applied sciences

MDPI

Article

A Two-Phase Deep Learning-Based Recommender System: Enhanced by a Data Quality Inspector

William Lemus Leiva, Meng-Lin Li and Chieh-Yuan Tsai *

Department of Industrial Engineering and Management, Yuan-Ze University, Taoyuan City 320, Taiwan; wlemusleiva@gmail.com (W.L.L.); ben8340313@gmail.com (M.-L.L.)
* Correspondence: cytsai@saturn.yzu.edu.tw; Tel.: +886-3-463-8907

Featured Application: Enhances the performance, specifically the accuracy, of a collaborative filtering-based recommender system, by exploiting textual data and filtering the initial input.

Abstract: Research regarding collaborative filtering recommenders has grown fast lately. However, little attention has been paid to discuss how the input data quality impacts the result. Indeed, some review-rating pairs that a user gave to an item are inconsistent and express a different opinion, making the recommendation result biased. To solve the above drawback, this study proposes a two-phase deep learning-based recommender system. Firstly, a sentiment predictor of textual reviews is created, serving as the quality inspector that cleans and improves the input for a recommender. To build accurate predictors, this phase tries and compares a set of deep learning-based algorithms. Secondly, besides only exploiting the consistent review-rating pairs generated by the quality inspector, this phase builds deep learning-based recommender engines. The experiments on a real-world dataset showed the proposed data quality inspector, based on textual reviews, improves the overall performance of recommenders. On average, applying deep learning-based quality inspectors result in an above 6% improvement in RMSE, and more than a 2% boost in F1 score, and accuracy. This is robust evidence to prove the importance of the input data cleaning process in this field. Moreover, empirical evidence indicates the deep learning approach is suitable for modeling the sentiment predictor, and the core recommendation process, clearly outperforming the traditional machine learning methods.

Keywords: data quality; recommendation system; collaborative filtering; deep learning; text classification

Citation: Lemus Leiva, W.; Li, M.-L.; Tsai, C.-Y. A Two-Phase Deep Learning-Based Recommender System: Enhanced by a Data Quality Inspector. *Appl. Sci.* **2021**, *11*, 9667. https://doi.org/10.3390/app11209667

Academic Editor: Elisa Quintarelli

Received: 2 September 2021
Accepted: 13 October 2021
Published: 16 October 2021

1. Introduction

Big data allows online vendors to offer immeasurable options of items and services on their web pages. Normally, users feel overwhelmed by having more information than needed to make a simple consumption or purchase decision and commonly experience the information overload problem [1]. As a result, research regarding recommender systems has rapidly grown in the past years, and many e-commerce industries, such as music, movie, and tourism, have massively applied them. As personalization tools, the recommender systems automatically adapt to a particular user owing to prior monitoring of their interactions and needs, thus improving human-website interactions in the era of information overload [2,3]. Typically, the three most popular approaches for recommender implementation are: (a) content-based filtering, (b) collaborative filtering, and (c) hybrid approach. Among them, collaborative filtering is the most popular since it does not need to analyze the content of the items. Instead, it relies on the relationship between users and items, which is typically encoded in a numeric rating matrix.

Currently, the latest investigation related to recommenders focuses on (a) the use of a collaborative filtering strategy, or a hybrid recommender with collaborative filtering as the main base, (b) the implementation of a deep learning technique to extract latent features,

(c) the utilization of textual reviews, mouse clicks, triggered events, or eye-tracking as auxiliary information for personal interest verification, and (d) the recommendation accuracy as the main challenge to enhance [1–14].

The up-to-date direction of researchers in a collaborative filtering strategy is to deal with the recommendation accuracy by focusing on problems correlated to it, such as sparsity or cold-start [1–14]. Less attention has been paid to an essential problem that is highly correlated to accuracy: the quality of the ratings as input data. The authors of [15] use the votes of readers that denote if a review was helpful or not to compute a quality score. These quality scores are taken as weights assigned to the ratings in a probabilistic matrix factorization for performing the rating prediction. The authors of [16] investigated three ways of combining overall textual opinions with real ratings in a biased matrix factorization model. The results showed that the opinion post-filtering method, compared to the other two, and a standard algorithm based only on ratings, gives the best RMSE. The authors of [17] proposed a user-based collaborative filtering approach that considers explicit ratings and sentiment analysis extracted from users' reviews. Results from the collaborative filtering approach and sentiment analysis are combined to generate integrated ratings and create a list of recommendations. Although these researchers tried to improve the quality of the input, they only tuned the initial data, using as input a modified but complete version of the original ratings. The task where the input data is truly filtered or cleaned based on its quality was not conducted.

Commonly, e-commerce product platforms allow users to give feedback to items through (a) a numeric rating and (b) a textual review. The main problem is that there are cases where these two are inconsistent or in conflict [16], as the rating from a user reflects a different opinion of his or her written review for the same item. These rating-review pairs can be categorized as low-quality or noisy input data, and although this issue has not been widely mentioned in the literature, plenty of these cases can be found on the web. For example, a rating-review pair <1, "This movie is excellent"> might be conceptually inconsistent since this user indicates low satisfaction with the item through a poor rating "1", but he or she also shows high satisfaction with the same item through his or her textual opinion "This movie is excellent". Similarly, a rating-review record <5, "I think it was a little overhyped. To me, it wasn't as good as everyone raves about."> presents another contradictory case.

Unsurprisingly, if the raw ratings are the unique input for a product or service recommender, the low-quality ratings would also be entered, increasing the probability to generate biased results, bad recommendations, bad experiences for users, and a decrease in revenue of e-companies. Therefore, using textual reviews to inspect the quality of numeric ratings, separating the high-quality data from the noise, is a possible scenario worth to be analyzed. To fulfill the need, this paper proposes a two-phase deep learning-based approach: (1) use of textual reviews to create a sentiment predictor that serves as a raw data quality inspector, preprocessing and cleaning the input data for a (2) collaborative filtering recommender system, which consequently, boost its recommendation accuracy.

The main goal of our research is to create an input data quality inspector for the overall performance enhancement of collaborative filtering engines. The proposed inspector should be capable of differentiating between high-quality and noisy rating-review records well. The realization of this goal has a significant impact since current research has given little attention to discuss the effect of the rating quality as the input data for recommenders. Moreover, the inspector described in this article can be directly and automatically applied in collaborative filtering recommenders to boost their performance. This research allows perceiving the importance of the data quality inspection process and data cleaning in this field, guiding future research on directions to reduce bias.

The contributions of this study are the following:

(1) Develop a sentiment predictor of textual reviews that acts as a quality inspector of the ratings.

(2) Determine if a quality inspector based on textual comments truly enhances the performance of a recommender system.
(3) Apply state-of-the-art deep learning algorithms for constructing effective sentiment predictors and collaborative filtering recommenders.

The remainder of this paper is organized as follows. Section 2 reviews the relevant research. Section 3 introduces the proposed two-phase deep learning-based recommender system: enhanced by the data quality inspector. Section 4 describes the evaluation process to show the performance of the proposed system. Section 5 discusses the obtained results and mentions future work suggestions. Finally, Section 6 presents the conclusions.

2. Literature Review

Undoubtedly, almost all the challenges of a collaborative filtering approach are highly correlated with the accuracy of the recommender systems. Therefore, accuracy is the most discussed issue, and as it directly influences user satisfaction, it has been the top desired characteristic to enhance [1]. As a result, a present approach is to utilize deep learning algorithms to model the user-item preference matrix by extracting latent factors [14]. On the other hand, hybrid recommenders with a collaborative filtering strategy as a base, are often combined with another strategy or algorithm to minimize its limitations by exploiting the benefits of the other [1]. Nonetheless, the up-to-date scenario is to use deep learning for extracting hidden features from the content or auxiliary information and integrating them as inputs into the recommendation process [13,14].

2.1. Deep Learning-Based Collaborative Filtering

The following are some of the recent few studies, with the most impact, done for the core recommendation process with deep learning. The authors of [18] proposed AutoRec, which uses Autoencoders as a predictor for missing ratings. Their experimental results showed their model outperforms Matrix Factorization and Restricted Boltzmann Machine (RBM)-based collaborative filtering regarding accuracy on MovieLens and Netflix datasets. The authors of [19] applied to user-item preferences the Neural Autoregressive Distribution Estimation (NADE) algorithm. Compared to RBM, AutoRec, and several Matrix Factorization-based approaches, NADE demonstrates to provide more accurate recommendations and to be more efficiently optimizable. The authors of [20] also utilized NADE by modeling ratings but across both all users and items simultaneously. Their algorithm, CF-UserItem-CoAutoregressive, demonstrates to be a very successful model that achieves higher accuracy than the original version. Recently, the authors of [21] utilized a hybrid algorithm, composed of a Generalized Matrix Factorization component, which has some tweaks from a classic Matrix Factorization, and a Multi-Layer Perceptron module, to non-linearly model the interactions between users and items through the usage of implicit information. This is different from the other works which use the typical ratings.

2.2. Deep Learning-Based Sentiment Analysis

Mining sentiments from natural language is challenging because it requires a deep understanding of the explicit and implicit, regular and irregular, and syntactical and semantic language rules [22]. Consequently, deep learning-based sentiment analysis has lately received a lot of attention and can be divided into two steps: the first is text preprocessing which was demonstrated to directly influence the classification accuracy, and the second is the sentiment classifier which is the main core of sentiment analysis [23,24].

Related to text preprocessing works, the authors of [25] presented the design of a Convolutional Neural Network built on top of a Word2vec to allow for the use of task-specific and static vectors. His results add to well-established evidence the use of unsupervised pre-trained word vectors is an important ingredient for Natural Language Preprocessing. Similarly, the authors of [26] proposed a new model called SO-SD to build a classifier of Weibo messages using Word2vec. Likewise, the authors of [27] proposed a deep Convolutional Neural Network that easily exploits the richness of a word embedding to perform a comment's sentiment analysis. They showed that using only a word embedding as input features is enough to achieve state-of-the-art results.

Regarding the last advances in sentiment classifier algorithms, the authors of [28] performed sentiment analysis in short text answers with the use of a word embedding followed by two deep textual views: a CNN and a DSCNN, which is constructed with both a CNN layer and an LSTM layer. Their empirical evaluations proved that this method significantly improves predictive performance. Nevertheless, a few years ago, the authors of [29] introduced the attention concept to endow their translation method with the ability to give higher weight to more important words while taking care of the sequence structure of a text. Correspondingly, the authors of [30] implemented a hierarchical attention network for document classification, which consists of a word sequence encoder, a word-level attention layer, a sentence sequence encoder, and a sentence-level attention layer. Then, a new simple neural network architecture named the Transformer, based solely on attention mechanisms and dispensing with recurrence and convolutions entirely was proposed [31]. All these three attention neural networks are demonstrated to outperform other deep learning-based algorithms in sentiment analysis tasks. Presently, the authors of [32] introduced the Bidirectional Encoder Representations from Transformers (BERT) model, which is also based on the attention idea. The pre-trained BERT model can be fine-tuned with just one additional output layer to create simple and powerful algorithms and it obtained new state-of-the-art results on eleven different natural language processing tasks, including sentiment analysis. The authors of [33] proposed a movie recommender system that uses sentiment analysis data from Twitter, along with movie metadata and a social graph to recommend movies. Sentiment analysis provides information about how the audience responds to a particular movie and about how helpful this information is. The system used weighted score fusion to improve the recommendations. The authors of [17] proposed a user-based collaborative filtering approach that considers explicit ratings and sentiment analysis extracted from users' reviews. They use Matrix Factorization (MF) methods such as SVD, NMF, and SVD++ to predict explicit ratings, and use BERT, CNN, and LSTM to build the sentiment model. Results from the collaborative filtering approach and sentiment analysis are combined to generate integrated ratings and create a list of recommendations. However, inconsistent review-rating pairs can be very harmful and decrease the performance of the recommender system.

2.3. Use of Text Information in Collaborative Filtering

Collaborative filtering techniques perform well when there is sufficient rating information, but their effectiveness decreased when the sparsity problem occurs [34]. Consequently, in recent years, a variety of review-based recommender systems have been developed to incorporate valuable information embedded in user-generated textual comments [11]. Although the reviews are in unstructured textual forms, the advances in sentiment analysis make it possible to interpret and extract useful information [35].

Up until now, the information obtained from reviews is likely to benefit recommenders in four ways: (a) infer the required ratings as inputs, especially when the dataset is sparse, (b) determine the rating quality through the degree of the review's helpfulness, (c) extract a rating from the review and integrate it with the original rating, and (d) to provide additional information about user preferences such as latent features or feature opinions.

The authors of [36] applied Part-of-Speech tagging to extract adjectives and verbs as opinion words. Based on the relative strength into positive or negative classes of these words, they inferred the rating by aggregating all the opinion words contained in one review. Finally, using the inferred ratings, a collaborative filtering system was run. A similar approach was taken by [37], with the only difference they aggregate the sentiments expressed in emoticons and opinion words to infer an overall sentiment of a review. The authors of [38] also attempted to derive overall ratings from reviews, but they used machine learning to classify the opinions. Experimental results of the previous three studies indicate that collaborative filtering based on inferred ratings is comparable to the accuracy of a traditional approach based on real ratings, and even outperforms it in some cases.

Ozsoy [39] employed Word2Vec's skip-gram and CBOW techniques to make next check-in venue (location) recommendations on Location Based Social Networks (LBSNs). Unlike the previous works, this work used a non-textual feature, namely the past check-ins of the users, to make recommendations. Ozsoy [40] also proposed a method to recommend top-k venues. The method takes the sequentiality of check-ins into account and utilizes the FastText method from the NLP domain to learn the vector space embeddings of venues. The learned vector representations are used to calculate the similarity among venues and recommend the most similar venues to the already visited ones. The authors of [41,42] presented a new technique of incorporating reviews into collaborative filtering matrix factorization algorithms. The text of each review, of arbitrary length, is mapped to a continuous feature space of fixed length, using the Paragraph Vector model. Subsequently, the resulting feature vectors (the neural embeddings) are used in combination with the rating scores in a hybrid probabilistic matrix factorization algorithm, based on maximum a-posteriori estimation.

The authors of [15] used the votes of readers that denote if a review was helpful or not, to compute a quality score. These quality scores are taken as weights assigned to the ratings in a probabilistic matrix factorization for performing the rating prediction. The authors of [16] investigated three ways of combining overall opinions with real ratings in a biased matrix factorization model: (1) opinion pre-filtering, in which opinions are used to pre-process the whole real ratings, (2) opinion post-filtering, in which ratings and opinions are first used independently to train two prediction models, and then a linear combination is realized to obtain the final prediction, and (3) opinion modeling, in which opinions are used implicitly in the training of one joint model. The results showed that the second method, compared to the other two, and a standard algorithm based only on ratings, gives the best RMSE performance. The latter two stated research indicate that fusing reviews with ratings improves the accuracy of recommender systems relative to the standards that consider ratings alone.

The authors of [43] integrated extracted features from text and images through word2vec and Convolutional Neural Networks, respectively, into their proposed boosted inductive matrix completion method. The authors of [44] used Convolutional Neural Networks to extract latent factors from text data and integrate them into matrix factorization. [45] integrated the features extracted from content information through Stacked Denoising Autoencoders (SDAEs) with timeSVD++. The authors of [46] created a full Convolutional Neural Network with attention (D-Atnn) to obtain meaningful embeddings and an interpretable model mainly focusing on the words potentially important for the user. Practically, they directly executed the task of recommendation with the reviews as unique inputs. Similarly, the authors of [47] proposed a method based on embeddings and sentiment analysis. Their TRANSREV model learns vector representations for users, items, and reviews jointly, allowing the classifier to give a sentimental score for each review, which corresponds to the recommendation score.

3. Materials and Methods

3.1. System Overview

The proposed framework was designed to counteract the following two problems: (a) how to produce high-quality data, to be used as a boosted input for recommenders, and (b) how to generate product recommendations with higher accuracy. Consequently, its architecture comprises two main phases: (1) a quality inspector that assures that a rating provided by a user is consistent with the textual review that the same user expressed to a specific product, and (2) a recommender system that takes the preprocessed and clean data produced by a quality inspector and generates suitable items recommendations. Thus, the main functionality is to enhance the performance, specifically the accuracy, of a collaborative filtering-based recommender engine. Figure 1 illustrates the overall view of the proposed two-phase deep learning-based framework.

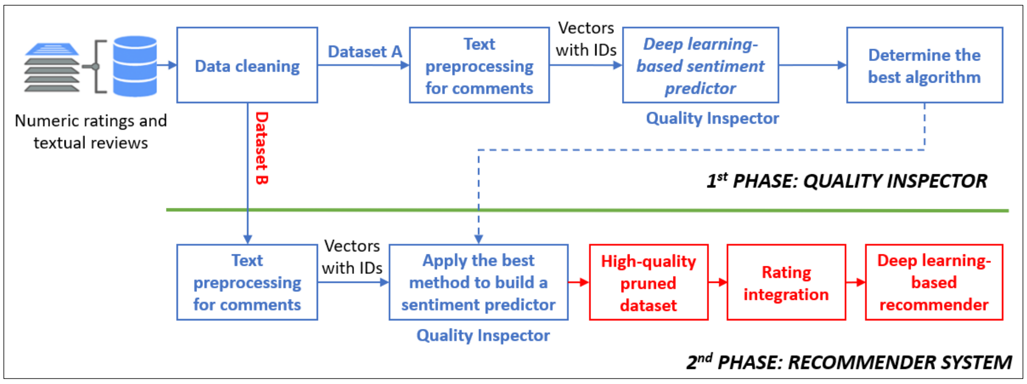
Numeric ratings and textual reviews
Data cleaning
Dataset A
Text preprocessing for comments
Vectors with IDs
Deep learning-based sentiment predictor
Quality Inspector
Determine the best algorithm
1st PHASE: QUALITY INSPECTOR
Dataset B
Text preprocessing for comments
Vectors with IDs
Apply the best method to build a sentiment predictor
Quality Inspector
High-quality pruned dataset
Rating integration
Deep learning-based recommender
2nd PHASE: RECOMMENDER SYSTEM

Figure 1. Overall view of the proposed framework.

Notice that two databases, one for each phase, are employed: dataset A and dataset B. They have the same structure, identical features, and an equal total number of data rows, but have different records. The main motivations to work with two sets are: (a) to apply dataset A in the first phase mainly to determine the best algorithm to model a sentiment predictor, and (b) to exploit dataset B in the second phase mostly to run the complete proposed procedure, composed of a quality inspector plus a recommender engine, completely from cero with a sole database.

The first phase aims to identify, through dataset A, the most accurate algorithm, for a sentiment predictor that predicts a rating from a textual review. In this study, it is assumed that the rating is an integer on a scale from 1 to 5. Accordingly, the core of the quality inspector is to check consistency between each review-rating record that a user gave to an item, determining whether it is high-quality or low-quality input data. Let r_i^s be the sentiment rating predicted by the predictor for record i, r_i^e be the explicit rating for record i, and k be the acceptable threshold defined by users. If

$$|(r_i^s - r_i^e)| \leq k \tag{1}$$

the i-th review-rating pair is considered as high-quality data and will be conserved. On the other hand, if $|r_i^s - r_i^e| > k$, the record is considered as inconsistent and will be discarded. The following equation is used to calculate the accuracy of the sentiment predictor:

$$accuracy = |\{i : |(r_i^s - r_i^e)| \leq k\}|/|\{i : i \in dataset\}| \tag{2}$$

In the second phase, the best algorithm determined in the first phase is applied to create from scratch, with dataset B, the quality inspector that produces a pruned high-quality version of dataset B. Afterward, the explicit rating r_i^e will be integrated with the predicted sentiment rating r_i^s into an integrated preference rating r_i^{pref} as:

$$r_i^{pref} = \alpha \times r_i^e + (1 - \alpha) \times r_i^s \tag{3}$$

where α is a weight factor ranging between 0 and 1. This integrated preference rating combines knowledge from two sources: (1) a numeric explicit rating, and (2) a textual sentiment review, being a robust input for a recommender system. Finally, to know the performance of the proposed recommenders, the following metrics will be tested (a) RMSE, (b) F1 score, (c) accuracy, and (d) running time. Naturally, the model with the highest results in nearly all the indicators is considered as the best recommender engine.

3.2. *Deep Learning-Based Quality Inspector*

In the first phase, the text preprocessing is applied to the text review of each record in dataset A. After a set of preprocessing tasks, the output is a vector with unique integer indexes (IDs). The obtained vector will be the input for a word embedding, which transfers the IDs of each vector to semantically-meaningful dense real-valued vectors, allowing words with similar meaning to have alike vectors and to position them close to each other in a high-dimension semantic space. Finally, the real-valued vector of each review and its explicit rating are the two inputs to train the sentiment predictor.

3.2.1. Text Preprocessing Tasks

Before applying the sentiment predictor to a dataset, which is composed of ratings and reviews, it is indispensable to preprocess every textual review. A total of seven sequential text preprocessing tasks were implemented to each textual opinion: (1) basic actions, (2) tokenization, (3) stop words removing, (4) stemming, (5) dictionary construction, (6) words transformation to unique integer indexes, and (7) the same length vector construction. The first task, basic actions, refers to three simple tasks: delete numbers and integers, delete the underscore symbol, and convert all the capital letters to lowercase. The second step is tokenization. For this specific research, the tokens are represented by words as the smallest indivisible units, and the tokenizer is conditioned to delete commas, apostrophes, and dots. The third step is to stop word removal. Since they are the most common words in a language and have a low contribution to the overall meaning of a sentence, they must be filtered out [48]. The fourth step is stemming, which helps to reduce words into their root forms by removing their prefixes and suffixes. As a result, a group of similar words is mapped to the same stem, even if it is not a valid word in the language.

Afterward, the whole set of preprocessed reviews can be used to construct a particular dictionary that allows preserving only the most important domain-specific words, and to convert them into unique integer indexes (IDs). The dictionary construction is based on the frequency count C of each unique word inside the entire set of preprocessed reviews. Then, a term count threshold T, established by the users, regulates which words to keep and to remove. If C of a word is higher than T, the token will be included in the dictionary and preserved in all the reviews, if not, the token will be excluded from the dictionary and removed from all the reviews. Next, a unique ID is assigned to every word of the dictionary. The sixth step is to transfer the remaining words of each vector to unique IDs through the use of the dictionary. The last task is to build all reviews to vectors of the same length of tokens. This is achieved with a length threshold L, which is specified by the user. If the real length R of a review is longer than L, the last words of the vector will be cut up. On the other hand, if R is shorter than L, then the initial part of the review will be padded with 0 s.

3.2.2. Deep Learning-Based Sentiment Predictor

The unique integer indexes, laying in the vectors of the same length, are projected to a dense vector space through the word embedding layer. The real-value matrices generated by the word embedding layer are the inputs of the deep learning network. Next to the deep learning layer, a Multi-Layer Perceptron (MPL) is applied. Let c_i be the i-th review vector, and r_i^e be the explicit rating for record i. The deep learning-based sentiment predictor can be then modeled as:

$$r_i^s = \phi_{out}\left(\phi_x\left(\ldots\phi_2\left(\phi_1\left(\phi_{dln}\left(\phi_{embedding}(c_i)\right)\right)\right)\ldots\right)\right) \quad (4)$$

where r_i^s is the predicted sentiment rating, $\phi_{embedding}$ is the embedding mapping function, ϕ_{dln} is the deep learning network mapping function, ϕ_x is the mapping function for the MLP x-th layer, and ϕ_{out} is the mapping function for the MLP output layer. The sentiment predictor utilizes a supervised learning technique for training, called backpropagation, where it tries to minimize the pointwise loss between r_i^s and its r_i^e. The proposed deep learning-based sentiment predictor is illustrated in Figure 2, and its accuracy can be determined using Equation (2).

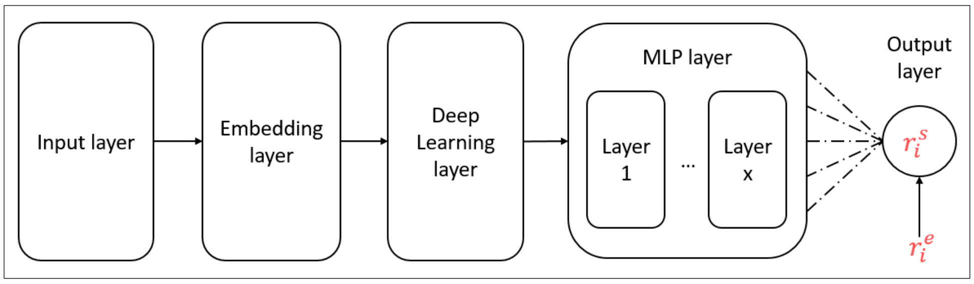

Figure 2. Architecture of the deep learning-based sentiment predictor.

In this study, a total of five deep learning-based sentiment predictors are tested in dataset A to create a highly accurate quality inspector: (1) LSTM, (2) Bidirectional LSTM, (3) CNN, (4) Two Deep Views Ensemble method, and (5) BERT.

- Model 1: Long-Short Term Memory Network

Long-Short Term Memory (LSTM) [49] was mostly developed to counteract the main constraint of Recurrent Neural Network (RNN). That is, as the words dependency in a sequence is longer, and the gap between previous relevant data and the current position grows, RNN becomes unable to learn to connect the information and often encounter with exploding gradients problem [50]. LSTM has a chain-like structure and is capable of learning long-term dependencies as a result of four interacting neural network layers set in its repeating module: a forget gate, an input gate, a normal layer, and an output gate.

- Model 2: Bidirectional LSTM

Bidirectional LSTM (Bi-LSTM) [51] is an extension of a traditional LSTM. It trains two LSTM layers on an input sequence: the first as-is from right to left and the second on a reversed copy from left to right. This can provide additional context to the network and result in even fuller learning on the problem, improving in some cases performance on sequence classification problems. BI-LSTM can efficiently make use of past features via forward states, and future features via backward states, using information from both ends of the sequence to estimate the output.

- Model 3: Convolutional Neural Network

Convolutional neural network (CNN) is a category of deep learning neural networks that were designed with the main purpose to solve image (characters) recognition and classification tasks. It was developed by [52], and its architecture was called LeNet. Afterward, many different CNNs, based on LeNet, were developed, resulting in many successful applications in the computer vision field. Nowadays, it is also starting to be used in Natural Language Processing (NLP) area and has gotten interesting and promising results [53]. For the previous reason, it was selected as the third deep-learning model to being tested.

- Model 4: Two Deep Views Ensemble Method

Two Deep Views Ensemble method (TDVE) is a modified text classification model inspired by the methodology proposed by [28]. Their suggested framework was adapted to the specific case of this study. Thus, the factorization machines-based co-training component is excluded, the prediction component is slightly modified, and the two deep views module is used as the core idea of this fourth model. The two deep views are: (1) a CNN, which focuses on extracting local features, and (2) a Dependency-Sensitive Convolutional Neural Network (DSCNN), which focuses on capturing long-distance dependency data. The attention of each one achieves the characteristic of two conditionally independent, and sufficient views, to create a fused sentiment predictor through ensemble learning. Figure 3 shows the design of this method.

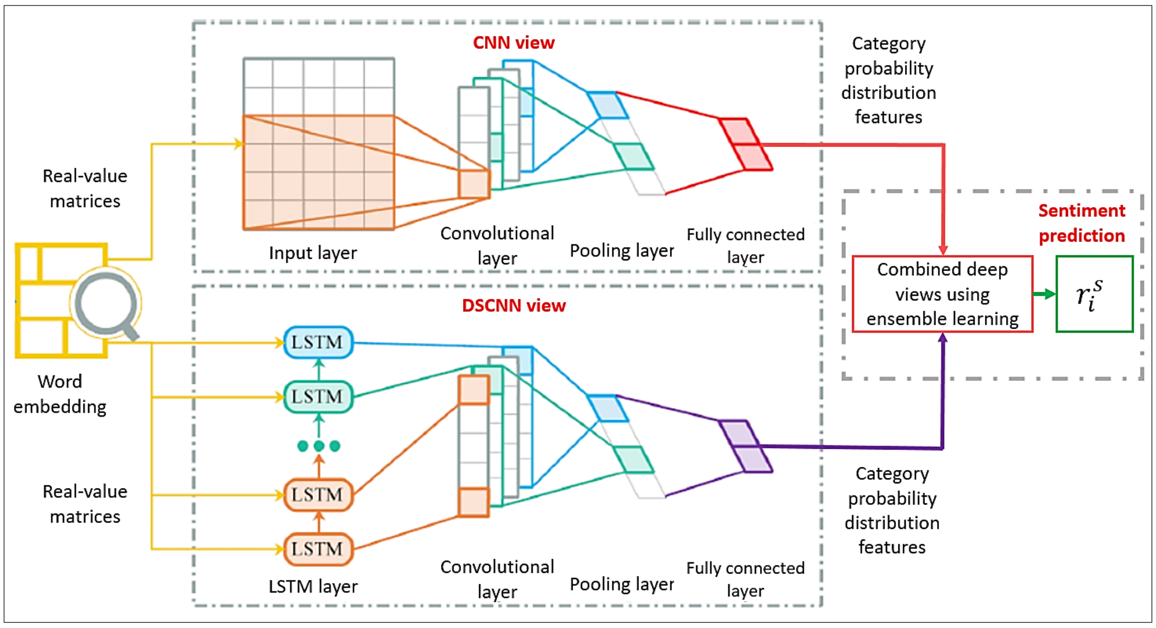

Figure 3. Design of the modified TDVE for text classification [28].

- Model 5: BERT

Bidirectional Encoder Representations from Transformers (BERT) is a language model proposed by [32]. This algorithm is designed to pre-train deep bidirectional representations from the unlabeled text by jointly conditioning on both the left and the right context in all layers. As a result, the pre-trained BERT model can then be just fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks. Since it was empirically proved to obtain top-of-the-line results on eleven natural language processing problems, including pushing the GLUE score to 80.5%, MultiNLI accuracy to 86.7%, SQuAD v1.1 question answering Test F1 to 93.2 and SQuAD v2.0 Test F1 to 83.1, and to outperform many famous designs that only pre-train deep unidirectional representations, it was chosen as the fifth model. Regarding its architecture, it relies on the Transformer [31], an attention mechanism that learns contextual relations between words in a text. A vanilla Transformer includes: (a) an encoder to read the text input, and (b) a decoder to produce a prediction for the task. However, since the goal of BERT is to generate a language representation model, it only needs the encoder, which reads the entire sequence of words at once, and understands the relation weights between all the words in the text, regardless of their position, as opposed to a directional strategy, which reads the input sequentially. Moreover, during its pre-training, instead of predicting the next word one by one, it uses two novel training techniques: (1) masked language model, and (2) next sentence prediction. These three characteristics, allow BERT to learn the context of a word based on all of its surroundings, taking both the previous and next tokens into account at the same time. Therefore, although it is considered bidirectional, it would be more accurate to say that it is a non-directional approach [54,55].

3.2.3. Process to Build a Quality Inspector

The visual guide regarding the process and the terminology used to create a quality inspector model, functional for both the first and second phases, is presented in Figure 4.

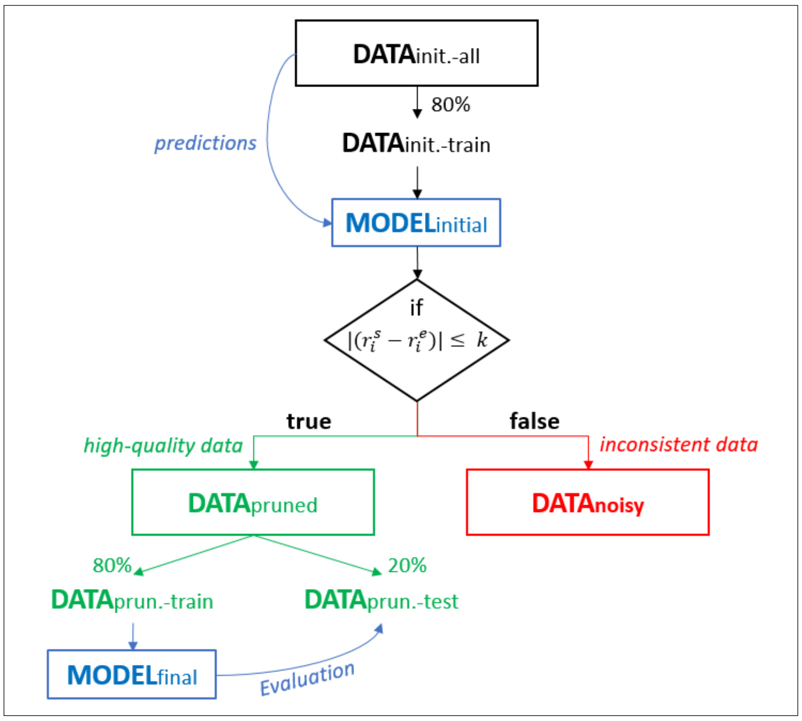

Figure 4. Visualization guide regarding the process, and terminology used, to create a quality inspector model.

In the following discussion, either dataset A or dataset B is denoted as the initial dataset $DATA_{init.-all}$. Note this process, founded on three sequential tasks, will be applied for each of the five deep learning-based models mentioned in Section 3.2.2.

In the first task, we randomly select 80% of records from $DATA_{init.-all}$ to form $DATA_{init.-train}$. The $DATA_{init.-train}$, a dataset containing noise, is used to train and produce a $MODEL_{initial}$ by using the reviews as features and the ratings as labels. In the second task, $MODEL_{initial}$ will be used for rating predictions using $DATA_{initial-all}$. As mentioned in Equation (1), if $\left|\left(r_i^s - r_i^e\right)\right| \leq k$, the i-th review-rating record is considered as high-quality data and will be conserved and saved in the dataset $DATA_{pruned}$. On the other hand, if $\left|r_i^s - r_i^e\right| > k$, the record is considered inconsistent and will be discarded. Here, the threshold k must be equal to 0, as it is critical to be strict and just keep the data that have high probabilities of being consistent. In the third task, $DATA_{prun.-train}$, randomly derived from the high-quality dataset $DATA_{pruned}$, is utilized to construct a $MODEL_{final}$ for each algorithm. Naturally, $MODEL_{final}$ is tested using the $DATA_{prun.-test}$ to report the performance indicators.

The last explained three tasks create a powerful sentiment predictor. Once our $MODEL_{final}$ is created, it can be successfully used as a highly accurate quality inspector to prune $DATA_{initial-all}$, producing an enhanced input for a recommender engine. Here, the threshold k used is 1, as this tolerance achieves a suitable balance between discarding an appropriate amount of noisy data while remaining a pertinent number of total records. Thus, the quality of the dataset is improved while its initial sparsity is not affected.

3.3. Deep Learning-Based Product Recommender

Three inputs are used to generate an accurate product recommendation: (a) user identification, (b) item identification, and (c) r^{pref}, which is an integer value between 1 and 5 derived by Equation (3). Formally, let $\boldsymbol{u}_j$ be the user ID vector of record j, $\boldsymbol{i}_j$ be the item ID vector of record j, and r_j^{pref} be the preference rating of record j. The vectors $\boldsymbol{u}_j$ and $\boldsymbol{i}_j$ are then projected to a dense vector space as real-value vectors $\boldsymbol{p}_{\boldsymbol{u}_j}$ and $\boldsymbol{q}_{\boldsymbol{i}_j}$, respectively, through an embedding layer. The interaction between the characteristics of u and i is then modeled and captured by an interaction function. This is the input for the deep neural network. The deep learning-based rating predictor of a product recommender can then be modeled as follow:

$$\hat{r}_j^{pref} = \phi_{out}\left(\phi_x\left(\ldots\phi_2\left(\phi_1\left(\phi_{int}\left(\phi_{embedding}\left(\boldsymbol{u}_j,\ \boldsymbol{i}_j\right)\right)\right)\right)\ldots\right)\right) \tag{5}$$

where $\hat{r}_j^{pref}$ is the predicted preference rating, $\phi_{embedding}$ is the embedding mapping function, ϕ_{int} is the mapping function to model the interaction between a user and an item, ϕ_x is the mapping function for the MLP x-th layer, and ϕ_{out} is the mapping function for the MLP output layer. The rating predictor of a recommender also utilizes for training, a backpropagation supervised learning technique, where it tries to minimize the pointwise loss between the predicted preference rating $\hat{r}_j^{pref}$ and its target r_j^{pref}. Figure 5 shows the framework of the proposed deep learning-based product recommender.

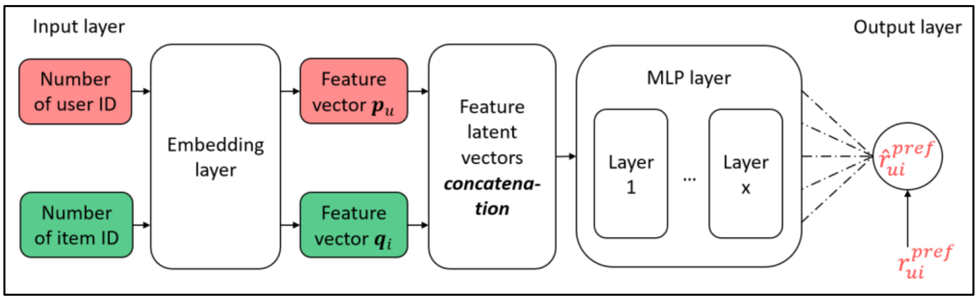

Figure 5. Architecture of the deep learning-based recommender.

Lastly, to recommend a product i to a user u, two criteria must be fulfilled: (a) the predicted preference rating $\hat{r}_j^{pref}$ must be equal or higher than 4, and (b) the user u must have not rated product i before. Only if these two conditions are satisfied, product i is recommended to user u. Two deep learning-based recommender models are applied in this study: (1) NNMF, and (2) DLbR.

- Model 1: Neural Network Matrix Factorization

Neural Network Matrix Factorization (NNMF) is an adapted version of the neural collaborative filtering-based recommender model [21]. The model is composed of a Generalized Matrix Factorization (GMF) component, and a Multi-Layer Perceptron (MLP) module, unifying linearity strength with non-linearity characteristics. Therefore, it was selected as the base, and slightly modified, to create the first deep-learning recommender. The following are the performed tweaks done to the NNMF model: (a) the explicit ratings were directly used as the inputs, instead of converting them to implicit information, (b) the model addresses a 5 label, instead of a binary, classification problem, (c) it uses a categorical, instead of a binary, cross-entropy loss function, (d) it utilizes softmax, instead of sigmoid, as the activation function of the output layer, and (e) it realizes a direct training with random weight initializations of the global model, instead of previously pre-train the two components separately. Figure 6 summarizes the architecture of the recommender. As a collaborative filtering approach, the inputs for the two components are just basic information: a user ID and a product ID. Note that each component has its own embedding layer, and thus, a different user latent feature vector $\boldsymbol{p}_u$, and a different item latent feature vector $\boldsymbol{q}_i$, are created for each component, where $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$ describe every user u and item i, respectively.

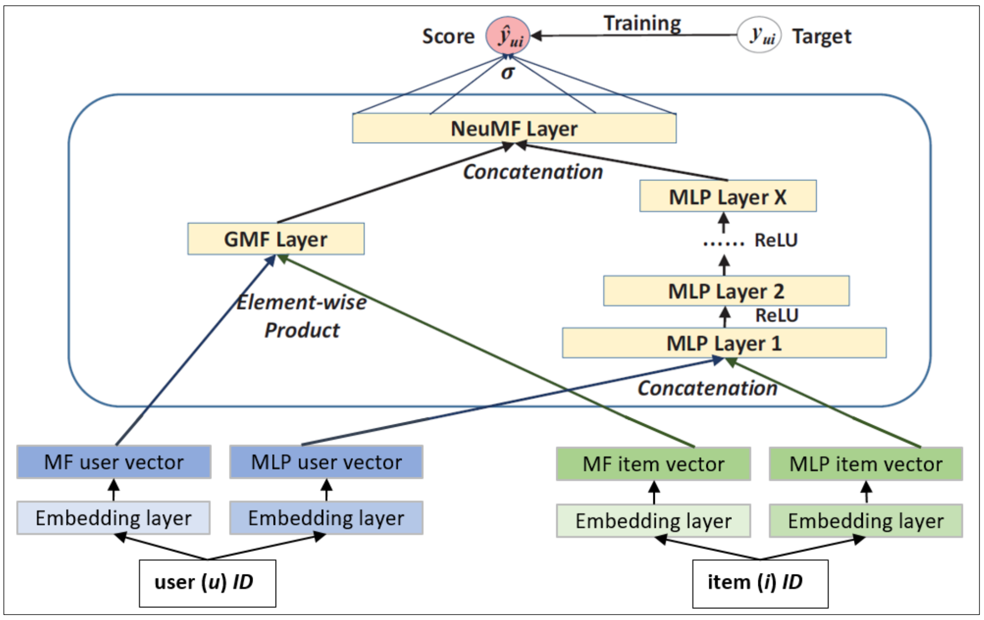

Figure 6. Architecture of the modified NNMF model [21].

- Model 2: Deep Learning-based Recommender

Deep Learning-based Recommender (DLbR) model is a neural network approach for movie recommenders [56]. It is suggested to take a product ID and a user ID as inputs. Both IDs are just single numbers and are turned into dense vector representations with an embedding layer. These two feature latent vectors are merged by concatenating them. The concatenation result is the input fed into an MLP network. Finally, the classification prediction is done with softmax as an activation function. It is easy to perceive that DLbR is exactly the same as the second component (MLP) of the NNMF model.

4. Results

4.1. Data Collection and Datasets Preparation

The data used in this research is the Amazon Review Data (2018) [57]. It is available on the web for public use in two versions: (a) Complete Review Data and (b) Small Subset for Experimentation. The Movies and TV database, located in the K-Cores files of the Small Subset for Experimentation, is selected for all the experiments. Naturally, not all the columns in each record are necessary for the proposed framework and thus seven features were omitted. Additionally, the records with information older than the year 2010 were excluded, and therefore, the preliminary dataset contains 2.92 M rows with five features. Table 1 shows the first 5 rows of this preliminary dataset.

Table 1. First 5 rows example of the preliminary dataset of 2.92 M rows with 5 columns.

Review Time	User ID	Item ID	Rating	Textual Review
2012	A2M1CU2IRZG0K9	5089549	2	So sorry I didn't purchase this years ago ...
2011	AFTUJYISOFHY6	5089549	3	Believe me when I tell you that you will ...
2005	A3JVF9Y53BEOGC	503860X	5	I have seen X live many times, both in ...
2005	A12VPEOEZS1KTC	503860X	4	I was so excited for this! Finally, a live ...
2010	ATLZNVLYKP9AZ	503860X	5	X is one of the best punk bands ever. I do ...

Unsurprisingly, the records in the preliminary dataset may contain many no valuable data such as (a) records with only a numeric value as the review, (b) rows with only two or one character in total as the review, (c) reviews with specific characters that generate an error in their specific cells when exporting the data file to Excel, or (d) rows with just a blank cell instead of a review. Consequently, a data cleaning process that includes three filters, and one preprocessing task is conducted. Figure 7 details the data cleaning process.

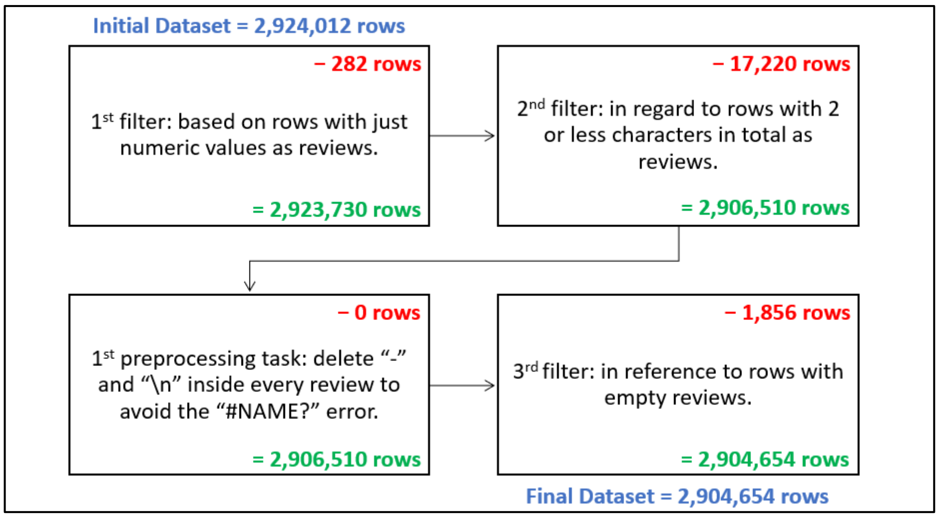
Initial Dataset = 2,924,012 rows
– 282 rows
1st filter: based on rows with just numeric values as reviews.
= 2,923,730 rows
– 17,220 rows
2nd filter: in regard to rows with 2 or less characters in total as reviews.
= 2,906,510 rows
– 0 rows
1st preprocessing task: delete "-" and "\n" inside every review to avoid the "#NAME?" error.
= 2,906,510 rows
– 1,856 rows
3rd filter: in reference to rows with empty reviews.
= 2,904,654 rows
Final Dataset = 2,904,654 rows

Figure 7. Complete view of the data cleaning process.

A final dataset of 2.90 M rows with five features is obtained. To facilitate the two-phase experiments, 600,000 records, 120,000 rows for every rating label, are randomly selected from the final dataset to create dataset A. Similarly, another 600,000 rows, 120,000 records per rating class, were randomly extracted from the final dataset to form dataset B.

4.2. Deep Learning-Based Sentiment Inspector

4.2.1. Performance Measures

The first phase experiments are divided into two steps. The first step aims to find the parameters that achieve the best rating prediction results for each model by performing various fast experiments. To fulfill the need, a smaller version of dataset A is employed. Once the parameters of each proposed algorithm are established and its best version is well-defined, the second step is executed, where the complete dataset A is exploited to test the quality inspectors. Remark that for each of these two steps, all the three tasks explained in Section 3.2.3, were run.

Three indicators are used to determine the performance of the different quality inspector models: (a) *accuracy0*, (b) *accuracy1*, and (c) *total time to run each model*. The first two indicators can be derived using Equation (2) where k is the allowed prediction threshold. Accordingly, *accuracy0* is calculated when $k = 0$, and *accuracy1* when $k = 1$. The third indicator is modeled as:

$$\begin{aligned} total\ time\ to\quad & run\ each\ model \\ & = MODEL_{initial}\ training\ time\ with\ DATA_{init.-train} \\ & + MODEL_{initial}\ prediction\ time\ to\ DATA_{init.-all} \\ & + pruning\ time\ of\ DATA_{init.-all} \\ & + MODEL_{final}\ training\ time\ with\ DATA_{prun.-train} \end{aligned} \tag{6}$$

4.2.2. Text Preprocessing Tasks

The first four text preprocessing tasks, described in Section 3.2.1, generates a set of preprocessed reviews. Subsequently, the entire set of preprocessed texts, with $T = 4$, is utilized to construct dataset A dictionary, which has 49,726 unique tokens. Remark that dataset B dictionary was also constructed with $T = 4$ and possesses 49,688 singular tokens. Finally, the entire set of preprocessed texts are modified through the last two text preprocessing tasks, producing the required input for a word embedding. Since the preprocessed reviews of dataset A averaged 41 ± 84 tokens per review, and the ones of dataset B averaged 41 ± 83 tokens per review, it was decided to use an $L = 125$.

4.2.3. Predictors Comparison

From dataset A, a total of 75,000 rows, 15,000 rows per rating, were extracted to create a smaller version, which allows fast experimenting of each model through an iterative parameter adjustment process. Specifically, 3 to 6 scenarios (versions) with different parameters were tested for every model. For example, five versions of LSTM were tried in which the length threshold L, the LSTM layer output size, the embedding layer output size, and the number of training epochs were adjusted. After that, the best version per model, based on the three evaluation indicators, was selected. The best architectures of the five deep learning-based sentiment predictors are detailed in Table 2.

Table 2. Parameter settings for the best version of deep learning-based models.

Model	Parameters
Model 1: LSTM (version 5)	1 embedding layer (32)—1 lstm layer (32)—1 hidden lay. (256)—1 output lay. (softmax)—batch: 500—epochs: 30—optimizer: adam
Model 2: Bi-LSTM (version 2)	1 embedding layer (32)—1 bi-lstm layer (32)—1 hidden lay. (256)—1 output lay. (softmax)—batch: 500—epochs: 30—optimizer: adam
Model 3: CNN (version 5)	1 emb. lay. (32)—1 cnn lay.—100 filters—filt. size: 10—1 max pool.—1 hidden lay. (256)—1 out. lay. (softmax)—batch: 500—epochs: 30—optimizer: adam
Model 4: TDVE (version 3)	*CNN view* 1 emb. lay. (32)—1 cnn with l2 (0.01) const.—100 filters with size: 3, 4, 5—1 max pooling—1 hidden lay. (256)—1 out. lay. (softmax)—batch: 500—epochs: 30—optimizer: adadelta *DSCNN view* 1 emb. (32)—1 lstm (32)—1 cnn without l2–100 filters with size: 3, 4, 5—1 max pooling—1 hidden lay. (256)—1 out. lay. (softmax)—batch: 500—epochs: 30—optimizer: adadelta
Model 5: BERT (version 2)	1 bert layer—1 avg. pooling—1 dropout: 0.2—1 hidden lay. (256)—1 out. lay. (softmax)—batch: 16—epochs: 8—optimizer: adam

The best version of each algorithm presented in Table 2 was preserved for the following model comparison, where the complete dataset A is applied.

The final results and the ranking of every algorithm per indicator are summarized in Table 3. Note a traditional machine learning model, Support Vector Machine (SVM), is included as a comparison base. Its design relies on an rbf kernel and a regularization parameter $C = 1$. The first sub-table contains the *accuracy0* results of the particular six versions, sorted from the best to the worst. Similarly, the second sub-table comprises *accuracy1* results, and the third sub-table the *total running time* in hours. It can be concluded that the best model related to *accuracy0* is BERT-2, i.e., version 2 of BERT. Regarding *accuracy1* the best model is also BERT-2. However, concerned with the *total running time*, the best model is CNN-5, i.e., version 5 of CNN.

The process to determine the best algorithm is described with the help of Table 4. The first sub-table has the ranking positions of each model for the three indicators, and it is based on the information in Table 3. For example, TDVE-3 is in ranking 2 respecting the 1st indicator, but in ranking 3 regarding the 2nd indicator, and in ranking 4 regarding the 3rd indicator. The first sub-table also contains the weights of importance, related to this study, for each indicator. Since the proposed framework is supposed to work offline, time is not a critical factor and that is why the *total running time* has only 15% of importance weight. On the other hand, to create an efficient data quality inspector, it is essential to create a model able to predict accurately the sentiment of a review. As a result, the sum of the importance weights of the two accuracy indicators is equal to 85%, where *accuracy0* has a 35% of importance weight and *accuracy1* has a 50% of importance weight. The second sub-table has the final score for each method. The final score is easily obtained by applying the sum product of the rankings per indicator of each model by the weights of each indicator. The six models were sorted, from the lowest to highest score, based on the second sub-table. Since BERT-2 achieves the closest score to 1.0 points, it is considered as the overall best quality inspector. LSTM-5 accomplishes second place, TDVE-3 earns the third position, and SVM-1 is the worst model.

Table 3. Performance results summary of the models per indicator.

	1st Indicator			**2nd Indicator**			**3rd Indicator**	
	Ranking	**Accuracy 0**		**Ranking**	**Accuracy 1**		**Ranking**	**Time (h)**
1	BERT-2	94.37%	1	BERT-2	99.12%	1	CNN-5	2.5
2	TDVE-3	92.02%	2	LSTM-5	99.04%	2	LSTM-5	3.6
3	LSTM-5	91.02%	3	TDVE-3	98.94%	3	BILSTM-2	4.7
4	BILSTM-2	89.90%	4	BILSTM-2	98.80%	4	TDVE-3	8.4
5	CNN-5	88.30%	5	CNN-5	98.44%	5	BERT-2	23.2
6	SVM-1	37.36%	6	SVM-1	69.53%	6	SVM-1	147.4

Table 4. Overall ranking of the models with weights per indicator.

Weights of Ind.	**35%**	**50%**	**15%**	**Final Score**	**Final Ranking of Models**	
Models	**1st Ind.**	**2nd Ind.**	**3rd Ind.**			
BERT-2	1	1	5	1.60	BERT-2	1st place
LSTM-5	3	2	2	2.35	LSTM-5	2nd place
BILSTM-2	4	4	3	3.85	TDVE-3	3rd place
CNN-5	5	5	1	4.40	BILSTM-2	4th place
SVM-1	6	6	6	6.00	CNN-5	5th place
TDVE-3	2	3	4	2.80	SVM-1	6th place

Moreover, Figure 8 shows four scenarios, where the weight distribution of the three indicators, critical to computing the final ranking of the models, is varied. The sensitivity analysis demonstrates that the final ranking of the six models is stable and not affected by tweaks in the weight distribution of the three metrics. It is concluded BERT-2 and LSTM-5 are the two best models and both are chosen to be tested as the algorithm behind the quality inspector of the second phase: deep learning-based product recommender.

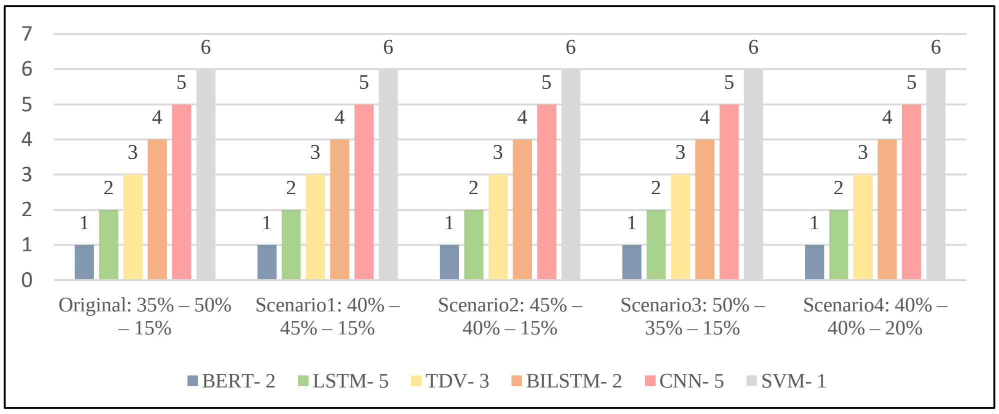

Figure 8. Final ranking of models through sensitivity analysis of the three indicators' weight distribution.

4.3. Deep Learning-Based Product Recommender

4.3.1. Performance Measures

Four indicators were used to determine the performance of the recommender engines. The first is the Root Mean Square Error (RMSE) and its formula is:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}\left(r_{ui}^{pref} - \hat{r}_{ui}^{pref}\right)^2} \tag{7}$$

where N denotes the total number of review-rating pairs that the dataset contains, r_{ui}^{pref} is the preference rating that user u gave to item i, and $\hat{r}_{ui}^{pref}$ is the predicted preference rating of u for i, done by the recommender engine.

The second indicator is the *F1 score*, which takes recall and precision into consideration. The equation is as follow:

$$f1\ score = (2 * precision * recall)/(precision + recall) \tag{8}$$

$$precision = (rec.\ relevant\ items)/(total\ of\ items\ recommended) \tag{9}$$

$$recall = (rec.\ relevant\ items)/(total\ of\ relevant\ items) \tag{10}$$

where *rec. relevant items* refers to the number of item recommendations made by the algorithm that are truly relevant for the user, i.e., r_{ui}^{pref} is equal to or higher than 4.0 points. The precision can be defined as, from the total of item recommendations produced, how many are truly relevant for the users. On the other hand, recall expresses, from the total of truly relevant items that must be suggested, how many were actually recommended.

The third indicator is the recommendation *accuracy*. Its math formula can be expressed as:

$$accuracy = (rec.\ relevant\ items + not\ rec.\ not\ relevant\ items)/N \tag{11}$$

where *not rec. not relevant items* denotes the number of items not recommended by the algorithm that are truly not relevant for the user, i.e., r_{ui}^{pref} is lower than 4.0 points. Finally, the fourth indicator is the required *time to run the recommender*, which is composed of the time of learning the patterns of the input data, plus the time of predicting $\hat{r}_{ui}^{pref}$ for each r_{ui}^{pref} located in the dataset.

4.3.2. Quality Inspector Results

The second phase experiments of the proposed framework run the complete process, which is composed of two parts. The first part applies the three sequential tasks described in Section 3.2.3 with BERT-2 and LSTM-5, to produce from dataset B, two different quality inspectors. Subsequently, these two are exploited to prune and clean, with k set to 1, the initial dataset B, also denoted as $DATA_{init.-all}$ (600,000 records). The output of the LSTM-5 quality inspector is denominated $DATA_{pruned\ LSTM5}$ and consists of 547,032 records (91.17% of 600,000). Similarly, the output of the BERT-2 quality inspector is named $DATA_{pruned\ BERT2}$ and has a total of 505,269 records (84.21% of 600,000). Then, the second part is executed, which consists to test the different recommender systems with these three databases generated from dataset B: (1) $DATA_{init.-all}$: no quality inspector applied, (2) $DATA_{pruned\ LSTM5}$: LSTM quality inspector applied, and (3) $DATA_{pruned\ BERT2}$: BERT quality inspector applied.

4.3.3. Recommenders Comparison

Table 5 displays the architecture of the NNMF model and Table 6 exhibits the design of the DLbR model. Besides, two traditional machine learning models, Singular Value Decomposition (SVD) and SVD++, are included as a comparison base. SVD is mainly based on 100 latent factors for users and items, a total of 20 training epochs, and is trained and tested with 100% of the dataset. In contrast, SVD++ is mostly based on 20 latent factors for users and items, a total of 20 training epochs, and is trained and tested with 100% of the dataset. It is remarked that the key objectives of this study are to identify if a recommender system works more accurately with or without a quality inspector, and to determine if deep learning algorithms for collaborative filtering recommenders are more efficient than traditional methods. Therefore, it is not necessary to have a fair comparison, none of these four methods was severely optimized.

Table 5. Architecture of NNMF recommender engine.

Users Inputs (IDs)		**Items Inputs (IDs)**	
GMF block		**MLP block**	
embedding layer (32)—user latent vectors.	embedding layer (32)—item latent vectors.	embedding layer (32)—user latent vectors.	embedding layer (32)—item latent vectors.
dropout layer: 0.5		dropout layer: 0.5	
GMF layer: inner product of user and item vectors.		MLP layer: concatenate user and item vectors.	
		dropout: 0.5—1 hidden lay. (128) with relu.	
		dropout: 0.5—1 hidden lay. (128) with relu.	
		dropout: 0.5—1 hidden lay. (128) with relu.	
concatenation layer to concatenate GMF and MLP feature vectors into a unique one.			
1 output layer (5) with softmax as activation function—Optimizer: adam—batch size: 500			
epochs: 20–100% (0% val.) trained and tested with the 100% of data.			

Table 6. Architecture of DLbR recommender system.

Users Inputs (IDs)	**Items Inputs (IDs)**
embedding layer (32)—user latent vectors.	embedding layer (32)—item latent vectors.
dropout layer: 0.5—concatenation layer to concatenate user and item vectors into a unique one.	
1 dropout layer: 0.5—1 hidden layer (128)—activation function: relu—1 batch normalization layer.	
1 dropout layer: 0.5—1 hidden layer (128)—activation function: relu—1 batch normalization layer.	
1 hidden layer (128)—activation function: relu.	
1 output layer (5) with softmax as activation function—Optimizer: adam—batch size: 500	
epochs: 20–100% (0% val.) trained and tested with 100% of data.	

Table 7 illustrates the results that the recommendation engines achieved in each of the three datasets. It can be perceived that the table is divided into three parts: (1) the indicators of the four models with $DATA_{init.-all}$, (2) the results of all the methods with $DATA_{pruned\ LSTM5}$, and (3) the indicators of every single algorithm with $DATA_{pruned\ BERT2}$. As shown in Table 7, it can be established that the best recommender is NNMF. It achieves the highest *F1 score*, the highest *accuracy*, and the lowest *RMSE*. Although it is the model that took the longest *time to run* its algorithm, the time is not taken as a critical factor because the proposed framework is supposed to work offline. The second position as the best recommender is assigned to DLbR since even though its *RMSE* is the worst of the four models, and its *total running time* is in the third position, it achieves a good *F1 score* and a suitable *accuracy*. Moreover, SVD++ earns the third position, as it achieves the second position in *total running time*, and the third position in *RMSE*, *F1 score,* and *accuracy*. Finally, SVD is concluded as the worst model.

Table 7. Results achieved per each model with three datasets generated from Movies and TV.

		NNMF	**DLbR**	**SVD**	**SVD++**
DATAinit.-all	RMSE	0.294	0.962	0.814	0.920
	F1 score	98.48%	83.07%	50.42%	60.67%
	accuracy	98.79%	84.69%	73.38%	76.15%
	time (min)	26.44	24.72	4.22	6.82
DATApruned LSTM5	RMSE	0.287	0.902	0.801	0.894
	F1 score	98.71%	86.95%	52.35%	61.81%
	accuracy	98.82%	87.17%	75.57%	77.90%
	time (min)	23.71	23.09	3.79	5.94
DATApruned BERT2	RMSE	0.284	0.869	0.805	0.892
	F1 score	98.62%	88.37%	53.45%	63.00%
	accuracy	98.75%	88.73%	77.00%	79.28%
	time (min)	22.19	21.13	3.48	5.24

As mentioned before, the F1 score is calculated by using precision and recall as inputs. Since recall is based on the list of relevant items to users within a dataset, it is important to clarify that the size of this list is different for each dataset. For example, $DATA_{init.-all}$ possesses 240,000 items (40% of 600,000) that are relevant to users, $DATA_{pruned\ LSTM5}$ has 205,126 items (38% of 547,032) that must be recommended to users, and $DATA_{pruned\ BERT2}$ holds 180,981 items (36% of 505,269) that are relevant to users.

The performance trend per indicator for every method, produced when using LSTM-5 or BERT-2 against the results obtained with $DATA_{init.-all}$, is clarified from Figures 9–12. For instance, as shown in Figure 9, SVD++ improves RMSE by 3.09% and 2.87% when executing BERT-2 and LSTM-5, respectively. The highest decrease is reported in DLbR, reducing RMSE by 9.60% and 6.25% when using $DATA_{pruned\ BERT2}$ and $DATA_{pruned\ LSTM5}$, respectively. Lastly, although SVD stated the lowest decrease, it still experienced satisfactory enhancements when using the two quality inspectors.

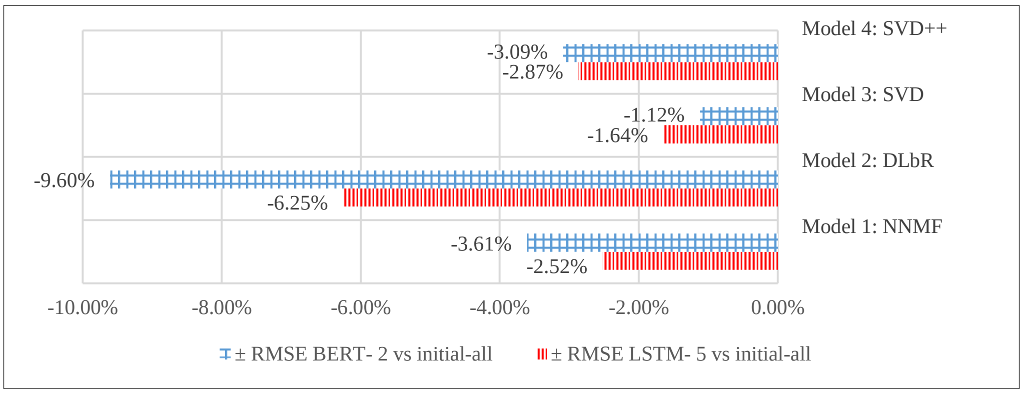

Figure 9. Comparison of the variations produced when using LSTM-5 or BERT-2 against the dataset $DATA_{init.-all}$, of the four models in the RMSE indicator.

Similarly, Figure 10 reports that DLbR enhances the F1 score by 5.30% and 3.89% when exploiting BERT-2 and LSTM-5, respectively. The lowest boost is generated by NNMF, which improves the F1 score by 0.14% when using $DATA_{pruned\ BERT2}$ and 0.23% when utilizing $DATA_{pruned\ LSTM5}$. Regarding SVD++ and SVD, both experienced a regular F1 score increase between 1.13% and 3.03% when applying BERT-2 or LSTM-5 quality inspectors. Likewise, Figure 11 proves the positive impact on accuracy when applying a quality inspector, where all the models, except for NNMF, reported a steady accuracy boost between 1.74% and 4.04%. Finally, as shown in Figure 12, total running time is the indicator that encountered the best enhancements, as the four models report a total running time reduction between 6.60% and 23.23% when using a cleaned version of the data.

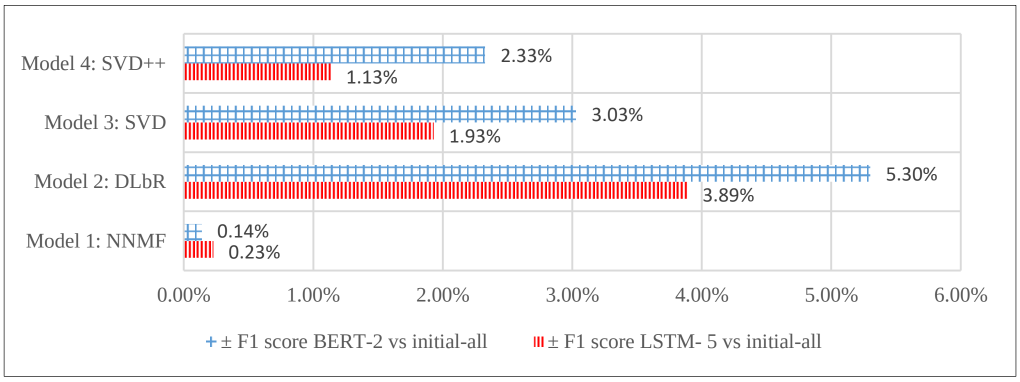

Figure 10. Comparison of the variations produced when using LSTM-5 or BERT-2 against the dataset $DATA_{init.-all}$, of the four models in the F1 score parameter.

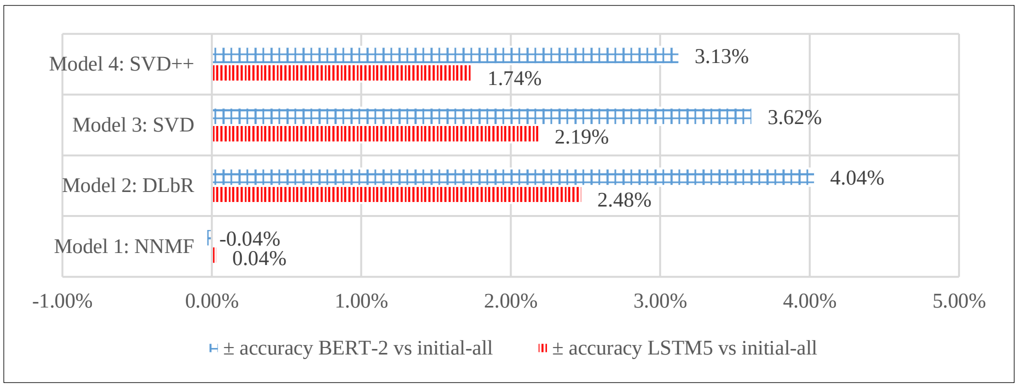

Figure 11. Comparison of the variations produced when using LSTM-5 or BERT-2 against the dataset $DATA_{init.-all}$, of the four models in accuracy indicator.

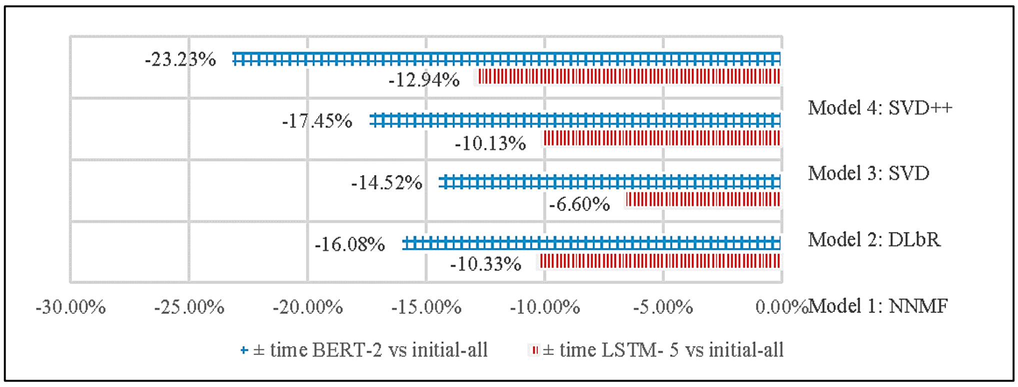

Figure 12. Comparison of the variations produced when using LSTM-5 or BERT-2 against the dataset $DATA_{init.-all}$, of the four models in total running time.

Figure 13 shows the average percent variations experienced by the four methods, per indicator, and for every quality inspector. For example, all recommenders averaged an accuracy increase of 2.69% when applying BERT- 2, but an increase of 1.61% when using $DATA_{pruned\ LSTM5}$. BERT- 2 outperforms the results of LSTM- 5 in all the four indicators, and thus, it is considered the best quality inspector.

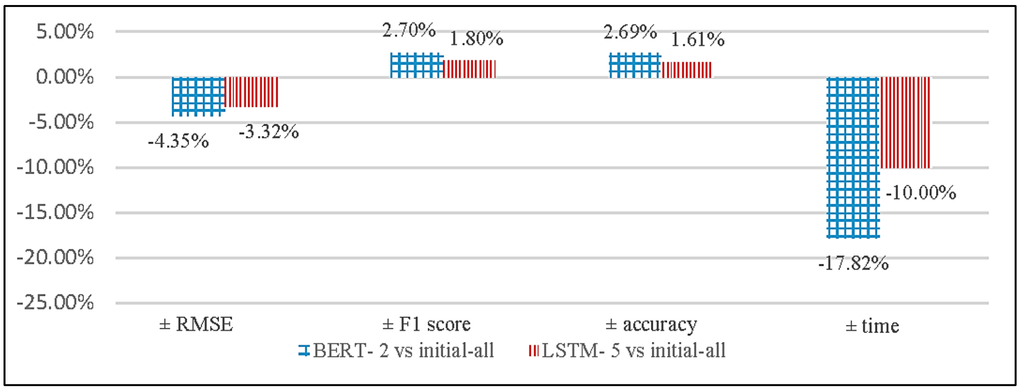

Figure 13. Percent variations averages experienced by the four models, per indicator, and for every quality inspector with Movies and TV data.

Additionally, popular rank-based metrics for Top-K lists, which are Normalized Discounted Cumulative Gain (NDCG) and Mean Average Precision (MAP), were evaluated using $DATA_{pruned\ BERT2}$. Table 8 summarizes the experiment results where K = 5, 10, and 15. It is noted that NNMF is the algorithm that achieved the best performance in NDCG, while DLbR is the algorithm performing the best in MAP for all K. SVD achieved the second-best outputs in NDCG but the lowest ones in MAP, and SVD++ accomplished the third position in both NDCG and MAP. These outcomes also indicate the benefits of applying deep learning to model the algorithm of recommender engines.

Table 8. Rank-based metrics for the four models using $DATA_{pruned\ BERT2}$ dataset.

		NNMF	**DLbR**	**SVD**	**SVD++**
K = 5	NDCG	99.77%	91.37%	95.90%	91.50%
	MAP	40.00%	55.00%	28.00%	28.00%
K = 10	NDCG	99.77%	91.37%	95.90%	91.50%
	MAP	20.00%	33.21%	14.00%	19.29%
K = 15	NDCG	99.77%	91.37%	95.90%	91.50%
	MAP	13.33%	22.14%	9.33%	9.33%

4.3.4. Other Datasets

Except for the Amazon Movies and TV data, another popular dataset is experimented with to show the efficiency of the proposed framework. This database is the Amazon Kindle Store [57], which is also located in the K-Cores files of the Small Subset for Experimentation. Just as with the Movies and TV data, 7 columns were omitted, and the records with information older than the year 2010 were excluded. Accordingly, the preliminary dataset contains 2,219,176 rows with the five identical features shown in Table 1. Subsequently, the data cleaning process presented in Figure 7 was applied to this preliminary set, obtaining a final dataset of 2,216,008 rows with five features. Out of these data records, only 47,954 have a rating equal to 1, and therefore, a total of 235,000 records, 47,000 rows for every rating label, were randomly chosen to form dataset B.

The two best quality inspectors, BERT-2 and LSTM-5, are rebuilt using dataset B of the Kindle Store. Afterward, these two are executed to prune and clean, with k set to 1, the initial dataset B denoted as $DATA_{init.-all}$ (235,000 records). The output of LSTM-5, $DATA_{pruned\ LSTM5}$, consists of 214,166 records (91.13% of 235,000), and the output of BERT-2, $DATA_{pruned\ BERT2}$, has a total of 202,124 records (86.01% of 235,000). Finally, the different four recommender systems are tested with these three databases. The outcomes shown in Table 9 illustrate similar trends with the Movies and TV dataset in which NNMF, DLbR, SVD++, and SVD, rank in the first, second, third, and fourth positions, respectively.

Table 9. Results achieved per each model with three datasets generated from Kindle Store.

		NNMF	DLbR	SVD	SVD++
DATAinit.-all	RMSE	0.097	0.792	0.832	0.844
	F1 score	99.86%	88.58%	28.83%	37.63%
	accuracy	99.89%	89.80%	66.71%	69.17%
	time (min)	10.67	10.75	1.46	1.92
DATApruned LSTM5	RMSE	0.092	0.695	0.818	0.827
	F1 score	99.87%	89.98%	29.91%	39.50%
	accuracy	99.90%	91.04%	70.00%	72.43%
	time (min)	9.03	8.58	1.21	1.69
DATApruned BERT2	RMSE	0.079	0.703	0.826	0.834
	F1 score	99.89%	90.94%	31.60%	40.77%
	accuracy	99.91%	91.91%	68.39%	70.94%
	time (min)	10.00	8.32	1.15	1.58

Moreover, Figure 14 reveals comparable tendencies with the Movies and TV database in which applying a quality inspector impacts all recommendation metrics positively, and in which the results of the BERT quality inspector outperform the improvements of the LSTM quality inspector in almost all the recommendation indicators.

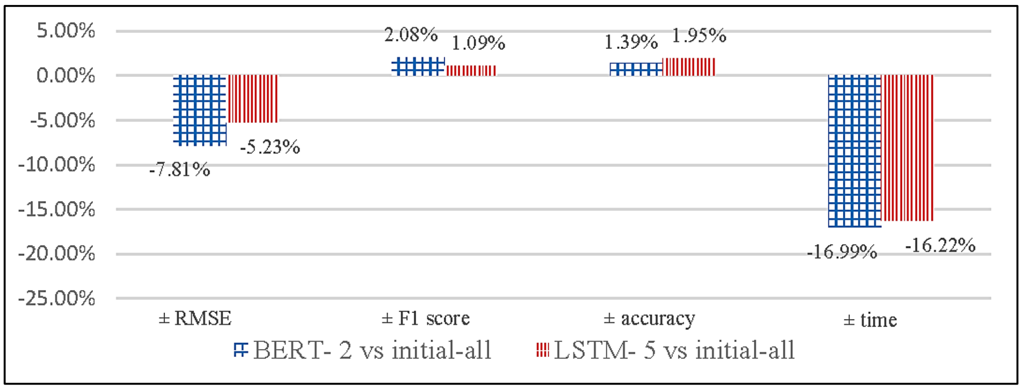

Figure 14. Percent variations averages experienced by the four models, per indicator, and for every quality inspector with Kindle Store data.

5. Discussion

After conducting several experiments, it can be concluded that the application of a data quality inspector, based on textual reviews, truly enhances the overall performance of a collaborative filtering recommender system. When exploiting the BERT quality inspector in two datasets, on average, it was registered a 17% decrease in the total time to run the algorithm, a 6% percentage enhancement in RMSE, and a percentage boost of between 2% and 3% in F1 score and accuracy. To assess the statistical significance of the obtained results, four recommender systems were run one time within six different data subsets: three generated from the Movies and TV data, and the other three from the Kindle Store data. The results show similar trends and comparable improvements when using the datasets pruned by LSTM5 and BERT2. The standard deviations (SD) of the percent improvements experienced by the four models when using the two subsets of $DATA_{pruned\ LSTM5}$ (produced from Movies and TV and from the Kindle Store), are an SD of 3.4% in RMSE, 1.1% in F1 score, 1.2% on accuracy, and 4.1% in total running time. Similarly, the SD of the percent improvements experienced by the four models when using the two subsets of $DATA_{pruned\ BERT2}$, are an SD of 5.9% in RMSE, 1.6% in F1 score, 1.4% on accuracy, and 5.1% in total running time.

Since the experiments were run within two datasets, it is robust evidence to prove the importance of the data quality inspection process, data cleaning, and the use of textual reviews as auxiliary data in the field of recommenders. Moreover, it can be determined that the five deep learning-based sentiment predictors (LSTM, Bi-LSTM, CNN, TDVE, and BERT) outperform the traditional machine learning algorithm (SVM). Similarly, it can be found that the two deep learning-based recommenders (NNMF and DLbR) exceed the performance of the two traditional machine learning methods (SVD and SVD++). Unquestionably, the deep learning approach is very suitable for these two kinds of tasks. Furthermore, comparing the results between the two deep learning-based recommenders, it is important to emphasize the leverage power of using a hybrid strategy. It was shown that NNMF, which is a mixture of one approach and two techniques, achieves better recommendation outcomes than DLbR, which is based only on MLP.

6. Conclusions

This work addresses one of the core and current challenges in a collaborative filtering recommender: accuracy enhancement. However, instead of specializing in the common cold-start or sparsity problems, it focuses on the quality of the ratings as input data, a critical issue highly correlated to the recommendation accuracy, which receives little attention in the research field. To our knowledge, this is the first study that truly filters or cleans the input ratings based on their quality. The authors of [15–17] took a similar approach, but they only preprocessed the ratings and used the complete records as input. It was demonstrated that working towards this problem promises encouraging results and that the application of hybrid recommenders and deep learning techniques to (a) model the key factor of recommenders and (b) extract features from auxiliary data, to integrate them with the input, are potent remedies to counteract the already mentioned difficulty.

In the future, more attention to the data quality inspection process and data cleaning of the input should be placed, and not only through the use of textual reviews, but with other methods or auxiliary data that does not exclusively depend on the users. For example, mouse clicks, triggered events, or eye-tracking should be helpful auxiliary sources for assisting data quality inspection. Furthermore, research related to the integration of multi-deep learning algorithms to build hybrid recommenders should be interesting. It is suggested to not only try hybrid recommenders with one approach and two techniques, but also with a mixture of two strategies, like collaborative filtering with content filtering or with context-aware recommenders. Finally, popular ranking metrics for top-N lists such as MRR (Mean Reciprocal Rank), MAP (Mean Average Precision), and NDCG (Normalized Discounted Cumulative Gain) should be applied to validate the performance of the proposed framework.

Author Contributions: Conceptualization, C.-Y.T.; Formal analysis, W.L.L. and M.-L.L.; Methodology, C.-Y.T. and W.L.L.; Software, M.-L.L. and W.L.L.; Validation, W.L.L. and M.-L.L.; Visualization, W.L.L.; Writing—review & editing, C.-Y.T. and W.L.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the Ministry of Science and Technology of Taiwan, R.O.C., No. MOST 108-2221-E-155 -005 -MY3.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available online at https://jmcauley.ucsd.edu/data/amazon/ (accessed on 15 September 2021).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Çano, E.; Morisio, M. Hybrid recommender systems: A systematic literature review. *Intell. Data Anal.* **2017**, *21*, 487–1524. [CrossRef]
2. Zdziebko, T.; Sulikowski, P. Monitoring human website interactions for online stores. *Adv. Intell. Syst. Comput.* **2015**, *354*, 375–384. [CrossRef]

3. Zins, A.-H.; Bauernfeind, U. Explaining online purchase planning experiences with recommender websites. In Proceedings of the 12th International Conference on Information and Communication Technologies in Travel and Tourism, Innsbruck, Austria, 26–28 June 2005; pp. 137–148. [CrossRef]
4. Tsai, C.-Y.; Chiu, Y.-F.; Chen, Y.-J. A Two-Stage Neural Network-Based Cold Start Item Recommender. *Appl. Sci.* **2021**, *11*, 4243. [CrossRef]
5. Maslowska, E.; Segijn, C.-M.; Vakeel, K.-A.; Viswanathan, V. How consumers attend to online reviews: An eye-tracking and network analysis approach. *Int. J. Advert.* **2020**, *39*, 282–306. [CrossRef]
6. Sulikowski, P.; Zdziebko, T.; Coussement, K.; Dyczkowski, K.; Kluza, K.; Sachpazidu-Wójcicka, K. Gaze and Event Tracking for Evaluation of Recommendation-Driven Purchase. *Sensors* **2021**, *21*, 1381. [CrossRef]
7. Park, D.; Kim, H.; Choi, I.; Kim, J. A literature review and classification of recommender systems research. *Expert Syst. Appl.* **2012**, *39*, 10059–10072. [CrossRef]
8. Liu, D.; Li, J.; Du, B.; Chang, J.; Gao, R.; Wu, Y. A hybrid neural network approach to combine textual information and rating information for item recommendation. *Knowl. Inf. Syst.* **2021**, *63*, 621–646. [CrossRef]
9. Roozbahani, Z.; Rezaeenour, J.; Emamgholizadeh, H.; Bidgoly, A.J. A systematic survey on collaborator finding systems in scientific social networks. *Knowl. Inf. Syst.* **2020**, *62*, 3837–3879. [CrossRef]
10. Bobadilla, J.; Ortega, F.; Hernando, A.; Gutiérrez, A. Recommender systems survey. *Knowl.-Based Syst.* **2013**, *46*, 109–132. [CrossRef]
11. Chen, L.; Chen, G.; Wang, F. Recommender systems based on user reviews: The state of the art. *User Model. User-Adapt. Interact.* **2015**, *25*, 99–154. [CrossRef]
12. Kumar, B.; Sharma, N. Approaches, issues and challenges in recommender systems: A systematic review. *Indian J. Sci. Technol.* **2016**, *9*, 1–12. [CrossRef]
13. Elahi, M.; Ricci, F.; Rubens, N. A survey of active learning in collaborative filtering recommender systems. *Comput. Sci. Rev.* **2016**, *20*, 29–50. [CrossRef]
14. Batmaz, Z.; Yurekli, A.; Bilge, A.; Kaleli, C. A review on deep learning for recommender systems: Challenges and remedies. *Artif. Intell. Rev.* **2019**, *52*, 1–37. [CrossRef]
15. Raghavan, S.; Gunasekar, S.; Ghosh, J. Review quality aware collaborative filtering. In Proceedings of the 6th ACM Conference on Recommender Systems, Dublin, Ireland, 9–13 September 2012; pp. 123–130. [CrossRef]
16. Pero, Š.; Horváth, T. Opinion-driven matrix factorization for rating prediction. In Proceedings of the 21st International Conference on User Modeling, Adaptation, and Personalization, Rome, Italy, 10–14 June 2013; pp. 1–13. [CrossRef]
17. Dang, C.-N.; Moreno-García, M.-N.; Prieta, F.D.L. An Approach to Integrating Sentiment Analysis into Recommender Systems. *Sensors* **2021**, *21*, 5666. [CrossRef] [PubMed]
18. Sedhain, S.; Menon, A.; Sanner, S.; Xie, L. Autorec: Autoencoders meet collaborative filtering. In Proceedings of the 24th International Conference on World Wide Web, Florence, Italy, 18–22 May 2015; pp. 111–112. [CrossRef]
19. Zheng, Y.; Tang, B.; Ding, W.; Zhou, H. A neural autoregressive approach to collaborative filtering. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 764–773.
20. Du, C.; Li, C.; Zheng, Y.; Zhu, J.; Liu, C.; Zhou, H.; Zhang, B. Collaborative filtering with user-item co-autoregressive models. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; pp. 2175–2182.
21. He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; Chua, T. Neural Collaborative Filtering. In Proceedings of the 26th International Conference on World Wide Web, Perth, Australia, 3–7 April 2017; pp. 173–182. [CrossRef]
22. Cambria, E.; Schuller, B.; Xia, Y.; Havasi, C. New avenues in opinion mining and sentiment analysis. *IEEE Intell. Syst.* **2013**, *28*, 15–21. [CrossRef]
23. Collobert, R. Deep learning for efficient discriminative parsing. In Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, Ft. Lauderdale, FL, USA, 11–13 April 2011; pp. 224–232.
24. Devipriya, K.; Prabha, D.; Pirya, V.; Sudhakar, S. Deep learning sentiment analysis for recommendations in social applications. *Int. J. Sci. Technol. Res.* **2020**, *9*, 3812–3815.
25. Kim, Y. Convolutional neural networks for sentence classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, Doha, Qatar, 25–29 October 2014; pp. 1746–1751. [CrossRef]
26. Xue, B.; Fu, C.; Shaobin, Z. A study on sentiment computing and classification of sina weibo with word2vec. In Proceedings of the 2014 IEEE International Congress on Big Data, Anchorage, AK, USA, 27 June–2 July 2014; pp. 358–363. [CrossRef]
27. Dos Santos, C.; Xiang, B.; Zhou, B. Classifying relations by ranking with convolutional neural networks. In Proceedings of the 53rd Annual Meeting of the ACL and the 7th International Joint Conference on Natural Language Processing, Beijing, China, 26–31 July 2015; pp. 626–634. [CrossRef]
28. Zhang, Z.; Hu, Z.; Yang, H.; Zhu, R.; Zuo, D. Factorization machines and deep views-based co-training for improving answer quality prediction in online health expert question-answering services. *J. Biomed. Inform.* **2018**, *87*, 21–36. [CrossRef]
29. Bahdanau, D.; Cho, K.; Bengio, Y. Neural machine translation by jointly learning to align and translate. In Proceedings of the International Conference on Learning Representations ICLR, Banff, AB, Canada, 14–16 April 2014; pp. 1–15.

30. Yang, Z.; Yang, D.; Dyer, C.; He, X.; Smola, A.; Hovy, E. Hierarchical attention networks for document classification. In Proceedings of the 2016 Conference of the North American Chapter of the ACL: Human Language Technologies, San Diego, CA, USA, 12–17 June 2016; pp. 1480–1489. [CrossRef]
31. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.; Kaiser, L.; Polosukhin, I. Attention is all you need. In Proceedings of the 31st Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 6000–6010.
32. Devlin, J.; Chang, M.; Lee, K.; Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Minneapolis, MN, USA, 3–7 June 2019; pp. 4171–4186. [CrossRef]
33. Kumar, S.; De, K.; Roy, P.P. Movie recommendation system using sentiment analysis from microblogging data. *IEEE Trans. Comput. Soc. Syst.* **2020**, *7*, 915–923. [CrossRef]
34. Su, X.; Khoshgoftaar, T. A survey of collaborative filtering techniques. *Adv. Artif. Intell.* **2009**, *2009*, 1–20. [CrossRef]
35. Barriere, V.; Kembellec, G. Short review of sentiment-based recommender systems. In Proceedings of the 1st ACM International Digital Tools & Uses Congress, Paris, France, 3–5 October 2018; pp. 1–4. [CrossRef]
36. Leung, C.; Chan, S.; Chung, F. Integrating collaborative filtering and sentiment analysis: A rating inference approach. In Proceedings of the ECAI 2006 Workshop on Recommender Systems, Riva del Garda, Italy, 28–29 August 2006; pp. 62–66.
37. Zhang, W.; Ding, G.; Chen, L.; Li, C.; Zhang, C. Generating virtual ratings from Chinese reviews to augment online recommendations. *Trans. Intell. Syst. Technol.* **2013**, *4*, 1–17. [CrossRef]
38. Poirier, D.; Fessant, F.; Tellier, I. Reducing the cold-start problem in content recommendation through opinion classification. In Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, Toronto, ON, Canada, 31 August–3 September 2010; pp. 204–207. [CrossRef]
39. Ozsoy, M.-G. From word embeddings to item recommendation. *arXiv* **2016**, arXiv:1601.01356.
40. Ozsoy, M.-G. Utilizing fasttext for venue recommendation. *arXiv* **2020**, arXiv:2005.12982.
41. Alexandridis, G.; Siolas, G.; Stafylopatis, A. ParVecMF: A paragraph vector-based matrix factorization recommender system. *arXiv* **2017**, arXiv:1706.07513.
42. Alexandridis, G.; Tagaris, T.; Siolas, G.; Stafylopatis, A. From free-text user reviews to product recommendation using paragraph vectors and matrix factorization. In Proceedings of the 2019 World Wide Web Conference, San Francisco, CA, USA, 13–17 May 2019; pp. 335–343. [CrossRef]
43. Shin, D.; Cetintas, S.; Lee, K.; Dhillon, I. Tumblr blog recommendation with boosted inductive matrix completion. In Proceedings of the 24th ACM International Conference on Information and Knowledge Management, Melbourne, Australia, 18–23 October 2015; pp. 203–212. [CrossRef]
44. Shen, X.; Yi, B.; Zhang, Z.; Shu, J.; Liu, H. Automatic recommendation technology for learning resources with convolutional neural network. In Proceedings of the International Symposium on Educational Technology, Beijing, China, 19–21 July 2016; pp. 30–34. [CrossRef]
45. Wei, J.; He, J.; Chen, K.; Zhou, Y.; Tang, Z. Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Syst. Appl.* **2017**, *69*, 29–39. [CrossRef]
46. Seo, S.; Huang, J.; Yang, H.; Liu, Y. Interpretable convolutional neural networks with dual local and global attention for review rating prediction. In Proceedings of the Eleventh ACM Conference on Recommender Systems, Como, Italy, 27–31 August 2017; pp. 297–305. [CrossRef]
47. Garcia, A.; Gonzalez, R.; Onoro, D.; Niepert, M.; Li, H. TransRev: Modeling reviews as translations from users to items. *Adv. Inf. Retr.* **2020**, *12035*, 234–248. [CrossRef]
48. GeeksforGeeks. Available online: https://www.geeksforgeeks.org/removing-stop-words-nltk-python/ (accessed on 22 May 2017).
49. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
50. Colah's Blog. Available online: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed on 27 August 2015).
51. Schuster, M.; Paliwal, K. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.* **1997**, *45*, 2673–2681. [CrossRef]
52. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
53. The Data Science Blog. Available online: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ (accessed on 11 August 2016).
54. Towards Data Science. Available online: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270 (accessed on 11 November 2018).
55. Towards Machine Learning. Available online: https://towardsml.com/2019/09/17/bert-explained-a-complete-guide-with-theory-and-tutorial/ (accessed on 17 September 2019).
56. Experiments in Data Science. Available online: http://blog.richardweiss.org/2016/09/25/movie-embeddings.html (accessed on 26 September 2016).
57. Ni, J.; Li, J.; McAuley, J. Justifying recommendations using distantly-labeled reviews and fined-grained aspects. In Proceedings of the Annual Conference on EMNLP, Hong Kong, China, 3–7 November 2019. [CrossRef]