, Shoes & Accessories > Jewelry & Watches > Earrings > Earring Sets	0	0	40.37	40.37	40.37
1608>-2320 > 3648	Clothing, Shoes & Accessories > Jewelry & Watches > Rings	0	0	0	0	36.72
1608>-2320 > 2495> 3682	Clothing, Shoes & Accessories > Jewelry & Watches > Accessories > Individual Charms	0	0	0	0	37.43

Table 3.2: Distinct Category id paths, corresponding categories and category scores within top k(k=1/3/5/7/10) documents' category id paths

function KNN_IRModel(ir_model, q, DC, k) **returns** predicted category id path
  input: ir_model: an IR model to use for generating relevance score
    q: the query (test title)
    DC: the document collection of product titles and corresponding category id paths
    k: the k value in KNN algorithm
  local variables: p_j: the jth matched training product containing pt_j (product title) and c_j(its corresponding category id path)
    $\hat{c}$: the predicted category id path of q

  search q in DC with ir_model and get top n ($n \leq k$) matches $\{pt_1, pt_2, ..., pt_n\} \subset DC$ and corresponding $\{c_1, c_2, ..., c_n\}$
  if n equals 0 **then**
    set $\hat{c}$ as "2296>3597>689"
  else
    set $\hat{c}$ as the category id path with highest category score among $\{c_1, c_2, ..., c_n\}$
  end if
  return $\hat{c}$

Table 3.3: pseudo code of the KNN classifier in action

procedure Main_program(ir_model, TR, TE, k) **returns** prediction file
  input: ir_model: an IR model to use for generating relevance score
    TR: the training dataset
    TE: the test dataset
    k: the k value for KNN algorithm
  local variables: p_j: the jth training product containing pt_j (product title) and
    c_j (its corresponding category id path)
    t_j: the jth test product title
    $\hat{c}_j$: the predicted category id path of t_j

for each $p_j \in TR$ **do**
  preprocess pt_j
  tokenize pt_j
  normalize pt_j
  lowercase pt_j
  index pt_j
  store (pt_j, c_j) in $DocumentCollection$
end for
for each $t_j \in TE$ **do**
  preprocess t_j and store it in $temp\_t$
  get $\hat{c}_j$ through KNN_IRModel($ir\_model, temp\_t, DocumentCollection, k$)
  write t_j and $\hat{c}_j$ in prediction file
end for

Table 3.4: pseudo code of the main system in action

components, cylinder is index and rounded rectangles are data input/output in interaction with the KNN system. Specifically, input product title can be effectively categorized through searching the index of product titles in training dataset. Training data are preprocessed, analysed to get a word-based index. Given a query (product title), the system calculates the IR model relevance scores (RSV) of the given product title and training titles, and classifies it as the category id path with highest category score among its most relevant product title(s)' category id paths in training set. The implementation is in JAVA. We tried different approaches and strategies for minimizing the classification error and matching the product titles to categories with high accuracy.

More precisely, as shown in Figure 3.1 and Table 3.4, the major flow of the product classification system is as follows:

First, the training dataset TR (product titles with category id paths) in tsv (tab-separated values) format is read line by line as document input. Each document $p_j \in TR$ has two fields, one for product title pt_j and one for category id path c_j. Second, documents' product titles pt_j are preprocessed. Specifically, "w/out" is replaced by "without", "w/" is replaced by "with", "&" is replaced by "and", "'" is replaced by "feet", """" is replaced by "inches" and ":" is replaced by " : ". After that, documents' product titles pt_j and category id paths c_j are indexed in text field and string field respectively. Product titles pt_j in text fields are analysed by Lucene Standard Analyser. Specifically, they are tokenized by Lucene's standard tokenizer, before being normalized by Lucene's standard filter (for example, "bags" is replaced with "bag") and turned into lowercase (case folding) by lowercase filter. In contrast, category id paths c_j in string fields are not analysed, since they are target categories for later classification. After that, the test dataset TE (product titles without category id paths) is read line by line as query input. Test titles $t_j \in TE$ are preprocessed in the same way as training product titles. Then, as shown in Table 3.3, the KNN searcher searches the index with test title t_j with the IR model to get

top n ($n \leq k$) most relevant training titles $\{pt_1, ..., pt_n\}$ and their corresponding category id paths $\{c_1, ..., c_n\}$. This is followed by our KNN voting algorithm (as shown in Table 3.3) that returns the category id path with highest category score among these training title(s)' category id paths $\{c_1, c_2, ..., c_n\}$ as predicted category id path $\hat{c}_j$. Searching and category prediction are done using multiple threads (e.g. 16) in parallel. Finally, test titles and their predicted category id paths are written in a tsv file.

3.3 Advantages of IR Model Based Weighted KNN for Large-scale E-commerce Product Classification

In [82], the coarse-level classifier is weighted KNN algorithm based on eBay search system, which can classify an item in less than 100ms. However, our work is different from [82]: they used a different weighting scheme (i.e. reciprocal of the rank position) and dynamically updated k with dynamic threshold depending on similarity of a given item and its nearest neighbour. We have tried several weighting schemes and the one in use in our system performs best in the validation set (we will compare them in Section 5.6). In addition, our work is straightforward compared to [82] and we used IR models as similarity function of KNN for the product classification task (which are different from the similarity function used in [82], i.e. cosine similarity of top features selected with information gain criterion). Furthermore, our IR model based approach does not require feature selection in [82], as we will show in Section 5.2 and 5.4, feature selection is probably not a good strategy of improving product classification performance.

In our previous work [31], we just used 1 IR model (i.e. BM25) as similarity function in KNN with simple voting. In contrast, in this thesis, we used weighted KNN instead and compared different IR models' classification performance, and we proposed to enhance classification performance by combining prediction results of our weighted KNN

classifier and SotA LSTM-BPV(s) [84].

One benefit of our KNN system is its scalability: when new training examples come in, the system simply updates an existing index. In contrast, as we will show in Section 5.8 and 5.10, neural network based methods often require careful fine-tuning and are usually prone to catastrophic forgetting.

Another benefit of this KNN system is its ease of implementing, as e-commerce websites already have product search systems. According to [6], some large e-commerce companies have even developed proprietary search systems.

Additionally, compared to other complicated methods like BERT, the prediction results generated from our KNN system are easy to interpret, which is important especially in an e-commerce setting.

3.4 A Hybrid Approach Based on Weighted KNN and LSTM-BPV to Product Classification

To enhance classification performance, we use linear interpolation to combine the prediction results of our KNN system with the winner solution in SIGIR eCom Data Challenge, namely LSTM-BPV(s) [84]. Specifically, the method works as follows:

Let dc_j be the predicted category id path given a test product title t predicted by our KNN system. First, we estimate the probability of a test product title t belonging to the category id path dc_j predicted by our KNN algorithm as follows:

$$P_{KNN}(t \in dc_j) = \max_{i \in \{1,\ldots,3008\}} P_{LSTM\text{-}BPVs}(t \in dc_i) + 0.1, \tag{3.6}$$

where $P_{LSTM\text{-}BPVs}(t \in dc_i)$ is the raw probability of t belonging to a category id path dc_i generated from LSTM-BPV network(s). We add a 0.1 to ensure that there would not be 2 equal maximum probabilities. Then, the combined raw probability is calculated as

follows:

$$P_{ensemble}(t \in dc_j) = (1 - \lambda) \times P_{LSTM\text{-}BPVs}(t \in dc_j) + \lambda \times P_{KNN}(t \in dc_j), \quad (3.7)$$

where $P_{LSTM\text{-}BPVs}(t \in dc_j)$ is the raw probability of t belonging to dc_j generated from LSTM-BPV network(s) and λ is a tuning parameter that controls how much we rely on our KNN system's prediction.

In addition, please note that other raw probabilities are unchanged, i.e.:

$$P_{ensemble}(t \in dc_i) = P_{LSTM\text{-}BPVs}(t \in dc_i), \quad (3.8)$$

where $i \in \{1, \ldots, j-1, j+1, \ldots, 3008\}$. After calculating the combined raw probability, the final prediction is made with two approaches the same as in [84] based on all raw probabilities (as shown in Subsection 2.12.1).

The instructions of the hybrid system are shown in Appendix D (we modified the implementation codes from Michael Skinner[4], which are in PyTorch [69]).

3.5 Advantages of the Hybrid System for Product Classification

A wide range of ensemble methods have been developed in the field of ML so far, because they can effectively improve classification performance (as we have discussed in Section 2.12, many top participants used ensemble strategy). For details, [16] provided a good review on different ensemble methods. Generally, according to [16], there are 5 strands of methods to generate ensemble classifiers, namely Bayesian voting, to manipulate the training examples, to manipulate the input features, to manipulate the output targets and to inject randomness.

There are many ways to combine prediction results of classifiers within an ensemble. In [94], 3 approaches of ensemble, namely fixed rule, meta-classifier and weighted

[4] https://github.com/mcskinner/ecom-rakuten

combination, were used for sentiment classification task (a subtask of text classification). [94] found that weighted combination is the most attractive choice. Their weighted combination method involves finding the best weights by training a perceptron network.

In the ensemble of LSTM-BPV networks [84], the raw probabilities from individual networks were averaged to make the combined predictions:

$$P_{ensemble}(t \in dc_j) = \frac{\sum_{i=1}^{n} P_{LSTM\text{-}BPV,i}(t \in dc_j)}{n}, \tag{3.9}$$

where n is the number of networks in the ensemble, $P_{LSTM\text{-}BPV,i}(t \in dc_j)$ is the raw probability of product title t belonging to category id path dc_j generated from the ith LSTM-BPV network in the ensemble. This combination method is equivalent to sum rule and belongs to fixed rule approach.

In this thesis, we used the linear interpolation method to combine prediction results of our weighted KNN algorithm and ensemble of LSTM-BPV networks. This method is similar to weighted combination in [94], except that we manually tuned the weights (only 1 parameter (i.e. λ) to tune). One benefit of our ensemble strategy is that the hybrid system is able to take both word-level morphological information from exact match (through KNN) and character-level global semantic information of a product title (through LSTM-BPV(s)) into account. Although LSTM-BPV is better than conventional LSTM and capable of capturing more semantic signals that are beneficial to classification than LSTM, adding morphological signals from word-level exact match can make it even better. In a sense, this ensemble strategy is similar to that in Duet model [66], where morphological and semantic information are considered by 2 constituent models, namely local-interaction model and distributed model, respectively.

Chapter 4

Experiments

In this chapter, we will introduce the dataset involved in our experiments, experimental set-ups, evaluation methods and baselines. Then, we will evaluate our system's performance on SIGIR eCom Data Challenge Dataset. In addition, we will introduce parameter tuning of k in KNN algorithm and of b in BM25. We will also conduct further experiments by using different IR models other than BM25 as similarity function in KNN classifier, and we will examine parameter tuning of these models.

4.1 Dataset

The SIGIR eCom Data Challenge Dataset contains **Training Dataset** and **Test Dataset**. The Training Dataset ("rdc-catalog-train.tsv") has 800,000 product titles and corresponding category id paths, and the Test Dataset has 200,000 product titles ("rdc-catalog-test.tsv"). The gold standard for Test Dataset was named "rdc-catalog-gold.tsv". These files can be downloaded via this link[5]. A part of the Training Dataset is shown in Appendix A. The Data Challenge competition was organized by Rakuten Institute of Technology Boston (RIT-Boston).

According to our analysis, there are 3008 distinct category id paths in the Training

[5] https://docs.google.com/forms/d/e/1FAIpQLSfwcBlO_Z_aEXSbohScyj1YRRdsPULagEn28YYtca2ZkY5Ocw/viewform

Dataset. The maximum depth of the product taxonomy is 8, i.e. 8 levels. For example, "$1608 > 2320 > 2173 > 2878$" is of depth 4. The distribution of distinct category id paths over depth is shown in Figure 4.1a. The number of nodes (e.g. "2320" is a node in level 2) in each level of the product taxonomy is shown in Figure 4.1b. We can see that most of the category id paths are of depth 4.

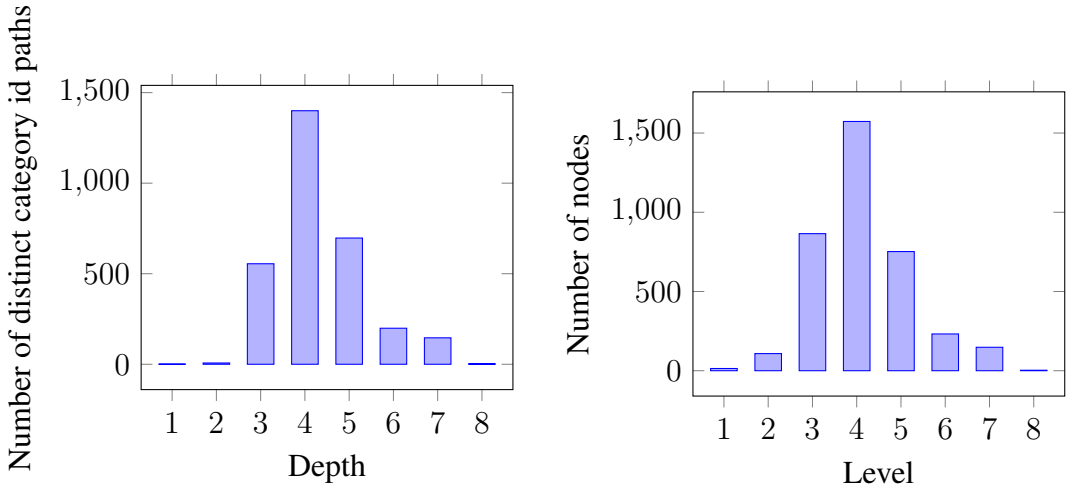
According to [50], the average and maximum word-level title length in the Dataset is 10.93 and 58 respectively. Also, [50] found the average and maximum character-level title length in the Dataset is 68.44 and 255 respectively. In addition, according to our experiments, the maximum WordPiece-level title length in the Dataset is 161.

For conducting further experiments with our KNN classifier, we split the Training Dataset into 2 training sets (i.e. **1-in-2-TRAIN** and **2-in-2-TRAIN**) of the same number of examples (i.e. 400,000 examples). Specifically, the Training Dataset with 800,000 examples was first split into 10 equal size subsets sequentially (so the 1st subset contains the first 80,000 examples), then the 1st, 3rd, 5th, 7th, 9th subsets were combined to make **1-in-2-TRAIN** and the remaining subsets were combined to make **2-in-2-TRAIN**. We then removed training sets' product category id paths to make two testing sets, i.e. **1-in-2-TEST** and **2-in-2-TEST**, respectively.

For doing experiments with LSTM-BPV(s) and BERT, we also used the validation set (200,000 instances extracted from the Training Dataset) and the training set (the remaining part of the Training Dataset, 600,000 examples) from the winner solution [84], denoted as **WINNER-VAL** and **WINNER-TRAIN** respectively.

4.2 Experimental Set-ups

Experiments were mainly done on a commercial server, with 64GB RAM and 6-core 2GHz CPU. Our implementation of KNN system is written in Java, and the instructions for doing experiments with it are shown in the Appendix B. Lucene is an open-source



(a) Distribution of distinct category id paths over depth

(b) Number of nodes at each level of the product taxonomy

Figure 4.1: Characteristics of product taxonomy in SIGIR eCom Data Challenge Dataset

IR software library which is able to do full text indexing and full text searching. This architecture is built on a document with fields of text (e.g. title field and abstract field). We conduct our experiments on the top of the Lucene API to get a full product list search for accurate product categorization. We also conduct experiments with Gensim [75], an open-source topic modelling library.

4.3 Evaluation Metrics

In this classification problem, the empirical results are evaluated with weighted-P, weighted-R and weighted-F1 respectively. The evaluation is based on exact category id path match. Because of the imbalanced distribution of classes in the dataset, weighted-{P, R and F1} are more appropriate than other evaluation methods like macro-{P, R and F1}. The P for a class cls_i is the fraction of the number of correctly predicted instances of the

class $tp(cls_i)$ and the number of instances predicted as the class $tp(cls_i) + fp(cls_i)$:

$$P(cls_i) = \frac{tp(cls_i)}{tp(cls_i) + fp(cls_i)} \tag{4.1}$$

The R for a class cls_i is the fraction of the number of correctly predicted instances of the class $tp(cls_i)$ and the number of true instances of the class $n_i = tp(cls_i) + fn(cls_i)$:

$$R(cls_i) = \frac{tp(cls_i)}{n_i} = \frac{tp(cls_i)}{tp(cls_i) + fn(cls_i)} \tag{4.2}$$

The F1 for a class cls_i is the harmonic mean of P and R for the class:

$$F_1(cls_i) = \frac{2 \times P(cls_i) \times R(cls_i)}{P(cls_i) + R(cls_i)} \tag{4.3}$$

Let m be the total number of classes in the dataset, N be the total number of instances in the dataset: $N = \sum_{i=1}^{m} n_i$ After P, R and F1 are calculated for each class, the weighted-$\{$P, R and F1$\}$ can be calculated as follows:

$$Weighted\text{-}P = \sum_{i=1}^{m} \frac{n_i}{N} P(cls_i), \tag{4.4}$$

$$Weighted\text{-}R = \sum_{i=1}^{m} \frac{n_i}{N} R(cls_i), \tag{4.5}$$

$$Weighted\text{-}F_1 = \sum_{i=1}^{m} \frac{n_i}{N} F_1(cls_i) \tag{4.6}$$

4.4 Baselines

The baseline methods we used are listed as follows:

The first one is fastText classifier [41]. Its hyperparameters are set as follows: "$dim = 300$, $minn = 4$, $maxn = 10$, $wordNgrams = 3$, $neg = 10$, $loss = ns$, $epoch = 3000$, $thread = 30$". The hyperparameter setting and preprocessing method are the same as those of RITB Baseline [50], which ranked 10th in the Data Challenge

(as shown in Table 2.1). We consider it as a strong baseline, as the number of epochs set is quite large (whereas the standard setting for number of epochs is within the range $[5, 50]$).

The second one is 1NN with document concatenation, a variation on our proposed KNN method, where product titles of the same category id path are concatenated into a single document. So, the total number of documents equals the number of distinct category id paths in training dataset. The system searches through these huge documents and use the corresponding category id path of the top document as prediction. We used optimized IB Model (IB-SPL-ADF-NormH1) ($c = 1.5$) as similarity function in 1NN, and replaced all digits (regular expression: "\d+", e.g. "88") with "0" during preprocessing.

4.5 Tuning k in KNN

We tried setting different values of k in KNN to see whether or not predicting based on individual match ($k = 1$) is better than on several matches ($k > 1$), since the individual match may be an outlier. In particular, k was set to 1, 3, 5, 7, 50 and 100.

Model	k Value	Weighted-P	Weighted-R	Weighted-F1
KNN-BM25	1	0.78	0.78	0.78
KNN-BM25	3	0.79	0.78	0.78
KNN-BM25	5	0.78	0.78	0.78
KNN-BM25	7	0.78	0.78	0.77
KNN-BM25	50	0.71	0.73	0.71
KNN-BM25	100	0.67	0.70	0.67

Table 4.1: **a subset of the Test Dataset**, performance comparison of KNN with BM25 as similarity function and different k values

The results in Table 4.1 show the official results of our primary submissions. In our experiment, with the setting of parameter $k = 3$ and BM25 as similarity function in KNN classification algorithm, our program achieved 0.79, 0.78 and 0.78 for the weighted-{P, R and F1} respectively in a subset of the Test Dataset. This subset (containing the first

20,000 product titles in the Test Dataset) is the one used for evaluation in the Stage 1 of the SIGIR eCom Data Challenge[6]. In the Stage 2 of the Data Challenge, the performance of our KNN-based system with $k = 3$ and BM25 as similarity function in the Test Dataset is shown in Table 2.1 (we ranked the 16th). As shown in Table 4.1, the results of k=1, 3 and 5 are roughly the same, because the top document matches of a query are highly similar to each other and thus have high probability of belonging to the same category. Also, the results for $k = 3$ rather than $k = 1$ is the best one among different settings of k, because the top 3 documents have high probability of having the same RSV with the query and thus the top 1 document's category may be an outlier. Generally, we can see that the prediction result declines as k increases, since titles with lower similarity are less likely to belong to the same category, as shown in the Example in Section 3.1.

4.6 Tuning BM25 Model

Apart from tuning k in KNN, we have tuned the parameters of the BM25 IR model to get better classification performance. We found a slight difference in between the tuning of the parameters. Specifically, with the same setting of $k = 1$ in KNN algorithm, by setting $k_1 = 1.2$, $b = 0.35$, we achieved slightly lower results of (0.78, 0.77, 0.77) for weighted-{P, R and F1} respectively than those of the default parameters ($k_1 = 1.2$, $b = 0.75$), i.e. (0.78, 0.78, 0.78) for weighted-{P, R and F1} respectively in the subset of the Test Dataset.

We conducted parameter tuning by fixing k value in KNN to 3, k_1 in BM25 to 1.2 and changing the value of b in BM25, with **1-in-2-TRAIN** and **2-in-2-TEST** as training set and tuning set respectively. We found the optimal weighted-F1 is obtained when $b = 0.92$ or 0.93, as shown in Figure 4.2. We can also see that weighted-R is consistently higher than weighted-F1 and weighted-P.

[6] https://sigir-ecom.github.io/data-task.html

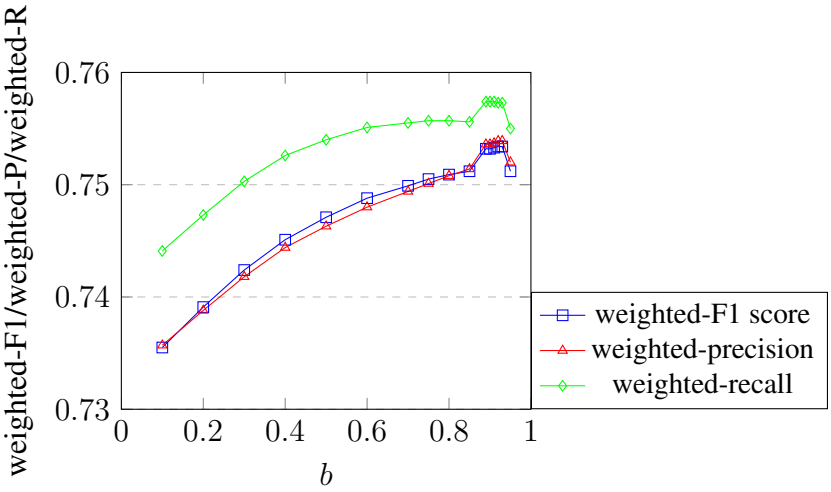


Figure 4.2: **2-in-2-TEST**, sensitivity of weighted-F1/weighted-P/weighted-R to b in BM25 model

4.7 Using Different IR Models as Similarity Function in Weighted-KNN

Apart from the BM25 model, we also used other IR models as similarity function in our weighted KNN classifier to see whether the classification performance can be improved or not. We tuned the parameters of the IR models with **1-in-2-TRAIN** and **2-in-2-TEST** as training set and tuning/test set respectively. Specifically, we tried the following IR models:

The first one is Lucene's implementation of VSM with TF-IDF weight (default setting). As this model is parameter-free, we do not need parameter tuning.

The second one is Lucene's implementation of Dirichlet LM [104]. We tuned the parameter μ of the model. We found the optimal weighted-F1 is obtained when $\mu = 0.1$ or 0.01, as shown in Figure 4.3. In contrast, the lowest weighted-F1 is obtained when there is no smoothing at all ($\mu = 0$). When $\mu > 0.1$, the weighted-F1 gradually decreases

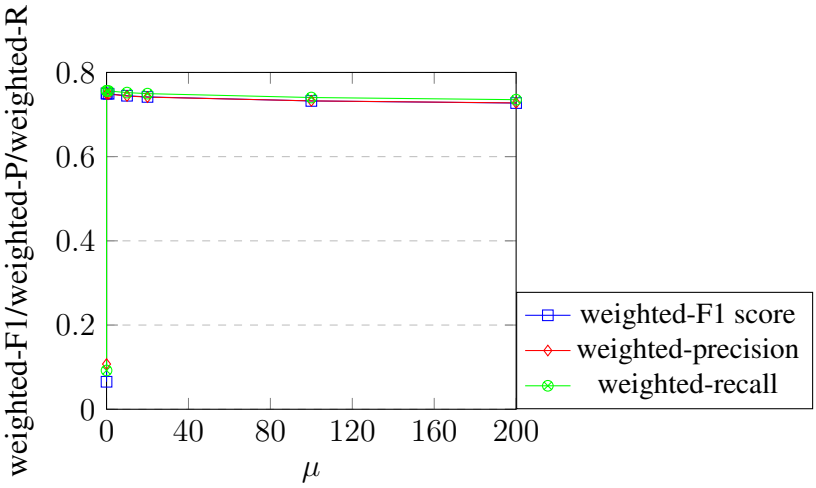


Figure 4.3: **2-in-2-TEST**, sensitivity of weighted-F1/weighted-P/weighted-R to μ in Dirichlet Language Model

as μ increases.

The third one is Lucene's implementation of Jelinek-Mercer LM [104]. We tuned the parameter λ of the model. We found the optimal weighted-F1 is obtained when $\lambda = 0.2$, 0.25 or 0.3, as shown in Figure 4.4. We found when $\lambda \in [0.1, 0.75]$, the weighted-F1 varies little. When $\lambda > 0.3$, the weighted-F1 gradually decreases as λ increases.

The fourth one is Lucene's implementation of IB Models [10]. We chose distribution, distribution parameter and TF normalization method according to the results obtained from IB models with default parameters, as shown in Table 4.2. We found the one using SPL distribution (as in Equation 2.31), average number of documents where a word occurs (ADF) as the distribution's parameter λ (as in Equation 2.33) and NormalizationH1, i.e. normalization model assuming a uniform distribution of TF (Equation 2.34), obtained the highest results. Then, we further tuned the normalization parameter c in Equation 2.34 and found the best result is obtained when c is set to 1.5, as shown in Figure 4.5.

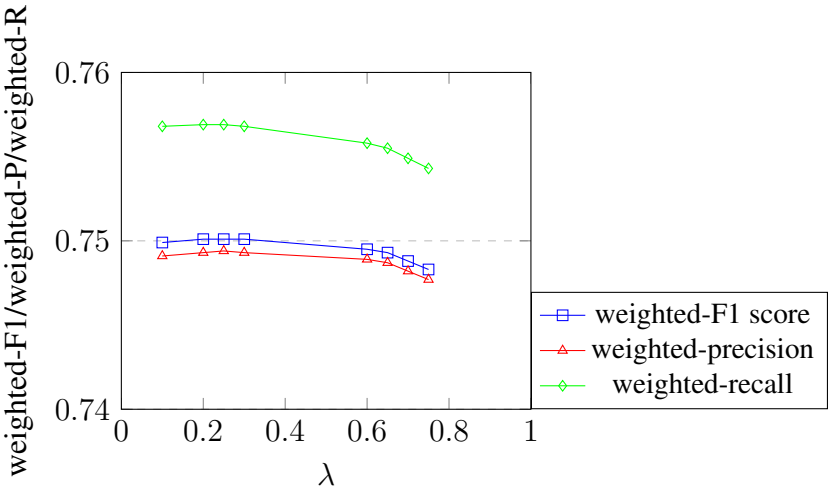


Figure 4.4: **2-in-2-TEST**, sensitivity of weighted-F1/weighted-P/weighted-R to λ in Jelinek-Mercer Language Model

The fifth IR model is the cosine similarity of Gensim's [75] Doc2vec embeddings [46]. We have built a similar product classification system with Gensim's [75] Doc2vec [46] document embedding cosine similarity as similarity function in weighted KNN algorithm. The instructions for doing experiments with it are shown in the Appendix C. The system used the same preprocessing method as we mentioned in Chapter 3.2, plus replacing all digits (regular expression: "\d+", e.g. "808") with "numericals ". But during analysis, only tokenization and case folding is done, i.e. no normalization. Testing is done using 16 threads in parallel. After hyperparameter tuning by training Doc2vec model on 1-in-2-TRAIN and testing on 2-in-2-TEST, the best result is obtained with the following hyperparameter setting:"$dm = 0, vector\_size = 300, dbow\_words = 0, dm\_concat = 1, dm\_tag\_count = 1, window = 10, min\_count = 2, epochs = 500, hs = 1$", as in Table 4.4 and 4.5. Different from reported in [46], PV-DBOW was found to perform better than PV-DM in this product classification task (as in Table 4.3). We tuned the window size, running 100 epochs, with the same setting for other hyperparameters, as

IR Model	Weighted-P	Weighted-R	Weighted-F1
IB-SPL-ADF-NormH1	0.7564	0.7595	**0.7557**
IB-SPL-ADF-NormH2	0.7545	0.7600	0.7547
IB-SPL-ADF-NormH3	0.7347	0.7434	0.7339
IB-SPL-ADF-NormZ	0.7459	0.7533	0.7461
IB-SPL-ADF-NoNormalization	0.7321	0.7398	0.7298
IB-LL-ADF-NormH1	0.7499	0.7570	0.7506
IB-LL-ADF-NormH2	0.7477	0.7554	0.7484
IB-LL-ADF-NormH3	0.7384	0.7469	0.7381
IB-LL-ADF-NormZ	0.7428	0.7510	0.7430
IB-LL-ADF-NoNormalization	0.7344	0.743	0.7333
IB-SPL-ATF-NormH1	0.7562	0.7591	0.7555
IB-SPL-ATF-NormH2	0.7543	0.7598	0.7546
IB-SPL-ATF-NormH3	0.7352	0.7438	0.7344
IB-SPL-ATF-NormZ	0.7461	0.7534	0.7463
IB-LL-ATF-NormH1	0.7498	0.7569	0.7506
IB-LL-ATF-NormH2	0.7477	0.7553	0.7484
IB-LL-ATF-NormH3	0.7386	0.7471	0.7384
IB-LL-ATF-NormZ	0.7429	0.751	0.7431

Table 4.2: **2-in-2-TEST**, performance comparison of KNN with different IB models with default parameters. The highest weighted-F1 is printed in bold.

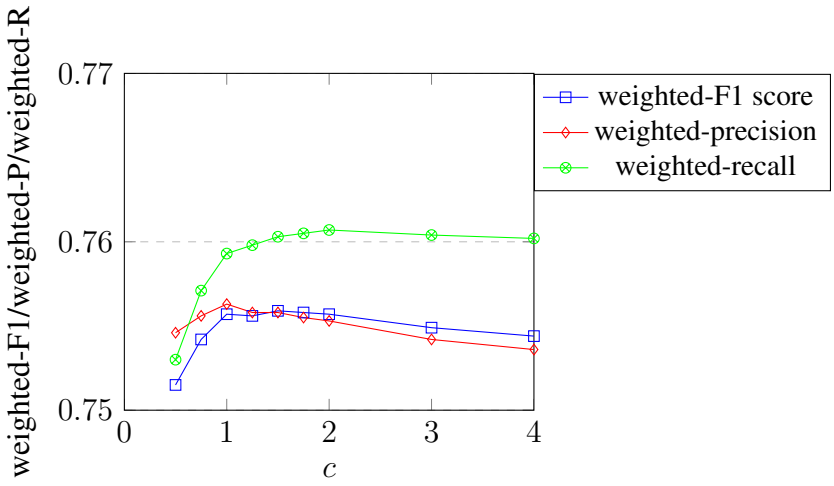


Figure 4.5: **2-in-2-TEST**, sensitivity of weighted-F1/weighted-P/weighted-R to c in NormalizationH1 within IB-SPL-ADF-NormH1

shown in Table 4.3. We use concatenation of context vectors instead of sum or average to preserve word positional information. We found removing very rare words by setting $min\_count = 2$ can help to improve performance in terms of weighted-F1 by around 0.001. This is because these words are noise that hinders generalization of the model. As cosine similarity is within the range $[-1, 1]$, we also used a threshold of 0.6 to prevent dissimilar items from contributing to prediction in KNN, which helped to improve weighted-F1 by 0.0011. Hierarchical softmax was used to train the model, since it is better for infrequent words compared to negative sampling.

Training method	Window size	Weighted-P	Weighted-R	Weighted-F1
PV-DBOW	9	0.7185	0.7287	0.721
PV-DBOW	10	0.7191	0.7292	**0.7216**
PV-DBOW	11	0.7185	0.7283	0.7208
PV-DM	9	0.0598	0.0622	0.0595
PV-DM	5	0.0832	0.0894	0.0851

Table 4.3: **2-in-2-TEST**, performance comparison of KNN with Doc2vec cosine similarity with different window size and different training method. The highest weighted-F1 is printed in bold.

Model	Weighted-P	Weighted-R	Weighted-F1
KNN-IB-SPL-ADF-NormH1	0.7558	0.7603	$0.7559^{\dagger\ddagger}$
KNN-IB-SPL-ADF-NormH1-digits-to-0	0.758	0.7632	$0.7583^{\dagger\ddagger}$
KNN-BM25	0.7545	0.7581	$0.7541^{\dagger\ddagger}$
KNN-VSM	0.7586	0.7553	$0.7546^{\dagger\ddagger}$
KNN-Dirichlet-LM	0.7499	0.757	$0.7506^{\dagger\ddagger}$
KNN-JelinekMercer-LM	0.7499	0.7574	$0.7507^{\dagger\ddagger}$
KNN-Doc2vec cosine similarity	0.7309	0.738	$0.7322^{\dagger\ddagger}$
fastText	0.7903	0.7754	**0.7807**
1NN-IB-SPL-ADF-NormH1-digits-to-0-concatenation-by-category	0.7509	0.6578	0.6755

Table 4.4: **2-in-2-TEST**, performance comparison of KNN with different IR models. The highest weighted-F1 is printed in bold.

Model	Weighted-P	Weighted-R	Weighted-F1
KNN-IB-SPL-ADF-NormH1	0.7574	0.7619	$0.7574^{\dagger\ddagger}$
KNN-IB-SPL-ADF-NormH1-digits-to-0	0.7592	0.7641	$0.7595^{\dagger\ddagger}$
KNN-BM25	0.7563	0.76	$0.756^{\dagger\ddagger}$
KNN-VSM	0.76	0.7564	$0.7558^{\dagger\ddagger}$
KNN-Dirichlet-LM	0.7518	0.7588	$0.7525^{\dagger\ddagger}$
KNN-JelinekMercer-LM	0.7515	0.7588	$0.7522^{\dagger\ddagger}$
KNN-Doc2vec cosine similarity	0.7323	0.7388	$0.7333^{\dagger\ddagger}$
fastText	0.7911	0.7766	**0.7819**
1NN-IB-SPL-ADF-NormH1-digits-to-0-concatenation-by-category	0.754	0.663	0.6808

Table 4.5: **1-in-2-TEST**, performance comparison of KNN with different IR models. The highest weighted-F1 is printed in bold.

Chapter 5

Analyses and Discussions

In this chapter, we will analyse the results obtained in Chapter 4. Also, we will perform various ablation analyses, such as analyzing the effect of removing stopwords, the effect of using different KNN weighting schemes and the effect of replacing digits with "0" during preprocessing. In the last 5 sections, we will introduce our experiments with LSTM-BPV networks and BERT models, examine the effect of batch size in fine-tuning BERT, and compare the performance of BERT pre-trained model [15] and that of OpenAI GPT pre-trained model [73].

5.1 Comparing KNN with Different IR Models

Using the same training set (1-in-2-TRAIN), test set (2-in-2-TEST) and $k = 3$ for weighted KNN, the best results from different IR models as KNN's similarity function are shown in Table 4.4. In addition, using the same training set (2-in-2-TRAIN), test set (1-in-2-TEST) and $k = 3$ for weighted KNN, the best results from different IR models as KNN's similarity function are shown in Table 4.5.

As shown in Table 4.4 and Table 4.5, fastText obtained the highest performance. This is probably because compared to most of IR models used by our KNN algorithm, it additionally takes word n-gram and character n-gram into account. Such information in

local dependencies and morphological information helps fastText outperform our KNN method. Another reason is that it has been trained for an unusually large number of epochs (3000). In contrast, 1NN with document concatenation obtained the lowest results, as shown in Table 4.4 and Table 4.5. This is probably because in this method, documents are much more lengthy and thus contain more noise, which makes it more difficult to match them precisely with query. We also conduct 2-tailed paired t-test ($\alpha = 0.05$) on weighted-F1 of our proposed methods and those of 1NN with document concatenation baseline, and we use "†" in Table 4.4 and Table 4.5 to indicate a significant difference between them. We also use "‡" in Table 4.4 and Table 4.5 to indicate a significant difference between our proposed methods and fastText. It is also interesting to note that the weighted-R obtained by KNN with IR Models are consistently higher than weighted-F1 obtained by them.

Furthermore, among all IR models used by our KNN algorithm, IB Model, i.e. IB-SPL-ADF-NormH1, and cosine similarity of Doc2vec embeddings obtained the highest and lowest performance in terms of weighted-F1 respectively. The IB model works better than BM25, which is in line with the observation in [10]. The neural IR model (i.e. cosine similarity of Doc2vec embedding) could not get good results, probably because all terms are treated equally instead of weighted based on their prominence (e.g. IDF) within the document collection (like in VSM with TF-IDF weight) (this is in line with the observation in [22]).

5.2 Impact of Removing Stopwords

We have also compared results of using the stopword filter in Lucene's standard analyser to those of not using it. We found that using stopword filter would slightly reduce the results (around 0.002 in F1), partly because after stop word removal, documents (training titles) may have no term. Also, this suggests that some stopwords are useful for product

classification.

5.3 Impact of Replacing Digits with "0"

We have also tried reducing feature space by replacing all digits (regular expression: "\d+", e.g. "356") with "0". We found that doing so would at least double the running time of our KNN system and significantly increase the Weighted-F1 ($p = 0.0424$ in 2-tailed paired t-test ($\alpha = 0.05$)) (as shown in Table 4.4 and 4.5). This shows that numbers are noise for product taxonomy classification.

5.4 Impact of Removing Infrequent Words

We have also compared results of setting different minimum DF threshold in our KNN classifier. DF for a word w_i is the number of documents containing that word in document collection C. A word whose DF is less than DF threshold ($DF(w_i) < DF_{threshold}$) would be removed. We used the same IR model, i.e. IB-SPL-ADF-NormH1, and set KNN's $k = 3$. We found that using a DF threshold of 2 could slightly improves the performance by 0.0001 in terms of weighted-F1 (although this is quite small difference). This is because very rare word is likely to be noise. However, when DF threshold is set larger than 3 (i.e. 3, 4, 5, 6, 10 and 100), the performance would degrade in terms of weighted-F1.

5.5 Impact of Using Standard Filter for Normalization

We found when there is no normalization by standard filter during analysis, the results in terms of weighted-F1 will slightly decrease (about 0.0002). On one hand, this means normalization has little effect on the performance, probably because most of product ti-

tles are noun phrases, which require little normalization. On the other hand, this suggests that normalization can help product classification.

5.6 Comparing KNN with Different Weighting Schemes

Originally, we [31] implemented a KNN with simple voting where the number of occurrences of a distinct category among top k matched products' categories was used instead of category score. And that one produced slightly lower results (about 0.0006 lower in weighted-F1) compared to the current weighted KNN we use (using the same training set (1-in-2-TRAIN), test set (2-in-2-TEST) and $k = 3$). This is because categories follow an unbalanced distribution and thus a category with large number of examples is more likely to win in KNN with simple voting.

We also implemented biweight kernel weighted KNN [28], where the category score is calculated as follows instead of using Equation 3.2:

$$Cat(dc_m) = \sum_{i=1}^{n-1} \frac{15}{16} \times (1 - (\frac{RSV(t_j, pt_n)}{RSV(t_j, pt_i)})^2)^2 \times \mathbf{1}_{\{pt_i \in dc_m\}}, \tag{5.1}$$

This produces slightly lower results (about 0.002 in weighted-F1) compared to the current weighted KNN we use (using the same training set (1-in-2-TRAIN), test set (2-in-2-TEST) and $k = 3$). This is probably because the kernel function's range is $[0, \frac{15}{16})$, which narrows the difference between very relevant document and not that relevant document. Another reason is that the nth nearest neighbour cannot contribute to KNN voting.

Furthermore, inspired by the smoothing used in language models, to better tackle the problem of unbalanced category distribution, we implemented a weighted-KNN algorithm with weighted category and smoothing, where each category is further weighted inversely proportional to its number of occurrences in the training set. The category score is calculated as follows instead of using Equation 3.2:

$$Cat(dc_m) = ((1-\alpha)\frac{128}{N_{pt \in dc_m}} + \alpha\frac{128}{N_{avg}}) \sum_{i=1}^{n} RSV(t_j, pt_i) \times \mathbf{1}_{\{pt_i \in dc_m\}}, \quad (5.2)$$

where α is a smoothing parameter, $N_{pt \in dc_m}$ is the number of instances belonging to dc_m in the training set, $N_{avg} = \frac{|TR|}{|\{dc_i | pt_j \in dc_i, pt_j \in TR\}|}$ (the average number of instances belonging to a category in training set TR) and $\{dc_i | pt_j \in dc_i, pt_j \in TR\}$ is the set of distinct categories in TR. As shown in Figure 5.1, with $\alpha = 0.95$, we got the optimal weighted F1, slightly higher (0.0005 higher in weighted-F1) than the current weighted KNN we use (using the same training set (1-in-2-TRAIN), test set (2-in-2-TEST) and $k = 3$). In contrast, with other values of α ($\alpha \leq 0.9$), the results are not better than the current weighted KNN we use (Equation 3.2), which is actually a special case of this weighted KNN with weighted category and smoothing (by setting α to 1). This is probably because rare categories are boosted too much. Nevertheless, our experiments suggest that incorporating category distribution information can improve the classification performance, which is in line with [90, 51]. The drawback is, however, that we need to tune an additional parameter (i.e. α).

5.7 Combining Prediction Results of KNN with LSTM-BPV Networks

We conducted parameter tuning of λ in Equation 3.7, using the **WINNER-VAL** and **WINNER-TRAIN** as testing/tuning set and training set respectively. The training of LSTM-BPV networks was mainly done on Paperspace Notebooks with Nvidia P100 or V100 GPU (16Gb memory) acceleration. We modified and used the implementation codes[7] from [84]. The instructions for conducting experiments with the ensembles are

[7] https://github.com/mcskinner/ecom-rakuten

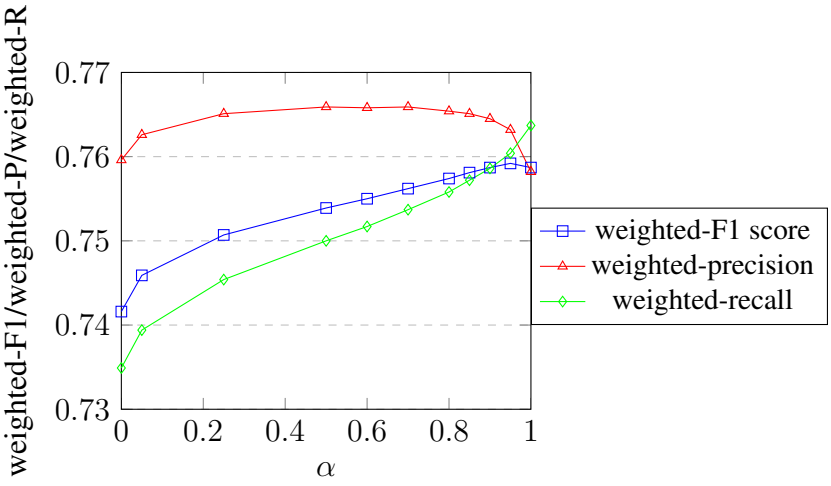


Figure 5.1: **2-in-2-TEST**, sensitivity of weighted-F1/weighted-P/weighted-R to α in weighted KNN with weighted category and smoothing

shown in Appendix D. We first trained a single forward LSTM-BPV network with default hyperparameters (i.e. $number\_of\_training\_epochs = 40$, $max\_learning\_rate = 0.8$, $learning\_rate\_factor = 20$, $highest\_momentum = 0.95$, $lowest\_momentum = 0.85$, $size\_of\_character\_embedding = 50$, $number\_of\_hidden\_units\_within\_a\_LSTM\_layer = 512$) as our base model. We then trained a reverse LSTM-BPV network with default settings and used a bidirectional ensemble of this network and the first one, denoted as B-LSTM-BPV, as another base model. The sensitivity of λ is shown in Figure 5.2.

The green line in Figure 5.2 shows the performance of our weighted KNN system with IB-SPL-ADF-NormH1 model as similarity function on **WINNER-VAL**. As we can see, generally, the performance in terms of weighted-F1 slightly decreases as λ increases within the range $[0, 0.9]$ when using base model LSTM-BPV without F1 optimizing, and we get the maximum F1 when setting $\lambda = 0$. However, when using base model B-LSTM-BPV without F1 tuning, we get the maximum F1 when $\lambda = 0.2$.

When using base model LSTM-BPV with F1 optimizing, the performance of the ensemble slightly fluctuates within the range $[0, 0.9]$, and the maximum is obtained with $\lambda = 0.6$. Similarly, when using base model B-LSTM-BPV with F1 tuning, we get the maximum value of weighted-F1 when $\lambda = 0.6$. When λ increases within the range $[0.9, 0.96]$, the weighted-F1 performance of both ensembles with F1 tuning drops dramatically, before gradually decreasing within $[0.96, 1]$. When λ is set higher than 0.91, the ensemble could not perform better than its consitituent KNN system. This is probably because, mathematically, the f1-tuning ensemble's prediction's probability of a given test title t belonging to a specific dc_m predicted by our KNN system $P_{ensemble}(t \in dc_m)$ is guaranteed to be higher than the original $\max_{i \in \{1,\ldots,3008\}} P_{LSTM\text{-}BPVs}(t \in dc_i)$ when $\lambda > 0.9091$ (proof attached in Appendix F) and thus the ensemble's prediction is too much affected by its constituent KNN system. In contrast, when λ increases within the range $[0.9, 1]$, the weighted-F1 performance of both ensembles without F1 tuning first drops a bit before λ reaching 0.99, and after that it drops dramatically. We think this is because the ensemble's prediction is too much predominated by its constituent KNN system.

Nevertheless, when λ is within the range $[0, 0.91]$, independent of whether the base model LSTM-BPV/B-LSTM-BPV is tuned to optimize weighted-F1 or not, combining with LSTM-BPV network or B-LSTM-BPV networks can improve the performance of our KNN system.

Moreover, when $\lambda = 0.6$, the ensembles could outperform their base models (LSTM-BPV/1 pair of B-LSTM-BPV/2 pairs of B-LSTM-BPVs/3 pairs of B-LSTM-BPVs) on the Test Dataset, as shown in Table 5.1. In this table, we also show the improvement rate in weighted-F1 when using ensemble instead of its base model in the brackets. Generally, we observe an average of around 0.001 or 0.004 increase in weighted-F1 compared to different base models when combining our system's prediction with them with or without

F1 optimization respectively (however, the differences are not significant according to 2-tailed paired t-test ($\alpha = 0.05$) on weighted-F1). This suggests that word-level matching of our system can help character-level neural networks generalize better. Interestingly, the increase of weighted-F1 with F1 tuning is always greater than that without F1 tuning.

Model	Tune F1 or not	Weighted-P	Weighted-R	Weighted-F1
KNN with IB-SPL-ADF-NormH1 (used whole Training Dataset)	not applicable	0.7866	0.7887	0.7851
LSTM-BPV	false	0.8097	0.8152	0.8094
ensemble of LSTM-BPV and KNN	false	0.8108	0.8162	0.8104 (+0.13%)
LSTM-BPV-fine-tuned	false	0.8114	0.8164	0.8098
ensemble of LSTM-BPV-fine-tuned and KNN	false	0.8132	0.8183	0.8117 (+0.23%)
LSTM-BPV	true	0.8332	0.8095	0.8179
ensemble of LSTM-BPV and KNN	true	0.8421	0.8121	0.8236 (+0.69%)
LSTM-BPV-fine-tuned	true	0.8467	0.8054	0.8212
ensemble of LSTM-BPV-fine-tuned and KNN	true	0.8571	0.8091	0.8282 (+0.85%)
B-LSTM-BPV	false	0.8257	0.8304	0.8246
ensemble of B-LSTM-BPV and KNN	false	0.8269	0.8314	0.8256 (+0.12%)
B-LSTM-BPV-fine-tuned	false	0.8267	0.8309	0.8243
ensemble of B-LSTM-BPV-fine-tuned and KNN	false	0.8282	0.8324	0.8258 (+0.18%)
B-LSTM-BPV	true	0.848	0.8247	0.8324
ensemble of B-LSTM-BPV and KNN	true	0.8557	0.8262	0.8371 (+0.55%)
B-LSTM-BPV-fine-tuned	true	0.8592	0.8199	0.8345
ensemble of B-LSTM-BPV-fine-tuned and KNN	true	0.8681	0.8226	0.8403 (+0.70%)
2 pairs of B-LSTM-BPV	false	0.8329	0.8369	0.8308
ensemble of 2 pairs of B-LSTM-BPV and KNN	false	0.8335	0.8376	0.8315 (+0.085%)
2 pairs of B-LSTM-BPV-fine-tuned	false	0.8333	0.8368	0.8301
ensemble of 2 pairs of B-LSTM-BPV-fine-tuned and KNN	false	0.8349	0.8383	0.8316 (+0.18%)
2 pairs of B-LSTM-BPV	true	0.8541	0.831	0.8383
ensemble of 2 pairs of B-LSTM-BPV and KNN	true	0.8612	0.8320	0.8423 (+0.48%)

2 pairs of B-LSTM-BPV-fine-tuned	true	0.8639	0.8259	0.8394
ensemble of 2 pairs of B-LSTM-BPV-fine-tuned and KNN	true	0.8719	0.8275	0.8444 (+0.59%)
3 pairs of B-LSTM-BPV	false	0.8347	0.8385	0.8324
ensemble of 3 pairs of B-LSTM-BPV and KNN	false	0.8357	0.8394	0.8333 (+0.11%)
3 pairs of B-LSTM-BPV-fine-tuned	false	0.8351	0.8385	0.8318
ensemble of 3 pairs of B-LSTM-BPV-fine-tuned and KNN	false	0.8366	**0.8397**	0.8330 (+0.16%)
3 pairs of B-LSTM-BPV	true	0.8553	0.8329	0.8397
ensemble of 3 pairs of B-LSTM-BPV and KNN	true	0.8626	0.8341	0.8440 (+0.51%)
3 pairs of B-LSTM-BPV-fine-tuned	true	0.8655	0.8280	0.8412
ensemble of 3 pairs of B-LSTM-BPV-fine-tuned and KNN	true	**0.8729**	0.8293	**0.8457** (+0.54%)
fastText (RITB-Baseline) [50] (trained on whole Training Dataset)	not applicable	0.8276	0.8077	0.8142
BERT-large	false	0.8386	0.8358	0.8356
BERT-large	true	0.8420	0.8341	0.8350

Table 5.1: **Test Dataset**, performance comparison of ensembles and base models. The highest value is printed in bold.

5.8 Fine-tuning Trained LSTM-BPV Network on WINNER-VAL

In this section, we fine-tuned LSTM-BPV network trained with **WINNER-TRAIN** ($600,000$ examples) from last section on **WINNER-VAL** (200,000 examples) to get higher performance in the Test Dataset. In this way, the model could be trained on the whole Training Dataset. Specifically, we fine-tuned the model with the same hyper-parameters as those during model training, except max learning rate set to 0.08, batch size set to 32 and number of epochs noe set to 1/4/5/6/8/10/15/20 (thus, we got 8 fine-tuned models). This is because fine-tuning usually requires smaller batch size, smaller learning rate and smaller noe. We report the empirical results on **WINNER-VAL** and **Test Dataset** in Figure 5.3. The green/yellow line in Figure 5.3 shows the performance

of our weighted KNN system with IB-SPL-ADF-NormH1 model as similarity function on **WINNER-VAL/Test Dataset** respectively (the performance on **Test Dataset** is better than that on **WINNER-VAL**, as we used full **Training Dataset** (800,000 examples) instead of **WINNER-TRAIN** (600,000 examples)).

As shown in Figure 5.3, the performance of the model improves consistently in **WINNER-VAL** as number of fine-tuning epoch noe increases. This is because during fine-tuning, the **WINNER-VAL** is used for training as well as validation. However, in the Test Dataset, the performance in terms of weighted-F1 first peaks when $noe = 5$ and then decreases as noe increases, which suggests that when $noe > 5$, the model begins to overfit on **WINNER-VAL**. We found fine-tuning the model for 5 epochs could improve absolute weighted-F1 by 0.04%/0.33% without/with F1 tuning respectively. Furthermore, the increase in weighted-F1 with F1 tuning is again greater than that without F1 tuning, which suggests the F1 tuning method is effective and robust.

We also fine-tuned other LSTM-BPV models from last section for 5 epochs in the same way. We present the results in Table 5.1. As shown in the table, combining our KNN algorithm with base models could consistently improve performance in Test Dataset. Furthermore, sometimes fine-tuning models degraded the performance in terms of weighted-F1 without F1 tuning (as shown in B-LSTM-BPV and 2 pairs of B-LSTM-BPV in the table). This is probably because the ensemble model forgot knowledge gained from **WINNER-TRAIN** as it acquired new knowledge. And this phenomenon is similar to catastrophic forgetting during continual training (i.e. a model is trained continuously on different tasks) observed in [101].

5.9 Fine-tuning Pre-trained BERT Model

Let H be the number of hidden units in a layer, i.e. hidden size, in the Transformer, C be the number of target classes. We used BERT-large uncased pre-trained model for

fine-tuning, the number of layers in the model is $L = 24$, hidden size is $H = 1024$ and number of attention heads is $A = 16$. In particular, we add a fully-connected linear layer (or classification layer) $W \in \mathbb{R}^{C \times H}$ ($C = 3008$ in this product classification task) on the top of the deep bidirectional Transformer (This method was presented in [15] for tackling single sentence classification problem). The instructions for using this classifier are presented in Appendix E (we modified the implementation codes from Google AI Language Team[8]).

We can achieve similar results to those of LSTM-BPV models by fine-tuning a single pre-trained BERT model. Specifically, we used the same validation set and training set as those in Section 5.7 (i.e. **WINNER-VAL** and **WINNER-TRAIN**). Our experiments relating to BERT were conducted on a Google Cloud Virtual Machine (VM) with pre-emptible Tensor Processing Unit (TPU) v3 (128Gb High Bandwidth Memory (HBM)). We used the same optimizer as in [15], i.e. Adam [43]. The hyperparameters used are as follows: learning rate set to 5×10^{-5}, learning rate warmup portion set to 0.1, linear decaying of learning rate after warmup, batch size set to 64. We chose a relative small batch size, as the minimum effective batch size for a TPU is 64 and according to [60], using smaller batch size often enables faster convergence, which is also confirmed in our experiments (we will examine this in Section 5.10). We first used a max sequence length of 128 for the first 3 epochs, after that we used 176 instead (this would have little effect on final classification performance, as most of the product titles' WordPiece-level lengths are less than 128 and we fine-tuned the model for many epochs). We chose max sequence length to be a multiple of number of attention heads ($A = 16$) for efficient computing. We trained a total of 40 epochs.

We used a continuing training scheme that evaluates the performance in the 5th, 8th, 10th, 15th, 20th, 30th and 40th epoch. So the learning rate actually decreased to 0 and

[8] https://github.com/google-research/bert

then restarted in the end of the 5th, 8th, 10th, 15th, 20th, 30th and 40th epoch, which is similar to Stochastic Gradient Descent with Warm Restarts (SGDR) [57]. This may produce slightly different results compared to typical learning rate scheme that reduce learning rate to 0 at the end of the whole training process. We report classification performance of BERT to number of fine-tuning epochs on both **WINNER-VAL** and **Test Dataset** in Figure 5.4.

As shown in Figure 5.4, we achieved high performance in terms of weighted-F1 after fine-tuning for 5 epochs. Generally, we can see that training more epochs boosts the performance in terms of weighted-F1. In both **WINNER-VAL** and **Test Dataset**, the performance of F1 tuned is better than that of no F1 tuning only when number of epochs $e \leq 20$. Interestingly, when $e \geq 30$, the performance of BERT with F1 tuning in terms of weighted-F1 is slightly lower than that of BERT without F1 tuning, which is not as expected. This is probably because the model has almost reached its full potential, as the performance only increased slightly (less then 0.0007 in terms of absolute increase in weighted-F1 in both cases) during the last 10 epochs, i.e. when $e \in [30, 40]$. The performance of this single model (as shown in Table 5.1) is slightly better than a bidirectional ensemble of 2 LSTM-BPVs regardless of whether the ensemble is F1 tuned or not. However, the fine-tuned BERT model's performance is still slightly worse than that of a F1-tuned bidirectional ensemble of 4 LSTM-BPVs in terms of weighted-F1.

As advertised in [15], this method is straightforward and requires only a few modifications to the model architecture for text classification. The performance is also comparable to the SotA. However, the computational cost is still quite high. Fine-tuning the pre-trained BERT-large uncased model for 1 epoch takes around 40 minutes on a newly-developed Cloud TPU v3, although we set 1,000 batches per training loop. This is probably because of large model size of BERT and relatively small batch size used for fine-tuning. Another reason is that, because products' titles have various length (from

1 to 161 in terms of WordPiece-level length), the fixed sequence length requirement in BERT causes excessive use of padding and thus makes computation less efficient.

5.10 Fine-tuning BERT with Larger Batch Size

We tried using larger batch size for fine-tuning BERT-large uncased model to use TPU more efficiently and thus to reduce training time. Specifically, we used the same hyper-parameters as those in the previous section, except for batch size (256) and learning rate (2.5×10^{-4}) (this learning rate is still small, slightly more than linearly increasing the learning rate in the previous section with respect to batch size (2×10^{-4})). We got the loss curve (training loss (vertical axis) versus number of iterations (horizontal axis)) of fine-tuning the model on **WINNER-TRAIN**, as shown in Figure 5.5. We can see that, with a smoothing of 0.6, the training loss first gradually decreases within the first 17.5 epochs (approximately), and after that it diverges drastically (so we stopped training after fine-tuning it for around 24.5 epochs). This suggests that smaller batch size can offer more stable training performance, which is in line with [60].

5.11 Comparing the Performance of Pre-trained BERT and OpenAI GPT

Apart from BERT model, we also tried fine-tuning pre-trained OpenAI GPT model [73] for this classification task. We fine-tuned the OpenAI GPT model for 3 epochs ($noe = 3$) using SGDM [88] optimizer. We used a batch size of 80, and peak learning rate was tuned to be 0.000625. We used a 1cycle learning scheme, as in [85], with highest momentum set to 0.95, lowest momentum set to 0.85 and learning rate factor set to 20. We compare it with BERT-base uncased model [15] here, as they have the same size and similar structure. We only fine-tuned pre-trained BERT-base model for 1 epoch ($noe = 1$) with

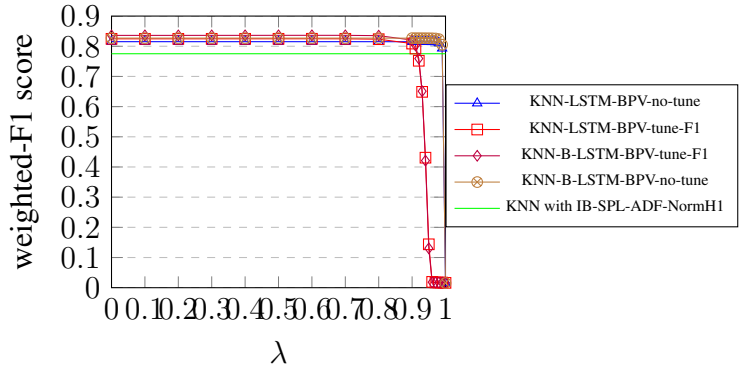
Adam optimizer, batch size of 120 and learning rate of 0.00025. We used learning rate warmup for the first 500 steps and linear decaying of learning rate afterwards. For both models, we used a max sequence length of 70. The results in **WINNER-VAL** and **Test Dataset** are shown in Table 5.2.

Model	Tune F1 or not	noe	Dataset	Weighted-P	Weighted-R	Weighted-F1
OpenAI GPT	false	3	**WINNER-VAL**	0.64	0.6995	0.6581
BERT-base	false	1	**WINNER-VAL**	0.6799	0.7285	0.6934
OpenAI GPT	true	3	**WINNER-VAL**	0.7136	0.6906	0.693
BERT-base	true	1	**WINNER-VAL**	0.743	0.7203	**0.7234**
OpenAI GPT	false	3	**Test Dataset**	0.642	0.7012	0.6596
BERT-base	false	1	**Test Dataset**	0.6792	0.7286	0.693
OpenAI GPT	true	3	**Test Dataset**	0.7141	0.6917	0.6939
BERT-base	true	1	**Test Dataset**	0.7429	0.7201	**0.7228**

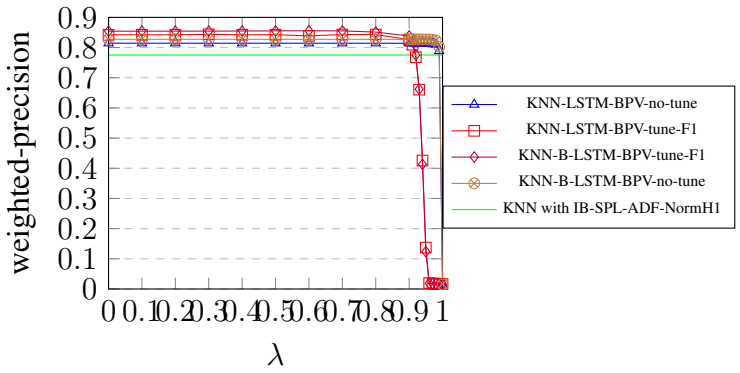
Table 5.2: **WINNER-VAL/Test Dataset**, performance comparison of BERT-base and OpenAI GPT. The highest weighted-F1 achieved on each dataset is printed in bold.

As shown in the table, it is clear that pre-trained BERT-base model is better than pre-trained GPT model, as it converged faster (1 epoch versus 3 epochs) and obtained higher performance. This observation is in line with that in [15]. According to [15], such difference is caused by the difference in the size of pre-training datasets (BERT was pre-trained on larger training corpus) and the difference in pre-training tasks (BERT's pre-trained tasks are MLM and NSP, whereas OpenAI GPT's is LtR NLM).

WINNER-VAL, sensitivity of weighted-F1 to λ



WINNER-VAL, sensitivity of weighted-P to λ



WINNER-VAL, sensitivity of weighted-R to λ

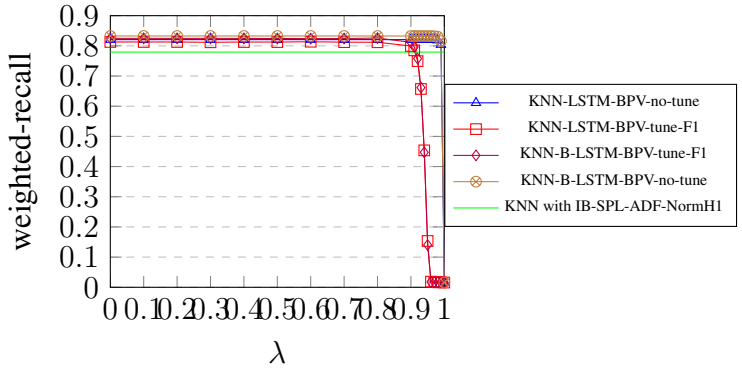
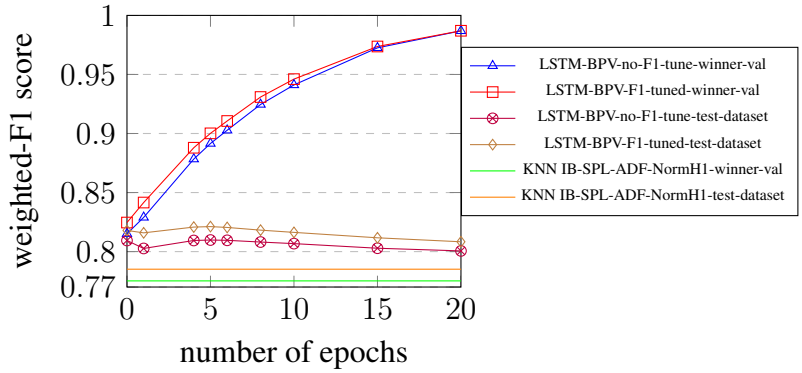
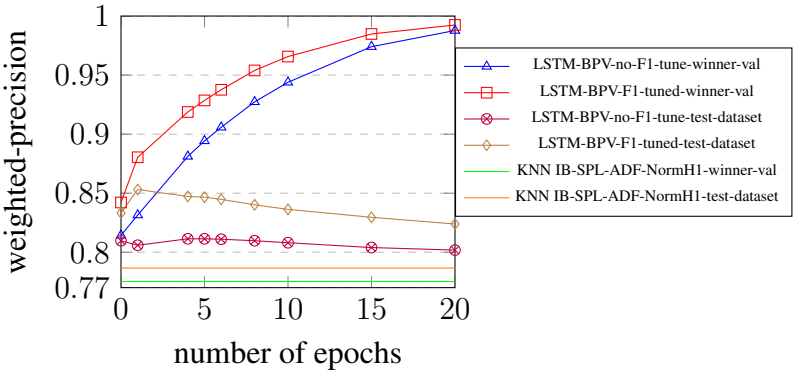


Figure 5.2: **WINNER-VAL**, sensitivity to λ in the ensemble of KNN with IB-SPL-ADF-NormH1 and LSTM-BPV/B-LSTM-BPV

WINNER-VAL/Test Dataset, sensitivity of weighted-F1 to number of epochs



WINNER-VAL/Test Dataset, sensitivity of weighted-P to number of epochs



WINNER-VAL/Test Dataset, sensitivity of weighted-R to number of epochs

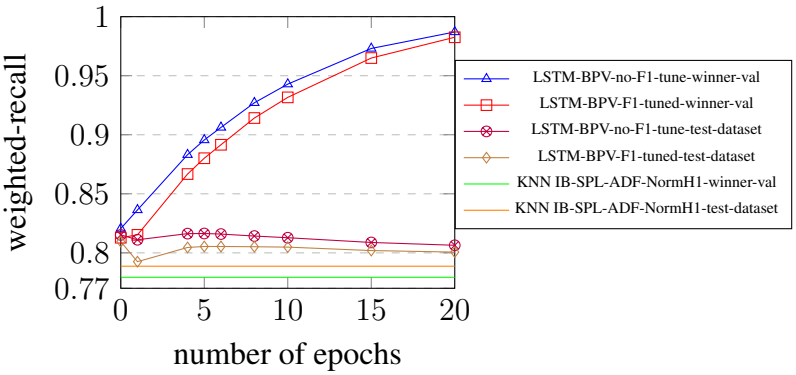
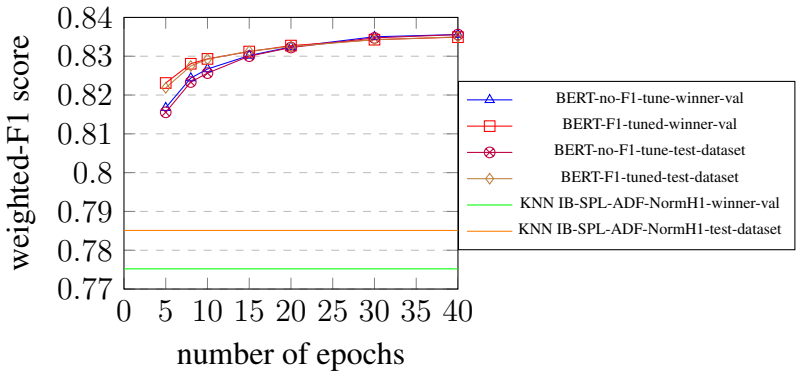
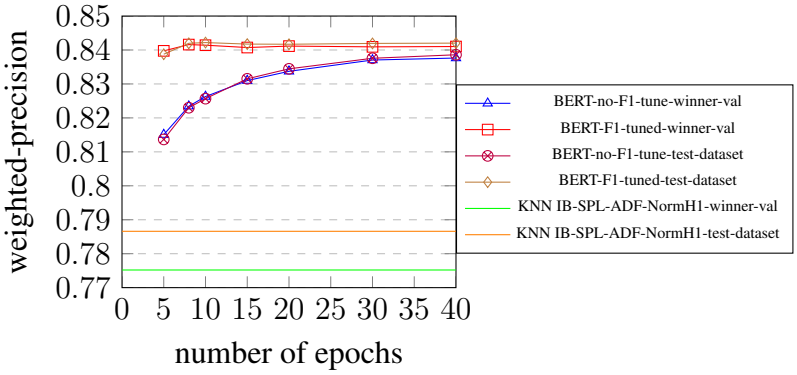


Figure 5.3: **WINNER-VAL/Test Dataset**, sensitivity to LSTM-BPV's number of epochs fine-tuned

WINNER-VAL/Test Dataset, sensitivity of weighted-F1 to number of epochs



WINNER-VAL/Test Dataset, sensitivity of weighted-P to number of epochs



WINNER-VAL/Test Dataset, sensitivity of weighted-R to number of epochs

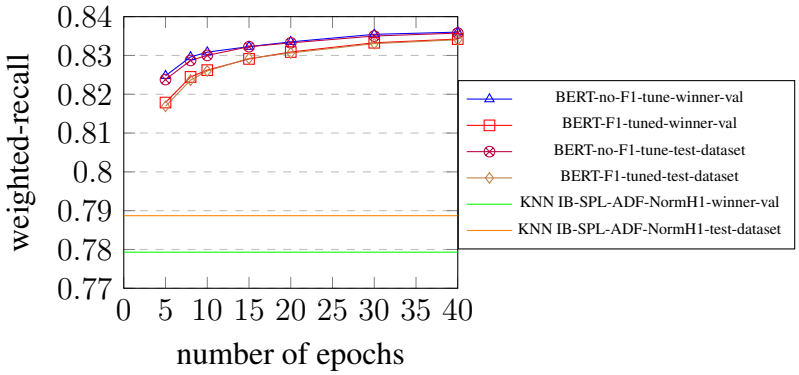


Figure 5.4: **WINNER-VAL/Test Dataset**, sensitivity to BERT's number of epochs fine-tuned

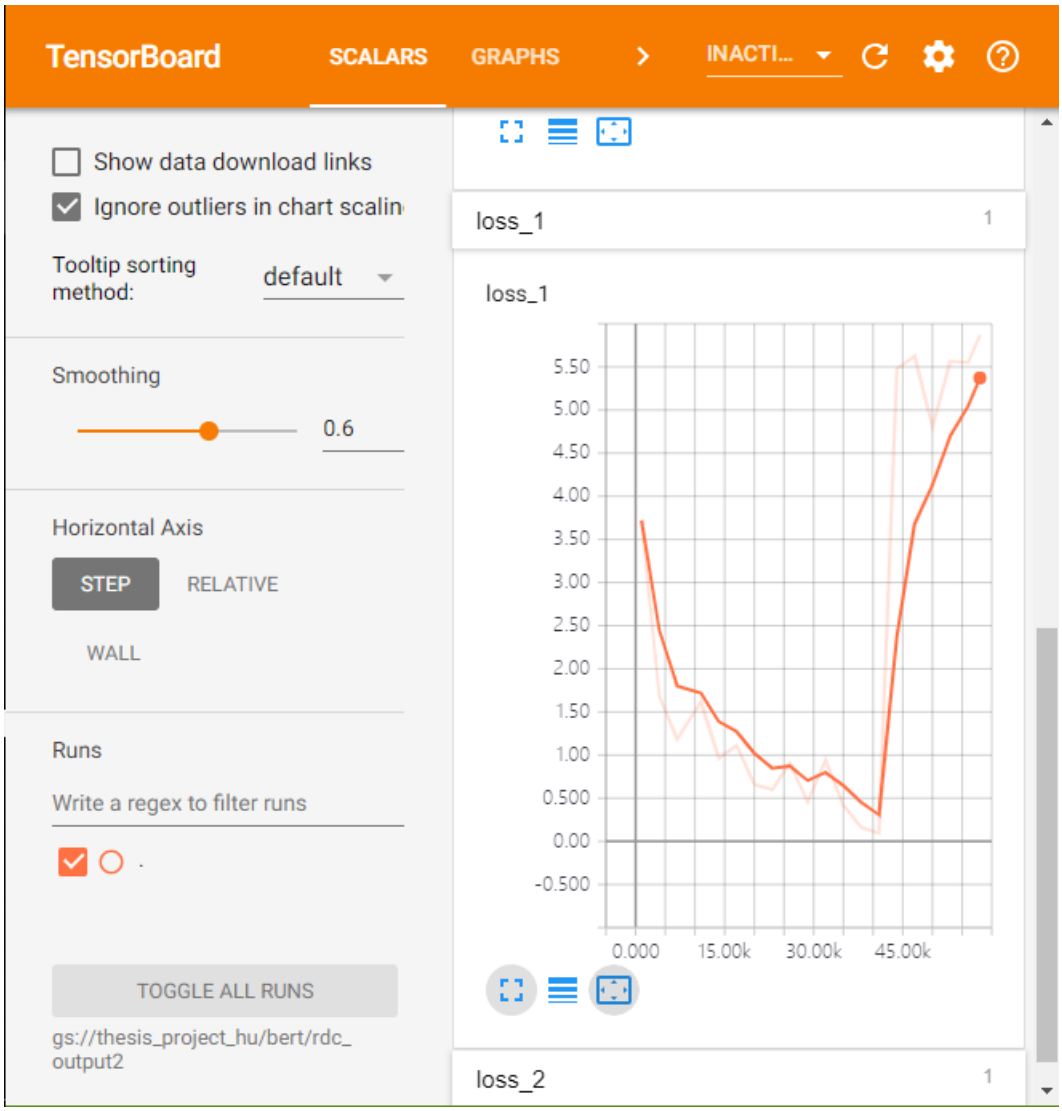


Figure 5.5: **WINNER-TRAIN**, the training loss curve of fine-tuning BERT-large uncased with batch size of 256 and learning rate of 2.5×10^{-4} for 40 epochs. (pink line): raw plot; (red line): with 0.6 smoothing

Chapter 6

Conclusions and Future Work

In this chapter, we will conclude this thesis with conclusions, the impact of this work and our future research direction.

6.1 Conclusions

In this thesis, we proposed a product taxonomy classification system based on weighted KNN with an IR model as similarity function. The system is fast and scalable compared to other methods such as fastText, although the performance is slightly lower than them. This suggests that our proposed method can serve as a fast and relatively good baseline. Within the IR models we used as similarity function in our KNN algorithm, IB Model and cosine similarity of Doc2vec document embedding obtained the highest and lowest classification performance, respectively. In addition, we proposed a hybrid approach that combines our KNN system with advanced neural network method, i.e. LSTM-BPV(s). This approach could improve the overall classification results of LSTM-BPV(s) and achieved performance comparable to the SotA (0.8457 in terms of weighted-F1 in the Test Dataset, trained on the **Training Dataset**) in this product taxonomy classification task. Apart from this, we conducted experiments with the pre-trained BERT-large uncased model and also obtained good results after fine-tuning it for 40 epochs (0.8356

in terms of weighted-F1 in the Test Dataset, trained on **WINNER-TRAIN**).

6.2 Impact of My Thesis Work

In my thesis work, we found the following insights:

The first insight is that IR model can be used as similarity function in weighted KNN algorithm to generate relatively good prediction results on this product categorization task. We can probably improve the classification performance through using supervised neural IR model like K-NRM [95] as similarity function in KNN. According to [6], K-NRM [95] could reduce 33% ranking errors compared to VSM with TF-IDF weight. Using supervised IR model would require constructing new training dataset based on SIGIR eCom Data Challenge Dataset. For example, the new dataset could be formulated for a text sequence pair regression task, where a pair of product titles of the same category id path is given value of 1 and that of different category id path is given value of 0. And a more feasible way may be to fine-tune a pre-trained NLM like BERT for product similarity modelling instead of training from scratch. However, in terms of computational efficiency, we are still unsure if this would be better than just training or fine-tuning a SotA neural network model for product classification, since the new dataset for product similarity modelling would be much larger (e.g. $10\times$) than the original dataset. In addition, according to [6], because of high computational cost, local-interaction models like K-NRM [95] can only be used to re-rank a list of product titles (e.g. 100) retrieved with other fast IR models like BM25, while distributed models can do both retrieval and ranking. So, there is a trade-off between computational cost and ranking performance. For fast online prediction and relatively good ranking performance, we think using local-interaction model for re-ranking is not better than distributed model, as product titles are usually short and semantic match will be difficult if titles are first retrieved by a conventional IR model like BM25.

The second insight is that combining our word-level IR model based weighted KNN system with SotA character-level neural network(s), namely LSTM-BPV(s), can effectively boost the overall performance of LSTM-BPV(s) in terms of weighted-{P, R and F1}, which demonstrates the effectiveness of our KNN system. According to [26], another reason why the hybrid ensemble could be better than its constituent LSTM-BPV(s) is that the 2 approaches are accurate (i.e. error rate better than random guessing) and diverse (i.e. are different in a lot of aspects). Their main difference is: the KNN system captures word-level morphological information of a product title through exact match ("hard match"), while LSTM-BPV captures character-level global semantic information of a product title, similar to semantic match ("soft match").

The third insight is that fine-tuning a pre-trained BERT model is a straightforward approach to obtaining good results in this product classification task. The drawback is, however, the relatively large computational cost involved. This is partly because of the large size of BERT-large uncased model and small batch size used for fine-tuning. This is also because input of fixed sequence length is needed by the model and thus excessive padding has to be used. For better accuracy and faster training, like in [13], we could use adaptive input representations [4] and adaptive softmax [20]. Both the 2 methods exploit the Zipfian distribution of words to reduce memory footprint and computational cost, and thus enable faster training. (As a brief introduction, according to [80], Zipf's Law states that a term's collection frequency $TTF(w_i)$ is proportional to its inverse rank within the vocabulary ($TTF(w_i) \propto \frac{1}{i}$) (i.e. w_1 has the highest number of occurrences in the document collection (collection frequency)).) We could also use better pre-trained BERT model for fine-tuning, such as the recently developed RoBERTa [55].

The fourth insight is that even BERT-large uncased model's [15] general linguistic understanding is not enough. Although the model is large in size with 340 million parameters and has been pre-trained on 2 large text corpuses, it took 40 epochs to fine-tune

the model to reach performance comparable to the SotA. This suggests that the model's knowledge acquired from general-domain pre-training is still not enough for fast adaptation to the product taxonomy classification task. Hence, we need to improve the model's general language understanding. One possible direction is to pre-train or fine-tune the model on larger text corpus and to increase model size, but this might be only feasible for large IT companies (like in [74]) due to the relatively large computational cost involved. Another direction is to modify the model architecture for better language modelling. For example, in 2019, [13] proposed Transformer-XL to learn dependencies beyond a fixed-length context. Through re-using hidden states from the previous fixed-length segments, Transformer-XL [13] is able to capture longer-term dependencies than Transformer and RNN.

The fifth insight is that fine-tuning a trained model is usually a more time-saving way compared to training a model from scratch, although we still need to tune hyperparameters. But usually such tuning can be done more quickly than that during training from scratch. We could firstly set a small batch size (e.g. 32 or 64), a smaller learning rate (e.g. $10\times$ smaller than the learning rate used during model training) and a small number of epochs (e.g. 5). However, sometimes it is still difficult to fine-tune a trained model because of the possibility of catastrophic forgetting. We think that to prevent this from happening, a proper validation set should be used during fine-tuning.

6.3 Future Work

For future work, we are going to incorporate word associations and word positional information into our analysis. For example, we may use dependence models, such as CRTER (CRoss TERm) [106] and Context-sensitive Proximity Model [105], as similarity function in our weighted KNN algorithm. We may also use recently developed supervised neural IR models (e.g. [8]) as similarity measure in our weighted KNN algorithm. We

are also going to use other weighting schemes of KNN to improve our KNN system's classification performance.

Another possible future direction is to use ensemble strategy to improve classification performance. The reason is that, modern L2R IR models, such as the one used in Google Search, usually incorporate many features (e.g. PageRank score [68]) more than single IR model (e.g. BM25) used in our experiments. This means using ensemble of several diverse IR models as similarity function in our KNN algorithm can probably improve the classification performance. Also, as we have already obtained good results through combining our KNN system and LSTM-BPV(s), we may try using ensemble of our method and other SotA methods.

Furthermore, it is interesting to fine-tune a pre-trained NLM like BERT [15] for product similarity modelling task first and then fine-tune it for product classification. We are also interested in developing a neural IR model based on a pre-trained NLM, e.g. by fine-tuning the NLM for product classification first and then for learning to rank.

We also plan to evaluate our implemented systems and proposed methods on more datasets, including some real document collections (e.g. [36, 53, 54, 108, 49]), and to apply our implemented system in real-world applications (e.g. [103, 64, 100]).

Bibliography

[1] ABADI, M., BARHAM, P., CHEN, J., CHEN, Z., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., IRVING, G., ISARD, M., ET AL. Tensorflow: A System for Large-scale Machine Learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)* (2016), pp. 265–283.

[2] AMATI, G. *Probability Models for Information Retrieval Based on Divergence from Randomness*. PhD thesis, University of Glasgow, 2003.

[3] AMATI, G., AND VAN RIJSBERGEN, C. J. Term Frequency Normalization via Pareto Distributions. In *European Conference on Information Retrieval* (2002), Springer, pp. 183–192.

[4] BAEVSKI, A., AND AULI, M. Adaptive Input Representations for Neural Language Modeling. *arXiv preprint arXiv:1809.10853* (2018).

[5] BOJANOWSKI, P., GRAVE, E., JOULIN, A., AND MIKOLOV, T. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics 5* (2017), 135–146.

[6] BRENNER, E. P., ZHAO, J., KUTIYANAWALA, A., AND YAN, Z. End-to-End Neural Ranking for eCommerce Product Search. *Proceedings of SIGIR eCom 18* (2018).

[7] CHEN, J., AND WARREN, D. Cost-sensitive Learning for Large-scale Hierarchical Classification. In *Proceedings of the 22nd ACM International Conference on*

Conference on Information & Knowledge Management (2013), ACM, pp. 1351–1360.

[8] CHEN, Q., HU, Q., HUANG, J. X., AND HE, L. CA-RNN: Using Context-aligned Recurrent Neural Networks for Modeling Sentence Similarity. In *Thirty-Second AAAI Conference on Artificial Intelligence* (2018).

[9] CHEN, Q., HU, Q., HUANG, J. X., AND HE, L. CAN: Enhancing Sentence Similarity Modeling with Collaborative and Adversarial Network. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval* (2018), ACM, pp. 815–824.

[10] CLINCHANT, S., AND GAUSSIER, E. Information-based Models for Ad Hoc IR. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (2010), ACM, pp. 234–241.

[11] CRAWFORD, S. L., FUNG, R. M., APPELBAUM, L. A., AND TONG, R. M. Classification Trees for Information Retrieval. In *Machine Learning Proceedings 1991*. Elsevier, 1991, pp. 245–249.

[12] DAI, A. M., AND LE, Q. V. Semi-supervised Sequence Learning. In *Advances in Neural Information Processing Systems* (2015), pp. 3079–3087.

[13] DAI, Z., YANG, Z., YANG, Y., COHEN, W. W., CARBONELL, J., LE, Q. V., AND SALAKHUTDINOV, R. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. *arXiv preprint arXiv:1901.02860* (2019).

[14] DEERWESTER, S., DUMAIS, S. T., FURNAS, G. W., LANDAUER, T. K., AND HARSHMAN, R. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science 41*, 6 (1990), 391–407.

[15] DEVLIN, J., CHANG, M.-W., LEE, K., AND TOUTANOVA, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).

[16] DIETTERICH, T. G. Ensemble Methods in Machine Learning. In *International Workshop on Multiple Classifier Systems* (2000), Springer, pp. 1–15.

[17] ELISSEEFF, A., AND WESTON, J. A Kernel Method for Multi-labelled Classification. In *Advances in Neural Information Processing Systems* (2002), pp. 681–687.

[18] FENG, W., ZHANG, Q., HU, G., AND HUANG, J. X. Mining Network Data for Intrusion Detection through Combining SVMs with Ant Colony Networks. *Future Generation Comp. Syst. 37* (2014), 127–140.

[19] GOUMY, S., AND MEJRI, M.-A. Ecommerce Product Title Classification.

[20] GRAVE, E., JOULIN, A., CISSÉ, M., JÉGOU, H., ET AL. Efficient Softmax Approximation for GPUs. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 1302–1310.

[21] GRAVES, A., AND SCHMIDHUBER, J. Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. *Neural Networks 18*, 5-6 (2005), 602–610.

[22] GUPTA, V., KARNICK, H., BANSAL, A., AND JHALA, P. Product Classification in E-commerce Using Distributional Semantics. *arXiv preprint arXiv:1606.06083* (2016).

[23] HA, J.-W., PYO, H., AND KIM, J. Large-scale Item Categorization in E-commerce Using Multiple Recurrent Neural Networks. In *Proceedings of the*

22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2016), ACM, pp. 107–115.

[24] HAN, E.-H. S., KARYPIS, G., AND KUMAR, V. Text Categorization Using Weight Adjusted K-nearest Neighbor Classification. In *Pacific-asia Conference on Knowledge Discovery and Data Mining* (2001), Springer, pp. 53–65.

[25] HANCOCK-BEAULIEU, M., GATFORD, M., HUANG, X., ROBERTSON, S. E., WALKER, S., AND WILLIAMS, P. W. Okapi at TREC-5. In *Proceedings of The Fifth Text REtrieval Conference, TREC 1996, Gaithersburg, Maryland, USA, November 20-22, 1996* (1996).

[26] HANSEN, L. K., AND SALAMON, P. Neural Network Ensembles. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 10 (1990), 993–1001.

[27] HE, B., HUANG, J. X., AND ZHOU, X. Modeling Term Proximity for Probabilistic Information Retrieval Models. *Inf. Sci. 181*, 14 (2011), 3017–3031.

[28] HECHENBICHLER, K., AND SCHLIEP, K. Weighted K-nearest-neighbor Techniques and Ordinal Classification.

[29] HOCHREITER, S., AND SCHMIDHUBER, J. Long Short-term Memory. *Neural Computation 9*, 8 (1997), 1735–1780.

[30] HOWARD, J., AND RUDER, S. Universal Language Model Fine-tuning for Text Classification. *arXiv preprint arXiv:1801.06146* (2018).

[31] HU, H., ZHU, R., WANG, Y., FENG, W., TAN, X., AND HUANG, J. X. A Best Match KNN-based Approach for Large-scale Product Categorization.

[32] HUANG, P.-S., HE, X., GAO, J., DENG, L., ACERO, A., AND HECK, L. Learning Deep Structured Semantic Models for Web Search Using Clickthrough

Data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management* (2013), ACM, pp. 2333–2338.

[33] HUANG, X., AND HU, Q. A Bayesian Learning Approach to Promoting Diversity in Ranking for Biomedical Information Retrieval. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (2009), ACM, pp. 307–314.

[34] HUANG, X., HUANG, Y. R., WEN, M., AN, A., LIU, Y., AND POON, J. Applying Data Mining to Pseudo-Relevance Feedback for High Performance Text Retrieval. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China* (2006), pp. 295–306.

[35] HUANG, X., PENG, F., SCHUURMANS, D., CERCONE, N., AND ROBERTSON, S. E. Applying Machine Learning to Text Segmentation for Information Retrieval. *Information Retrieval 6*, 3-4 (2003), 333–362.

[36] HUANG, X., ZHONG, M., AND SI, L. York University at TREC 2005: Genomics Track. In *Proceedings of the Fourteenth Text REtrieval Conference, TREC 2005, Gaithersburg, Maryland, USA, November 15-18, 2005* (2005).

[37] JIA, Y., WANG, X., CAO, H., RU, B., AND YANG, T. An Empirical Study of Using An Ensemble Model in E-commerce Taxonomy Classification Challenge. In *The 2018 SIGIR Workshop On eCommerce (Accepted), Ann Arbor, MI* (2018).

[38] JOACHIMS, T. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *European Conference on Machine Learning* (1998), Springer, pp. 137–142.

[39] JOHNSON, R., AND ZHANG, T. Supervised and Semi-supervised Text Categorization Using LSTM for Region Embeddings. *arXiv preprint arXiv:1602.02373* (2016).

[40] JORDAN, M. Attractor Dynamics and Parallelism in a Connectionist Sequential Machine. In *Proc. of the Eighth Annual Conference of the Cognitive Science Society (Erlbaum, Hillsdale, NJ), 1986* (1986).

[41] JOULIN, A., GRAVE, E., BOJANOWSKI, P., AND MIKOLOV, T. Bag of Tricks for Efficient Text Classification. *arXiv preprint arXiv:1607.01759* (2016).

[42] KIM, Y. Convolutional Neural Networks for Sentence Classification. *arXiv preprint arXiv:1408.5882* (2014).

[43] KINGMA, D. P., AND BA, J. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* (2014).

[44] KOZAREVA, Z. Everyone Likes Shopping! Multi-class Product Categorization for E-commerce. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2015), pp. 1329–1333.

[45] LARKEY, L. S., AND CROFT, W. B. Combining Classifiers in Text Categorization. In *SIGIR* (1996), vol. 96, Citeseer, pp. 289–297.

[46] LE, Q., AND MIKOLOV, T. Distributed Representations of Sentences and Documents. In *International Conference on Machine Learning* (2014), pp. 1188–1196.

[47] LECUN, Y., BOTTOU, L., BENGIO, Y., HAFFNER, P., ET AL. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[48] LI, M. Y., TAN, L., KOK, S., AND SZYMANSKA, E. Unconstrained Product Categorization with Sequence-to-Sequence Models.

[49] LIANG, Z., ZHANG, G., HUANG, J. X., AND HU, Q. V. Deep Learning for Healthcare Decision Making with EMRs. In *2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)* (2014), IEEE, pp. 556–559.

[50] LIN, Y.-C., DAS, P., AND DATTA, A. Overview of the SIGIR 2018 eCom Rakuten Data Challenge.

[51] LIU, X., REN, F., AND YUAN, C. Use Relative Weight to Improve the KNN for Unbalanced Text Category. In *International Conference on Natural Language Processing & Knowledge Engineering* (2010).

[52] LIU, Y., AN, A., AND HUANG, X. Boosting Prediction Accuracy on Imbalanced Datasets with SVM Ensembles. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2006), Springer, pp. 107–118.

[53] LIU, Y., HUANG, X., AN, A., AND YU, X. ARSA: A Sentiment-aware Model for Predicting Sales Performance Using Blogs. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2007), ACM, pp. 607–614.

[54] LIU, Y., HUANG, X., AN, A., AND YU, X. Modeling and Predicting the Helpfulness of Online Reviews. In *2008 Eighth IEEE International Conference on Data Mining* (2008), IEEE, pp. 443–452.

[55] LIU, Y., OTT, M., GOYAL, N., DU, J., JOSHI, M., CHEN, D., LEVY, O., LEWIS, M., ZETTLEMOYER, L., AND STOYANOV, V. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692* (2019).

[56] LIU, Y., YU, X., HUANG, J. X., AND AN, A. Combining Integrated Sampling with SVM Ensembles for Learning from Imbalanced Datasets. *Inf. Process. Manage. 47*, 4 (2011), 617–631.

[57] LOSHCHILOV, I., AND HUTTER, F. SGDR: Stochastic Gradient Descent with Warm Restarts. *arXiv preprint arXiv:1608.03983* (2016).

[58] MARON, M. E. Automatic Indexing: an Experimental Inquiry. *Journal of the ACM (JACM) 8*, 3 (1961), 404–417.

[59] MASAND, B., LINOFF, G., AND WALTZ, D. Classifying News Stories Using Memory Based Reasoning. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1992), ACM, pp. 59–65.

[60] MASTERS, D., AND LUSCHI, C. Revisiting Small Batch Training for Deep Neural Networks. *arXiv preprint arXiv:1804.07612* (2018).

[61] MCCANN, B., BRADBURY, J., XIONG, C., AND SOCHER, R. Learned in Translation: Contextualized Word Vectors. In *Advances in Neural Information Processing Systems* (2017), pp. 6294–6305.

[62] MERITY, S., KESKAR, N. S., AND SOCHER, R. Regularizing and Optimizing LSTM Language Models. *arXiv preprint arXiv:1708.02182* (2017).

[63] MERITY, S., XIONG, C., BRADBURY, J., AND SOCHER, R. Pointer Sentinel Mixture Models. *arXiv preprint arXiv:1609.07843* (2016).

[64] MIAO, J., HUANG, J. X., AND YE, Z. Proximity-based Rocchio's Model for Pseudo Relevance Feedback. In *Proceedings of the 35th International ACM SIGIR*

Conference on Research and Development in Information Retrieval (2012), ACM, pp. 535–544.

[65] MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G. S., AND DEAN, J. Distributed Representations of Words and Phrases and Their Compositionality. In *Advances in Neural Information Processing Systems* (2013), pp. 3111–3119.

[66] MITRA, B., DIAZ, F., AND CRASWELL, N. Learning to Match Using Local and Distributed Representations of Text for Web Search. In *Proceedings of the 26th International Conference on World Wide Web* (2017), International World Wide Web Conferences Steering Committee, pp. 1291–1299.

[67] MORIN, F., AND BENGIO, Y. Hierarchical Probabilistic Neural Network Language Model. In *Aistats* (2005), vol. 5, Citeseer, pp. 246–252.

[68] PAGE, L., BRIN, S., MOTWANI, R., AND WINOGRAD, T. The PageRank Citation Ranking: Bringing Order to the Web. Tech. rep., Stanford InfoLab, 1999.

[69] PASZKE, A., GROSS, S., CHINTALA, S., CHANAN, G., YANG, E., DEVITO, Z., LIN, Z., DESMAISON, A., ANTIGA, L., AND LERER, A. Automatic Differentiation in Pytorch.

[70] PENNINGTON, J., SOCHER, R., AND MANNING, C. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014), pp. 1532–1543.

[71] PETERS, M. E., NEUMANN, M., IYYER, M., GARDNER, M., CLARK, C., LEE, K., AND ZETTLEMOYER, L. Deep Contextualized Word Representations. *arXiv preprint arXiv:1802.05365* (2018).

[72] PONTE, J. M., AND CROFT, W. B. A Language Modeling Approach to Information Retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1998), ACM, pp. 275–281.

[73] RADFORD, A., NARASIMHAN, K., SALIMANS, T., AND SUTSKEVER, I. Improving Language Understanding by Generative Pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf* (2018).

[74] RADFORD, A., WU, J., CHILD, R., LUAN, D., AMODEI, D., AND SUTSKEVER, I. Language Models are Unsupervised Multitask Learners. Tech. rep., Technical Report, OpenAi, 2018.

[75] ŘEHŮŘEK, R., AND SOJKA, P. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks* (Valletta, Malta, May 2010), ELRA, pp. 45–50. http://is.muni.cz/publication/884893/en.

[76] ROBERTSON, S. E., AND JONES, K. S. Relevance Weighting of Search Terms. *Journal of the American Society for Information Science 27*, 3 (1976), 129–146.

[77] ROBERTSON, S. E., WALKER, S., BEAULIEU, M., GATFORD, M., AND PAYNE, A. Okapi at TREC-4. In *Proceedings of the Fourth Text Retrieval Conference* (1996), vol. 500, NIST Special Publication Gaithersburg, MD, pp. 73–97.

[78] ROBERTSON, S. E., WALKER, S., JONES, S., HANCOCK-BEAULIEU, M. M., GATFORD, M., ET AL. Okapi at TREC-3. *NIST Special Publication Sp 109* (1995), 109.

[79] SALTON, G., WONG, A., AND YANG, C.-S. A Vector Space Model for Automatic Indexing. *Communications of the ACM 18*, 11 (1975), 613–620.

[80] SCHÜTZE, H., MANNING, C. D., AND RAGHAVAN, P. *Introduction to Information Retrieval*, vol. 39. Cambridge University Press, 2008.

[81] SEVERYN, A., AND MOSCHITTI, A. Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2015), ACM, pp. 373–382.

[82] SHEN, D., RUVINI, J.-D., AND SARWAR, B. Large-scale Item Categorization for E-Commerce. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management* (New York, NY, USA, 2012), CIKM '12, ACM, pp. 595–604.

[83] SHEN, Y., HE, X., GAO, J., DENG, L., AND MESNIL, G. A Latent Semantic Model with Convolutional-pooling Structure for Information Retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management* (2014), ACM, pp. 101–110.

[84] SKINNER, M. Product Categorization with LSTMs and Balanced Pooling Views. In *SIGIR 2018 Workshop on eCommerce (ECOM 18)* (2018).

[85] SMITH, L. N., AND TOPIN, N. Super-convergence: Very Fast Training of Neural Networks Using Large Learning Rates. *arXiv preprint arXiv:1708.07120* (2017).

[86] STEVENS, K., EDEN, M., POLLACK, I., FICKS, L., ET AL. Nearest Neighbor Pattern Classification.

[87] SUN, A. Short Text Classification Using Very Few Words. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2012), ACM, pp. 1145–1146.

[88] SUTSKEVER, I., MARTENS, J., DAHL, G. E., AND HINTON, G. E. On the Importance of Initialization and Momentum in Deep Learning. *ICML (3) 28*, 1139-1147 (2013), 5.

[89] SUZUKI, S. D., ISEKI, Y., SHIINO, H., ZHANG, H., IWAMOTO, A., AND TAKAHASHI, F. Convolutional Neural Network and Bidirectional LSTM Based Taxonomy Classification Using External Dataset at SIGIR eCom Data Challenge.

[90] TAN, S. Neighbor-weighted K-nearest Neighbor for Unbalanced Text Corpus. *Expert Systems with Applications 28*, 4 (2005), 667–671.

[91] VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, Ł., AND POLOSUKHIN, I. Attention is All You Need. In *Advances in Neural Information Processing Systems* (2017), pp. 5998–6008.

[92] WERMTER, S., AREVIAN, G., AND PANCHEV, C. Recurrent Neural Network Learning for Text Routing.

[93] WU, Y., SCHUSTER, M., CHEN, Z., LE, Q. V., NOROUZI, M., MACHEREY, W., KRIKUN, M., CAO, Y., GAO, Q., MACHEREY, K., ET AL. Google's Neural Machine Translation System: Bridging the Gap Between Human and Machine Translation. *arXiv preprint arXiv:1609.08144* (2016).

[94] XIA, R., ZONG, C., AND LI, S. Ensemble of Feature Sets and Classification Algorithms for Sentiment Classification. *Information Sciences 181*, 6 (2011), 1138–1152.

[95] XIONG, C., DAI, Z., CALLAN, J., LIU, Z., AND POWER, R. End-to-end Neural Ad-hoc Ranking with Kernel Pooling. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2017), ACM, pp. 55–64.

[96] YANG, Y. Expert Network: Effective and Efficient Learning from Human Decisions in Text Categorization and Retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1994), Springer-Verlag New York, Inc., pp. 13–22.

[97] YANG, Y., AND CHUTE, C. G. An Example-Based Mapping Method for Text Categorization and Retrieval. *ACM Transactions on Information Systems 12*, 3 (1994), 252–277.

[98] YANG, Y., AND LIU, X. A Re-examination of Text Categorization Methods. In *International ACM SIGIR Conference on Research and Development in Information Retrieval* (1999).

[99] YANG, Z., DAI, Z., YANG, Y., CARBONELL, J., SALAKHUTDINOV, R., AND LE, Q. V. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv preprint arXiv:1906.08237* (2019).

[100] YIN, X., HUANG, J. X., LI, Z., AND ZHOU, X. A Survival Modeling Approach to Biomedical Search Result Diversification Using Wikipedia. *IEEE Transactions on Knowledge and Data Engineering 25*, 6 (2012), 1201–1212.

[101] YOGATAMA, D., D'AUTUME, C. D. M., CONNOR, J., KOCISKY, T., CHRZANOWSKI, M., KONG, L., LAZARIDOU, A., LING, W., YU, L., DYER, C., ET AL. Learning and Evaluating General Linguistic Intelligence. *arXiv preprint arXiv:1901.11373* (2019).

[102] YU, W., SUN, Z., LIU, H., LI, Z., AND ZHENG, Z. Multi-level Deep Learning based E-commerce Product Categorization.

[103] YU, X., LIU, Y., HUANG, X., AND AN, A. Mining Online Reviews for Predicting Sales Performance: A Case Study in the Movie Domain. *IEEE Transactions on Knowledge and Data Engineering 24*, 4 (2010), 720–734.

[104] ZHAI, C., AND LAFFERTY, J. A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (2001), ACM, pp. 334–342.

[105] ZHAO, J., AND HUANG, J. X. An Enhanced Context-sensitive Proximity Model for Probabilistic Information Retrieval. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval* (2014), ACM, pp. 1131–1134.

[106] ZHAO, J., HUANG, J. X., AND HE, B. CRTER: Using Cross Terms to Enhance Probabilistic Information Retrieval. In *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25-29, 2011* (2011), pp. 155–164.

[107] ZHAO, J., HUANG, J. X., AND ZHENG, Y. Modeling Term Associations for Probabilistic Information Retrieval. *ACM Transactions on Information Systems 32*, 2 (2014), 1–47.

[108] ZHU, J., HUANG, X., SONG, D., AND RÜGER, S. Integrating Multiple Document Features in Language Models for Expert Finding. *Knowledge and Information Systems 23*, 1 (2010), 29–54.

Appendix A

A Part of the Training Dataset

Replacement Viewsonic VG710 LCD Monitor 48Watt AC Adapter 12V 4A 3292>114>1231

HP COMPAQ Pavilion DV6-1410EZ 4400mAh 48Wh 6 Cell Li-ion 10.8V Black Compatible Battery 3292>1370>4767>3975>1420

Bonjour 2296>3597>2989

Two Pack 6V 12Ah Eaton POWERRITE PRO II 2400 6V 12Ah UPS Replacement Battery - SPS BRAND 3292>114>1231

Generations Small Side Table White 4015>3636>1319>1409>3606

Mont Blanc Mb Starwalker Men Eau De Toilette Edt 2.5Oz / 70Ml 3625>4399>1598>3903

4-Pack Replacement Engine Air Filter for 2009 Sterling Truck Bullet 55 L6 6.7 Car/Automotive 4015>2337>1458>40

GREEN TRI-SHIELD SOFT SKIN HARD CASE COVER KICKSTAND FOR SAMSUNG GALAXY NOTE 5 3292>3581>3145>2201

50 Pcs White Universal Car Door Plastic Push in Fastener Rivets 8.5mm Hole Dia 2199>915>4085>3205

Luscious Pink Perfume 3.4 oz Eau De Parfum Spray For Women By MARIAH CAREY 3625>3005

21 Kenmore 11625614501 Vacuum Bags & 4 HEPA Filter - 5055

Bags & 86889, EF-1 Filter 4015>2337>2943>4735>3582>1998

SURFACE SHIELDS DA106150 Door Protection, 6 In. x 150 Ft., Clear 4238>1625>3571>1318

"1’10"" x 2’10"" Cork Wool Rug – Athena Hand Tufted Rug with Black Border" 4015>3636>526>3639

Bedroom Polyester Knitted Handmade Decoration Braid Handicraft Chinese Knot 4015>2824>2205>3477

Sunnydaze Rocking Wave Lounger w/ Pillow, Green 4015>3636>1319>2055

90210 Men’s West Beverly Hills High T-shirt Large Pink 1608>4269>1667>4910

Circuit Breaker Standard BR-330 2199>4592>12

6-Pack 10 MFD 370 Volt Oval Run Capacitor Replacement for Carrier 579EEW060100B 4015>2337>1458>40

Hot Wheels Masters Of The Universe 1:64 Scale Diecast Car: ’57 Buick 1395>2736>1061>3871

Ka-Bar Desert MULE Serrated Folding Knife 4238>321>753>3121

Gucci Womens Eyeglasses 3517 WW2/14 Plastic Rectangle Black Crystal Frames 1608>2227>574>2226

5.11 TACTICAL 74280 Taclite TDU Pants, R/M, Dark Navy 4015>3285>1443>20

Westin 72-41111 Wade; Truck Bed Side Rail Protector 2199>4592>12

"Gelco Multi-Flue 3/4"" Mesh Cap with 4"" Overhang – 16"" x 37"" x 8""" 4015>3636>502>191

M4x8mm 304 Stainless Steel Button Head Hex Socket Tamper Proof Screws 30pcs 4015>3754>3580>4753>4146

Bilstein 24-181488 Shock Absorber 46mm Monotube Shock Absorber; 4600 Series; 2199>4592>12

Margaritaville Salt Rimmer and Lime Set 4015>3271>2768>4244

"South Seas 1.5"" Cabinet Knob - Finish: Vibra Pewter" 4015>3754>3663>512>3161

Discount Starter and Alternator 6651N Jaguar S-Type Replacement Starter 2199>661>333>3609

3M 16016 PPS Adapter 9 2199>4592>12

4-Pack 45/5 MFD 370 Volt Dual Round Run Capacitor Replacement for Carrier 583BNW036090AB 4015>2337>1458>40

BODY GLOVE Tweedle Series Red Nylon Vest L/XL 16289-RED-LX 2199>4592>12

Lauren Keiser Music Publishing 3 Tangos for Flute, Harp and Strings LKM Music Series Composed by Lalo Schifrin 4015>2824>2964>1002>200

CHAT 60 U 3292>3581>1878>4304

SimStars Reflections Snake Bead 4015>2824>2205>1315

Lionel Richie - Renaissance 2296>3597>689

2Gb (1X2Gb) Memory Ram Compatible With Dell Vostro 320 (All-In-One) By CMS (A91) 3292>1370>3233>332

DecalGirl LS-BLDRNG DecalGirl Laptop Skin - Blood Ring 92

0.8mmx8mmx60mm 304 Stainless Steel Tension Springs Silver

Tone 5pcs 2199>661>646>311

DecalGirl LGSN-BASEBALL LG Shine Skin - Baseball 92

Woodstock Mayonaise 32 Oz -Pack of 12 1208>546>4262>2775

Sedona - Gray 2'x8' 4015>3636>526>3639

IT'S NOT THE EAT IT'S THE HUMIDITY [Vinyl Record] 2296>3597>208

Full Body Harness, Miller By Honeywell, AC-TB2/3XLBL 4015>3285>4803

Autograph Warehouse 97078 Ron Bass Football Card South Carolina 1991 Collegiate Collection No. 117 4015>2824>3210>4573

Shabby Chic by Patty Tuggle Canvas Wall Art 14 x 19 4015>3636>526>2454>589

Epson EB-C2040XN Projector Lamp (Original Philips/Osram Bulb Inside) with Housing 3292>290>497

Women's Black Cat Mini Dress Costume S 1608>1150>1244>615

Smiling Einstein Icon Kids' Premium T-Shirt by Spreadshirt ™ 1608>4269>4411>4306

New Laptop Battery Acer Aspire one 532H-2727 532H-2730 532 H-2742 532H-2789 532H-2806 2600mah 3 Cell 3292>1370>4767>3975>1420

Penthouse Women's Audrey Peep Toe Pump,Leopard,7 M US 2199>4592>12

Kaboom! Family Day: The Berenstain Be 2296>3706>3834

5x Replacement Critikon 7300 Battery - 12V, 7Ah, Sealed Lead Acid, SLA 3292>114>2641>3360

Youth Screw Lab Safety I Want Superpowers Nerdy Science T shirt for Kids 1608>4269>4411>4306

Invasion Of The Boobie Sn 2296>3706>3231

East West Furniture MLT-MAH-T Milan Rectangular Dining Table 4015>3636>1319>1409>3606

Unique Bargains Unique Bargains Golden Tone Metal Soldering Iron Holder w Black Rectangle Base 4015>3754>3663>512>4921

Mightyskins Protective Vinyl Skin Decal Cover for Nintendo 2DS wrap sticker skins Spaced Out 3292>3581>3145>2201

YG-1 TOOL COMPANY 93103 Solid Carb End Mill, Sq, 1/8inDiax 2-1/4Lin 4015>3754>3663>512>319

"Krator 5"" Chrome LED Headlight w/ Light Mounting Bracket for Harley Davidson Softail Cross Bones Deuce Rocker ..." 2199>4592>12

Jaw Puller, Locking, Westward, 23MD27 4015>3754>3663>1500>1717

"Reaudio Re Audio 10"" Rex Series Woofer 200W Rms Svc 4 Ohm 12.000000In. X 7.000000In. X 12.000000In." 3625>4399>1598>3903

Heavy Duty 3 Layers Silver Tarp 10ft x 16ft 4015>3754>3663>512>4157>2157

Standard DS838 Door Jamb Switch Standard Motor Products Door Jamb Switch [Misc.] 2199>4592>12

Savage Seamless Background Paper 107 x 12 yd Gulf Blue 3292>1041>3198>1109

Appendix B

The Instructions of the KNN Classification System Based on Lucene

To run experiments with our KNN system based on Lucene, you could do the following steps:

Step 1: download the zip file of the whole repository of the project "LuceneTaxonomyClassification" from this link[9]. Unzip the zip file.

Step 2: install JAVA 8 (if not installed) and an Integrated Development Environment (IDE) that can work with Maven, such as NetBeans IDE 8.2 (if not installed).

Step 3: open IDE and open the project.

Step 4: change the parameters of the source file "./src/main/java/com/mycompany/lucenedocumentvectorconverting/luceneindexing.java". Detailed instructions of parameter setting (e.g. number of threads) has been included in the source file.

Step 5: run the project. After running the program, concatenate the prediction files in the specified folder into a single tsv file for evaluation (you can use bash commands (e.g. "cat") to do this).

Step 6: move the tsv file into the folder "./dataChallengeEvaluationScript" for evaluation. The evaluation script is "./dataChallengeEvaluationScript/eval.py" (borrowed from this repository[10]) and its instructions are also included in the repository.

---

[9] https://drive.google.com/drive/folders/1cxJHT1k2NPKWizHgCWEP2tmbExoRjwc1?usp=sharing

[10] https://github.com/sigir-ecom/dataChallenge

Appendix C

The Instructions of the KNN Classification System Based on Gensim Doc2vec API

To run experiments with our KNN system based on Gensim Doc2vec, you could do the following steps:

Step 1: download the zip file of the whole repository of the project (" Doc2vec_product_classification") from this link[11]. Unzip the zip file.

Step 2: install Python 3 (if not installed) and Gensim (if not installed).

Step 3: change the parameters of the source file "./rdc_taxonomy_classification_remote_Doc2vec_combined_multithread_python3-morepreprocessing_min_1.py". The instructions of parameter setting (e.g. number of threads) has been included in the source file.

For reference, hyperparameters for training document embedding with Gensim's [75] Doc2vec API are listed as follows:

1. dm: when set to 1: use PV-DM to train the Doc2vec model; when set to 0: use PV-DBOW to train the Doc2vec model;

2. $vector\_size$: dimension of document embedding trained: $\mathbf{D_j} \in \mathbb{R}^{vector\_size}$;

3. $dbow\_words$: when set to 1:train word embeddings with Skipgram simultaneously with PV-DBOW training; when set to 0: only train PV-DBOW;

[11] https://drive.google.com/drive/folders/18RzqPwd6PS0tad6UI4EI-60zsR_SrkhT?usp=sharing

4. dm_concat: when set to 1: use concatenation of context vectors instead of sum or average; when set to 0: use sum/average of context vectors;

5. dm_tag_count: number of document tags per document in dm_concat mode;

6. $window$: the size of context window, i.e. c;

7. min_count: a word w_i whose total term frequency $TTF(w_i, C) = \sum_{k=1}^{N} TF(w_i, D_k) < min\_count$ is removed from the corpus;

8. $epochs$: the number of iterations over the whole corpus C;

9. hs: when set to 1: use hierarchical softmax to train the Doc2vec model; when set to 0: use negative sampling to train the Doc2vec model;

Step 4: open a bash shell and run the project by typing "python3 ./rdc_taxonomy_classification_remote_Doc2vec_combined_multithread_python3-morepreprocessing_min_1.py". After running the program, concatenate the prediction files in the specified folder into a single tsv file for evaluation (you can use bash commands (e.g. "cat") to do this).

Step 5: use the same evaluation script as in Appendix B for evaluation.

Appendix D

The Instructions of the Hybrid System Based on LSTM-BPVs and Weighted KNN

To run experiments with the ensembles of our KNN system and LSTM-BPV(s), you could do the following steps:

Step 1: download the whole repository of the project ("ecom-rakuten") from this Github repository[12]. Unzip the zip file. Also, download "kerosene" repository from this link[13], and move subfolder "./kerosene" into "ecom-rakuten" folder.

Step 2: install Python 3 (if not installed). After that, change directory into "ecom-rakuten" folder and install the dependencies listed in "./requirements.txt" via commands like "pip install -r ./requirements.txt" (if not installed).

Step 3: open a bash shell and use bash commands to run the program. The instructions has been included in the Github repository. Also, when you want to evaluate the trained models on validation/test sets , λ can be set using a flag, e.g. "–i=0.6".

As for evaluation, the evaluation script has already been included in the program so that the performance can be seen after program execution.

[12] https://github.com/haohao-hu/ecom-rakuten

[13] https://github.com/mcskinner/kerosene/tree/66bc8b12178c7826cc5f5a09a3e7886df0b2d8a2

Appendix E

The Instructions of the Classification System Based on BERT Model

To run experiments with the BERT classifier, you could do the following steps:

Step 1: download the whole repository of the project ("bert") from this Github repository[14]. Unzip the zip file. You can follow the instructions shown in "README.md" in the folder to download the BERT-large-uncased model. Change directory into "bert" folder.

Step 2: install Python 2 (if not installed) and TensorFlow 1.12 [1] (if not installed).

Step 3: if you have not downloaded SIGIR eCom Data Challenge Dataset yet, you can download them via this link[15]. Then, put "rdc-catalog-gold.tsv" in the folder "./rdc_dataset".

Step 4: set the training parameters in "./run_rdc_clf.sh". The instructions has been included in "README.md" in the Github repository. The instructions for setting hyperparameters of "run_classifier.py" are in the file itself.

Step 5: open a bash shell and use bash commands "bash ./run_rdc_clf.sh" to run the program.

As for evaluation, the evaluation script has already been included in the program so that the performance can be seen after program execution.

---

[14] https://github.com/haohao-hu/bert/tree/master

[15] https://docs.google.com/forms/d/e/1FAIpQLSfwcBlO_Z_aEXSbohScyj1YRRdsPULagEn28YYtca2ZkY5Ocw/viewform

Appendix F

Proof regarding the hybrid approach

We would like to find a solution for this inequality:

$$
\begin{gathered}
P_{ensemble}(t \in dc_m) > \max_{i \in \{1,\ldots,3008\}} P_{LSTM\text{-}BPVs}(t \in dc_i) \\
\Updownarrow \\
(1-\lambda) P_{LSTM\text{-}BPVs}(t \in dc_m) + \lambda(\max_{i \in \{1,\ldots,3008\}} P_{LSTM\text{-}BPVs}(t \in dc_i) + 0.1) > \\
\max_{i \in \{1,\ldots,3008\}} P_{LSTM\text{-}BPVs}(t \in dc_i) \\
\Updownarrow \\
0.1\frac{\lambda}{1-\lambda} > \max_{i \in \{1,\ldots,3008\}} P_{LSTM\text{-}BPVs}(t \in dc_i) - P_{LSTM\text{-}BPVs}(t \in dc_m)
\end{gathered}
\quad , \quad \text{(F.1)}
$$

Since probability is always within the range $[0, 1]$:

$$
\max_{i \in \{1,\ldots,3008\}} P_{LSTM\text{-}BPVs}(t \in dc_i) - P_{LSTM\text{-}BPVs}(t \in dc_m) \leq 1, \quad \text{(F.2)}
$$

Here, since t is a variable, to make it guaranteed that the inequality holds, we try solving this instead:

$$
\begin{gathered}
0.1\frac{\lambda}{1-\lambda} > 1 \\
\Updownarrow \\
\lambda > \frac{10}{11} \approx 0.9091
\end{gathered}
\quad , \quad \text{(F.3)}
$$