Fast Sentence Classification using Word Co-occurrence Graphs*

Ashirbad Mishra
Dept. of C.S.E
Pennsylvania State University
University Park, PA, USA
amishra@psu.edu

Shad Kirmani
LinkedIn Corp.
Sunnyvale, CA, USA
skirmani@linkedin.com

Kamesh Madduri
Dept. of C.S.E
Pennsylvania State University
University Park, PA, USA
madduri@psu.edu

2024 IEEE International Conference on Big Data (BigData) | 979-8-3503-6248-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/BigData62323.2024.10825869

***Abstract*—We consider a supervised classification problem of categorizing e-commerce products based on just the words in the title. If done in real-time, the categorization can greatly benefit sellers by enabling them to offer immediate feedback. We present a deterministic algorithm by constructing weighted word co-occurrence graphs from the listing/item titles. We empirically evaluate this algorithm on two publicly available product listing datasets, Etsy and Amazon. Our method's accuracy is comparable to that of a supervised classifier constructed using the fastText library. The inference time of our model is up to $2.9\times$ faster than the fastText classifier and has small training times. The training and inference of our model scales well for big datasets performing large-scale classification on millions of listings. We perform a detailed analysis and provide insights into our method and the product categorization task.**

***Index Terms*—Sentence classification, word co-occurrence graphs, e-commerce product categorization, large-scale classification.**

I. Introduction

Automated product categorization in marketplaces such as Amazon, Etsy, and eBay, is of interest to sellers. As the sellers may upload information corresponding to multiple listings, and may not be aware of the category tree the marketplace is using. It may be difficult for sellers to manually enter the appropriate category for each listing. Another related application is detecting fraud. To sell certain electronic products such as software licenses, sellers may intentionally classify the listing to be in an unrelated category (e.g., souvenirs), in order to evade checks based on price and other information.

Consider the following scenario; while a seller is setting up their inventory, as the first step the listing requires the item's title. Upon entering the title, the seller can be shown a small subset of categories to choose from depending on the title. If the seller doesn't choose any option then the top-ranking recommendation can be used. While providing suggestions to the seller, this process can also limit fraud by restricting the seller's choices. Such predictions need to be done in real time so as to not hinder the seller's experience of inventory setup.

A transparent method can help improve subsequent fraud detection by finding its root causes in the model's decision process and/or misinformation in the data. For instance, a method can show how certain words in an item's title can lead to miscategorization of the listing. We focus on listing titles only and do not use other information provided with listings, to assess how well we can classify without using auxiliary information. We formulate the problem of classifying short sentences, where the inputs are sentences (short text) corresponding to the titles of products.

The predominant approach to solving the sentence classification problem is to learn word vectors and build a classifier. However, such methods typically lack interpretability, i.e., it is difficult to ascertain why a sentence was correctly or incorrectly classified. Another issue is that Large Language Models (LLMs) like BERT, GPT, and others have high training and inference times [1], [2]. Thus their predictions are difficult to serve in real-time and would require a weekly batched categorization. Also, the inferences generally require expensive hardware like GPUs and TPUs which aren't guaranteed to be available at the time of inferencing.[1] Such resource-heavy models conflict with the goals of facilitating real-time or high-throughput applications. The models have difficulty scaling on very large datasets with hundreds of millions of inputs and are problematic for latency-sensitive applications [3], [4]. Frequent model re-training (or model refresh) is a crucial goal that can include newer data points with many products being listed regularly.

In this work, we develop an alternate method that does not require learning vector representations of words. Instead, we construct weighted word co-occurrence graphs and use these graphs directly for classifying listings. The primary advantage of this method is the improved interpretability. However, the burden shifts from learning to developing an appropriate classification strategy that makes effective use of the word co-occurrence graphs. Also, deterministic strategies can be implemented efficiently, thus achieving faster execution times. We present a principled approach to classification based on graph structure.

Another commonly used graph-based approach is to define a similarity measure to relate two sentences, compute similarities for all sentence pairs, use a threshold to filter low-similarity pairs, and then execute a graph-based semi-

*Supported by NSF grants 1955971 and 2120361.

[1]Due to sharing in cloud systems. A single GPU for an application is expensive.

supervised learning method. However, such a method would require quadratic work in the number of listings to compute pairwise similarities and is computationally expensive for data with millions of listings. Our word co-occurrence graph construction approach avoids this quadratic cost.

At a high level, our method works by constructing multiple weighted word co-occurrence graphs, one for each category. Given a new sentence/listing, we look for occurrences of the words in this sentence in each of the graphs and predict the category based on a scoring mechanism. We show that for two publicly available datasets with millions of listings (from Etsy and Amazon), our approach's veracity is at par with a classifier built using the fastText library. Our model is faster than fastText and also scales well for big datasets. Our method could thus be used to provide fast recommendations and simple post-hoc explanations using probabilistic methods and complex schemes. We also develop an ensemble classifier combining the fastText classifier with our classifier.

II. Related Work

Constructing word-occurrence graphs for information retrieval problems is an established and well-studied technique [5]. Recent graph-based approaches focus on generating quantified weights for each word (i.e., term weights), instead of using word frequencies, to signify the importance of each word in the text. The weights generated are based on in-degree [6], random walks [7], [8] or using centrality measures [9], [10]. In the end, a TF-IDF based scoring function [6]–[8] or a classifier such as SVM [10] is used to generate a score for each document or sentence which can be used for text classification or sentiment analysis. The initial ranking of vertices can also be used for tasks such as text summarization or keyword detection [9], [11]. These approaches indirectly encode the co-occurrence relations into weights associated with each word. In contrast, our approach uses the co-occurrence relations explicitly, which aids interpretability.

In recent years, models based on deep neural networks have emerged as the preferred technique for multi-task learning. Prior neural network models for single-sentence classification tasks include [12]–[14]. Due to the infeasibility of real-time or high throughput inference, LLMs are generally replaced with shallow perceptrons. To further mitigate inference times, these models typically employ techniques such as quantization, distillation, and pruning. Conversely, while simpler methods like Support Vector Machines (SVMs) and Decision Trees offer faster and more interpretable results, they generally exhibit lower accuracy. CPUs-only models are favorable as the cores/memory can be easily shared among applications while avoiding overheads like CPU-GPU transfer times.

We primarily compare our approach to a classifier built with the fastText library [15], specifically one using a *subword* model enhancement [16] of the skipgram model [17]. The subword approach learns the representation of character n-grams aggregating them to generate word embeddings. This helps in recognizing words not encountered in the training data. By mapping the contextual meaning of the unseen words to words in vocabulary improves fastText's performance. The fastText classifier's linear model is shown to be significantly faster than several deep neural network models for a variety of natural language processing tasks [18]. While pre-trained models such as BERT [19] provide embeddings with better representation [20], fastText is known to scale well for datasets with millions of inputs and even for millions of labels [21].

The *LinearSVC* package for Support Vector Machines (SVM) is an efficient implementation based on the *LIBLINEAR* [22] library. Its solver for the linear optimization problem utilizes the Coordinate Descent algorithm that iteratively performs approximate minimization along coordinate directions or hyperplanes. Thus, it scales wells for big datasets resulting in faster training times. The inference of SVM is a simple matrix (weights) and vector (feature) multiplication to determine its sign.

The Decision Tree model creates a tree structure where each node makes a binary decision on each input based on one or more features. The tree is grown starting with the root adding nodes one by one. Each node splits the training inputs into left or right groups. The goal is to improve the homogeneity such that each group belongs to the same class. We use the entropy loss which measures the amount of training inputs in each leaf that belong to different classes. Leaves are tagged with the true class of the majority of the inputs that follow the path from the root to that leaf. Similarly, during inference, an input follows the path from the root until it reaches a leaf tagged with a single class thereby classifying the input.

The Random Forest technique builds multiple decision trees as a forest where each tree is also built from a random subset of the training dataset. During inference, each tree predicts one class for each input, and voting on the outputs of multiple trees determines the final recommendation.

In general, works on e-commerce product categorization look at assigning multi-level categories based on a taxonomy tree [23]–[25]. As in this work, we focus on categorizing listings at a single level in such a tree, where prior work would not be directly applicable. While these models use neural networks to perform text classification, we do direct comparisons with fastText and above mentioned models along with BERT. Prior methods on categorizing e-commerce listings in supervised settings include [26]–[28]. Another related problem is sentiment analysis of sentences and paragraphs [29]–[31].

III. Our Graph-based Approach

Suppose we are given s sentences belonging to one of k categories. Let the number of unique words/tokens after preprocessing be n, the sum of the lengths of each sentence be r, and the sum of the squares of the lengths of each sentence be r'. The average number of words per sentence is thus r/s. Suppose we map each unique word to a 4-byte integer and also store the category of each sentence as a 4-byte integer. Thus, the input size is $4(s+r)$ bytes.

A. Constructing Word Co-occurrence Graphs

Consider processing sentences one at a time. For a sentence i of length l_i belonging to category j, we generate two triples for each pair of words (say, w_1 and w_2) in this sentence: $\langle j, w_1, w_2\rangle$ and $\langle j, w_2, w_1\rangle$. The total number of triples we generate for a sentence is $l_i(l_i - 1)$, representing a clique with the words as vertices and co-occurrences as edges. If the triples are stored in a contiguous array, we can sort this array using the concatenated fields as the key and then deduplicate this array to get the co-occurrence count of each word pair in a category. Assuming 4-byte integers, the array before sorting is of size $6(r' - r)$ bytes. Using the sorted array, we can construct k weighted graphs, one for each category. Suppose the number of edges (word pair co-occurrences) in each category is m_j. Then, the space required to store all graphs in the *compressed sparse row* (CSR) sparse matrix storage format is $\sum_{j=1}^{k} 4(2m_j + n + 1)$ bytes, or $8m + 4nk + 4k$ bytes (defining m to be $\sum_{j=1}^{k} m_j$).

The *model size* is characterized by the value of m, which is input-dependent. A mathematically equivalent viewpoint is that for each of the k categories, we create a matrix of dimensions $n \times n$, with the entry in row i and column j denoting the number of co-occurrences of words i and j. Since we expect these matrices to be sparse, we use a sparse matrix storage format. For the weighted graphs, a vertex weight is defined as the number of occurrences of a word, whereas an edge weight is the number of co-occurrence of each word pair in the training data. These weights are computed per-category and denoted as local weights. In contrast, the global weights are frequencies across all categories in the training set.

Figure 1 shows the graphs constructed for a toy example (two categories with three sentences each). *Category 1* is similar to a category for "Cellphones" while *Category 2* is similar to a category for "Cellphone Accessories". $k = 2, n = 11, m = 19, r = 18$. Tokenized words are given unique non-negative integer identifiers across all categories. These identifiers are the vertices of the graphs for each category. An edge is marked between two vertices in a category graph if they co-occur in the same sentence for that category.

Category 1:
Apple^0 iPhone^1 12^2
new^3 Apple^0 iPhone^1 XS^4
Apple^0 phone^5

Category 2:
new^3 iPhone^1 canvas^6
wall^7 art^8
stylish^9 phone^5 art^8 (used)^{10}

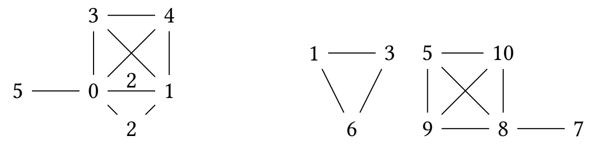


Fig. 1: Example word co-occurrence graphs. Superscripts indicate word to vertex integer identifier mapping. Unit weight edges are not labeled.

B. Classification

Given a new sentence, we want to use the k weighted co-occurrence graphs/matrices and classify this sentence. We first describe our rule-based classification technique and then provide some intuition behind its design. Suppose the sentence we wish to classify is of length l. We form an unweighted clique using words in the sentence. We then check each of the k graphs for the existence of $l(l-1)/2$ edges (or word pair occurrences). For each category, we store the matched edges and then assign an integer score between 0 and $l(l-1)/2$, corresponding to the number of occurrences of these edges. If there is no tie for the category with the highest value, then the unique highest-value category is returned as the prediction. In case of a tie and a non-zero highest value, we consider a second rule, given by the sum of the logarithm of the non-zero local weights of the matched edges (or equivalently, the *product of non-zero local edge weights*) minus the logarithm of the number of listings in the category. As a third rule, we consider the sum of the logarithm of non-zero local *vertex weights* in each category.

C. Illustration

Before understanding the significance behind our rule-based let's look an example to better under the classification process. Let's take a test listing title "*new phone canvas*" to classify into one of the categories in Figure 1. Ideally, the item should belong to the "Cellphone Accessories" category (Category 2). The first step is to map the words to their IDs, which are 3, 5, and 6 for the words new, phone, and canvas respectively. We first construct a clique using the words in the title which gives us three edges $(3, 5)$, $(5, 6)$ and $(3, 6)$ as shown in Figure 2. To apply the first rule we match the three edges with the graphs in Category 1 and Category 2. There is only one edge match $(3, 6)$ with
matches. Thus C

Test titl
new^3 ph
3
5
6

Fig. 2: Example test listing's title and the clique corresponding to the words in the title.

D. Discussion on Rules

In the example described in the previous section, only the first rule (number of matching edges) was used. In order to come up with the above scheme, we first experimented with single rules. We considered sums, products, and ratios of vertex and edge weights with various normalization schemes. Many of these rules have an underlying probabilistic interpretation. Our experimentation revealed that using the unweighted edge existence count outperformed other single-rule schemes. This is because it is challenging to normalize weights. In fact, as shown in our analysis in section IV-F1 the first rule is sufficient to classify the majority of the listings in the datasets that we consider.

For the remaining set of listings where there is a tie in the number of edges matched. We then considered a second rule to break ties. The second rule is based on the conditional probability of finding a listing given a category, $P(listing|category) = P(listing \wedge category)/P(category)$. The listing can be viewed as a group of words or a group of word occurrences (edges) thus the probability of a listing occurring in a category can be approximated as a probability of the listing's words co-occurring in the category. Therefore, the term $P(listing \wedge category)$ can be computed as the product of the count of edge weights in the category to the product of the count of edge weights across all categories (assuming conditional independence). If we apply logarithm to the equation above we get our second rule the log of the sum of edge weights minus the log of the number of listings in the category (probability of the category) with an additional term. Thus the logarithm of the sum (or product) operation as described above can be viewed as taking joint probabilities where each probability is the ratio of local weight to the global weight. The additional term is the negative of the logarithm of global weights or (denominator product of weights) is common for all categories which can be ignored. Please note that the global weights are used for illustration and we do not store them.

Here, considering the sum of the logarithms of the weights performed better than taking the product of the weights. This may be due to the distribution of edge weights in a category, and applying the log operation lowers the impact of the very high-frequency co-occurrences on the eventual selection. Another benefit is that the product is a large value requiring more bits to store increasing execution time. The third filter using vertex weights is a fallback option in case there are no edge matches, and thus all edge weights are zero. Other heuristics are possible (which we discuss in the empirical evaluation section), but in order to aid interpretability, we wanted to the keep the method as simple as possible.

The per-query cost is proportional to kl^2 because we require $O(l^2)$ edge lookups in k graphs. Since adjacencies of a vertex are stored as a sorted array in the CSR format, we can use binary search. If the average adjacency length is d, then the edge lookup cost is proportional to $O(\log d)$.

E. Modified Graph Construction

In the graph construction described in Section III-A, the word order in a sentence does not affect the undirected graph structure. Further, all word pairs are given equal weight. Also, generating $O(l^2)$ word pairs may be expensive for large l. To address these issues, we modify the graph construction to use a *sliding window* approach. Consider a window of length $w \leq l$. We generate tuples corresponding to word pairs in this window and then slide the window to the right by one unit. The number of tuples generated per listing is $(l - w + 1)w(w - 1)$. When $w = l$, this scheme is the same as the original scheme. When $w = 2$, only $l - 1$ edges are generated per listing and adjacent words are given more importance. The value of w could be set based on the maximum listing length and the main memory capacity of the system.

An alternative would be to construct the graphs as described in Section III-A and then remove vertices and edges with weights lower than some thresholds. This change will also *sparsify* the graphs.

F. Implementation Details

We develop a C++ implementation using OpenMP for thread-based parallelization. The sorting phase is the dominant compute-intensive step in graph construction. We use a coarse-grained parallelization approach to graph construction, where each thread constructs a graph corresponding to one category after the sort. A fine-grained approach of multiple threads constructing a graph in parallel is left for future work. An alternative to the sort, deduplication, and graph construction is to use k hash tables and not explicitly construct graph structures. Another alternative is to construct just one graph, but have a dense k-dimensional vector associated with each edge. Although such representation might require more memory, the classification phase could be faster due to fewer edge lookups. We also parallelize the classification step with threads working independently on different test sentences to compute rule values for all categories and ranking them.

G. Avoiding quadratic-cost graph construction

We construct graphs where words, and not listing titles, are the vertices. This approach averts the quadratic complexity associated with *all-pairs* similarity calculations, which in turn reduces the time taken for training phase. The sliding window construction adds the window size as a tuning parameter and the optimal size depends on the importance of word order in a sentence and distribution of the sentence lengths in a dataset. Even though a window of size two can be restrictive, it can be useful if we were to use a directed graph representation and thereby consider word order. The graph structure is chosen assuming k is in the range of 20 to 50. For larger values of k, a hash table-based representation may be more appropriate.

H. Interpretability

Neural network models generally use interpretable linear models to generate post-hoc explanations for predictions. The interpretation models compare the difference in their outputs based on input variation, which shows the importance of the words in the input. Our approach can serve a similar purpose. We can examine the true and mispredicted categories by comparing the edge counts and weights to determine possible co-occurrences that resulted in the misclassification. Section IV-F2 shows a way of such interpretation with a test example.

I. Unseen Words

It is possible that new words that do not occur in the training data appear in the test data. Neural network models incorporate *subword* information to generate word vectors, which enables matching unseen words to word in vocabulary. While the unseen word problem was not a significant issue for the datasets we consider, we also implement a simple prefix- and suffix-based matching scheme to match unseen words to words in the vocabulary. Given an unseen word of size w

characters that is not present in our vocabulary, we check for the prefix and suffix of size $w - k$ in the vocabulary (for a small k). If a successful match is found, the corresponding word is used as a replacement for the unseen word.

J. Alternate Rules

We considered replacing the edges matched rule with other connectivity-related topological features that do not rely on weights, such as connected components, largest clique size, and number of triangles. In the ideal scenario, the true category of a test sentence would have the maximum possible value for clique size, the number of triangles, and the there would just be one connected component in the induced subgraph. Alternately, the largest (or smallest) value of a graph property can be used as the deciding factor. Each property can be broken down into the following cases (assuming maximization). *Case 1*: The true category of the test sample has the maximum possible value for the property, while all other categories have a lower value. *Case 2*: The true category has a value lower than max value, while some other category has a larger value. *Case 3*: None of the categories have the maximum value, but the true category does not have the highest value. *Case 4*: None of the categories have the max value, but the true category still has the largest value. *Case 5*: There are multiple categories with contending values. Cases 1 and 4 correspond to correct predictions, while Cases 2 and 3 pertain to misprediction. In case 5, there are multiple categories and it is difficult to distinguish between them without a second rule. The evaluations are shown in Section IV-F1 for each scenario.

K. Ensemble Strategy

Typically, an ensemble of similar models is formed based on some criteria such as majority agreement among all predictions. Other factors, like the confidence level in the model's predictions, can also be used to select the prediction with the highest confidence. However, since our model does not provide a confidence level, majority voting serves as the primary strategy.

Luckily, since our model is transparent we can somewhat determine the scenarios where mispredictions will occur. If we can reliably identify such scenarios (of low confidence), we can preemptively switch the recommendation to another model, which might be more accurate. Our analysis in section IV-C attempts to determine the scenarios where our model has a tendency to misclassify. Based on the analysis we devised a strategy for an ensemble of our model and fastText which is described in section IV-C5 determined based on this analysis.

IV. Evaluation

A. Test Setup

We use two large publicly available e-commerce datasets to evaluate our method. The first dataset contains listing metadata for products sold on Etsy in March 2015 [15]. The second dataset has a sampling of reviews for products sold on Amazon [32]. For both the datasets, we extract the product/listing title and the top-level category for the listing. We process titles by keeping all ASCII characters except spaces and full stop, remove tokens of unit length, and convert HTML codes to corresponding characters wherever possible. Further, we remove words present in the NLTK [33] stop word list and keep only the first occurrence of a repeating word. We also remove categories with a relatively low listings/product count, and remove listings of length less than six.

Using the accuracy metric (i.e., percentage of test sentences correctly classified), we compare our method primarily to fastText. We additionally show results with Recall@k for different k values. Following a linear search for the hyperparameters, we found the following values to work best with fastText: 15 training epochs, learning rate of 0.1, n-gram value of 2, subword size between 3-6, loss function "ova", and word vector dimension of 128.

The running time results are reported on a dual-socket Intel server with 2.8 GHz Intel Xeon E5-2650 v4 processors. Each socket has 12 cores and the server has 126 GB DDR4 memory.

Additionally, we compare our approach to BERT [19] and DistilBERT [34] using the *huggingface* [35] framework with pre-trained models *bert-base-uncased* and *distilbert-base-uncased* respectively. The BERT models are run on a Nvidia Tesla V100 GPU with 32 GB of memory. We use a large batch size of 90 to speed up the training along with 5 epochs and 4e-05 learning rate.

We also compare to a linear Support Vector Machine (SVM) classifier implementation with TF-IDF vectorization enabled through *LinearSVC* [22] classifier of *scikit-learn* [36] that runs on multi-cores. Similarly, we also compare with two other models from scikit-learn; *DecisionTreeClassifier* using the entropy loss function and *RandomForestClassifier* with 50 estimator trees using the above vectorization scheme. The max depth parameter is not set to improve the accuracy of the tree models.

B. Dataset Characteristics and Summary

We show results when creating the vocabulary from all words in the training dataset, which includes tokens with alphanumeric and punctuation characters. We perform an 80-20 split of the datasets after randomly shuffling the input. More details of the 80% split used as training data are given in Table I. We also consider other 80-20 splits of the original data as well as other splits of the data (50-50 and 90-10), but most of the results in this section are with the training data in Table I. The test data is also preprocessed in a manner similar to the training data.

The three most frequently occurring words in the Etsy training data are baby (100 583), hand (80 679), and art (78 037). For Amazon, the words are women's (355 194), case (311 281), and black (299 009). For the 80-20 split training data given in Table I, our method achieves an accuracy of **87.5%** for the Etsy dataset and **90.4%** for the Amazon dataset. With fastText, we get 87.4% for Etsy and 90.1% for Amazon. We achieve the best performance with a window

TABLE I: Training data after preprocessing.

Attribute	Value	
	Etsy	Amazon
# sentences s	1 360 826	6 348 428
# categories k	28	23
# vertices n	1 106 600	3 789 268
# edges (combined) m	22 219 516	92 002 197
# edges mean (Coeff. of Var.)	793 554 (0.6)	4 000 095 (0.9)
Sentence length l range	[6,23]	[6,29]
Input size in GB	0.05	0.24
Tuple array size in GB	1.3	6.5
Graph size in GB	0.28	1.01

size of 10 corresponding to the *sliding window* as discussed in section III-E. We use this window size for all our results.

C. Performances on Accuracy

In this section, we show the various analyses we conducted to compare the accuracy of our model with fastText.

1) Accuracy with different splits: We consider alternate 80-20 splits first. With nine additional splits, the standard deviations for accuracy with the Etsy dataset are 0.05 (our method) and 0.06 (fastText). For Amazon, the standard deviations are 0.04 (our method) and 0.03 (fastText). Thus, both methods show similar variation with alternate shuffling of the data, and the variation is low.

We next consider two alternate splits of the data, 50-50 and 90-10. These accuracy results, along with the 80-20 split above, are summarized in Table II. The results show a similar trend to the 80-20 split, with fastText slightly trailing in all cases. We do not report the standard deviation, but it is similar to 80-20 runs. As expected, the accuracy drops when there is a relatively smaller size of training to test data.

TABLE II: Accuracy with alternate splits.

Split	Etsy		Amazon	
	Our	fastText	Our	fastText
50-50	86.1 %	85.9%	89.5%	89.4%
80-20	87.5 %	87.4%	90.4%	90.1%
90-10	88.7 %	88.5%	90.5%	90.2%

2) Accuracy across sentence lengths: In Figure 3, we show the accuracy of both our method and fastText for different listing title lengths. The performance is somewhat higher for larger lengths and lower for smaller lengths on Etsy. This might be due to our method's primary focus on word co-occurrences, which might be limited on smaller length inputs. As smaller length inputs tend to form generic subgraphs which are more common and thus easier to misclassify into other categories unless the words are unique to the category. However, many of the listings have more than 8 tokens where our method performs superior to fastText.

On the other hand, our method performs better on Amazon on the smaller-length titles. This is due to a large number of unique titles or listings in each category providing more

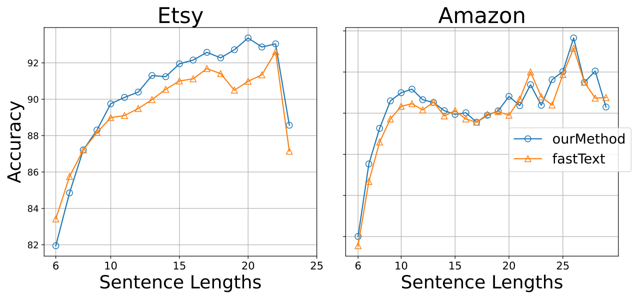


Fig. 3: Accuracy variation with sentence length.

unique words and co-occurrences per category. Listings of size >20 are quite small in amount as long-length titles can be distracting to buyers. Such a test listing will have an almost exact match or no matching training listing, leading to erratic performance (especially in Amazon).

3) Agreement of predictions: We also look at the percentage of listings for which our method and fastText give identical predictions. These results are summarized in Table III. Both methods agree for most of the correctly classified listings, while fastText performs slightly worse than our model on individual performance.

TABLE III: Prediction agreement results.

Prediction type	Etsy	Amazon
Both correct	82.4%	86.2%
Our method only	5.1%	4.2%
fastText only	5.0%	3.9%
Neither correct	4.7%	4.4%

For the use case defined in section I, where a seller has been presented with options to categorize during their inventory setup, the second or third-rank predictions are important. Figure 4 shows the recall scores for $R@k$ where k ranges from 1 to 10. The scores show whether the true label is among the k^{th} predictions. As expected both models scale well towards 100% prediction. While both models have similar recall scores for Etsy, our model performs slightly better for larger k. Whereas, in Amazon, our model performs noticeably better with increasing k. The performance gap is evident from the second-rank prediction which is crucial as it provides better choices to the seller.

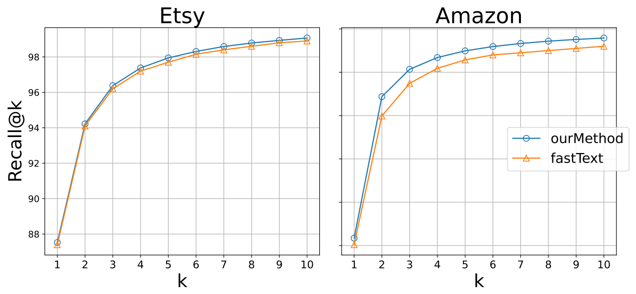
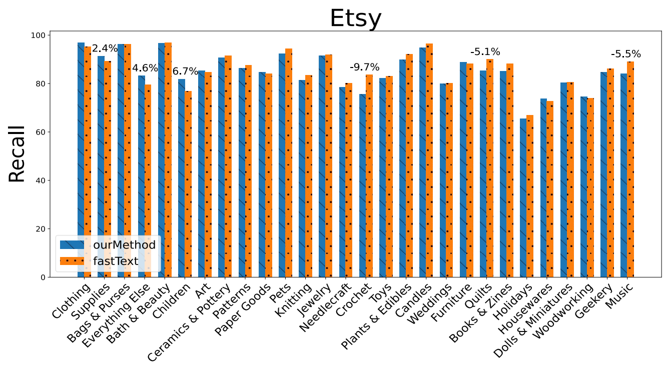
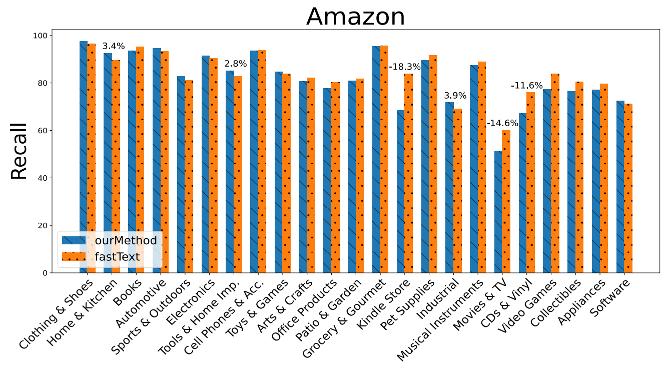


Fig. 4: Recall@k for $k = 1 - 10$ for both models.

4) Per-category Analysis: Next, we consider Recall@1 variation by category. Figure 5 shows the per-category recall scores for both datasets. In the x-axis, the categories are ordered left to right in the descending order of the number of listings belonging to each category. In both the datasets, fastText and our model, in general, both perform better for large-size categories than smaller ones due to more training data. The high accuracy of both models is primarily due to the large-size categories with many listings.



(a) Etsy dataset



(b) Amazon dataset

Fig. 5: Recall scores per category on our method and fastText. The percentage difference between both models is shown for the top 3 best and worst performing categories.

For our method, we look at mispredictions (with the 80-20 split) and list the top categories that were mispredicted as another category for both datasets in Table IV. The reasons for the dips in accuracy for our models for medium-sized categories "Needlecraft", "Crochet" and "Toys" shown in Figure 5 can be found in the table. Such as the category "Crochet" has some similar listings to "Clothing" leading to mispredictions into that category.

There seems to be some correlation between "Sports & Outdoors" and "Clothing & Shoes" for Amazon. There are relatively higher mispredictions of the "Sports & Outdoors" category which are classified as "Clothing & Shoes" than the other way round. Since both the categories are quite large, these are the majority of the mispredictions for those categories leading to their higher accuracy as seen in Figure 5b. Equivalently, "Kindle Store" is generally mispredicted (especially by our model) into "Books" due to their similar nature. Special post-processing can be performed on such co-related categories to minimize mispredictions.

TABLE IV: Top mispredicted categories of our model.

Etsy			Amazon		
True Cat.	Predicted Cat.	Perc. of. mispred.	True Cat.	Predicted Cat.	Perc. of. mispred.
Crochet	Clothing	2.5%	Sports & O.	Clothing & S.	7.8%
Children	Clothing	1.8%	Kindle S.	Books	7.2%
Toys	Children	1.7%	Clothing & S.	Sports & O.	3.1%
Patterns	Needlecraft	1.5%	Books	Kindle S.	2.9%

5) Results with Ensemble: The analysis done in the previous section shows that our model performs worse on small-length inputs and on those categories that are a bit similar to each other. In other terms, we have low confidence in our model's predictions in such scenarios. So, we can use the strategy described in section III-K.

The methodology for the ensemble is as follows; if a title length is small ($<= 7$ only on Etsy) then we recommend fastText's predictions instead of our model. Also, if fastText's top prediction is one of the specific categories where our model performs worse, then we also recommend fastText's predictions. Table V shows the accuracy achieved with the ensemble strategy along with the percentage of listings where each model's predictions were used in the final recommendation on the entire dataset (80-20 split). The gain in accuracy of the ensemble with respect to our model is close to 0.5% for both datasets. It is noteworthy that for Amazon our model's predictions are majorly used while still achieving a high accuracy.

TABLE V: Ensemble of both models with the percentage of the total items predicted with each model.

Dataset	Ensemble Accuracy	% Items predicted with our model	% Items predicted with fastText model
Etsy	88.1%	63.7%	36.3%
Amazon	90.8%	94.1%	5.9%

6) Comparison with various models: We show in Table VI, the accuracies of models from scikit-learn and BERT models with the previous models. We use 80-20 training/test splits of the entire Etsy dataset and also of a 25% sample of the Amazon dataset. The sampled dataset for Amazon is due to the large training times of BERT models. As expected, the performance of BERT is superior due to their model complexity. DistilBERT is a faster version of BERT that uses knowledge distillation to compress the large BERT model to a smaller model. This results in some drop in accuracy in contrast to BERT.

For the scikit-learn models, we use TF-IDF vectorization for transforming the text into numeric vectors. It is noteworthy that the Random Forest technique performs better than SVM and also Decision Tree. It has a large number of decision trees each generating a prediction that is ensembled for final

TABLE VI: Accuracy comparison using different baselines. A 25% sample of the Amazon dataset was used.

Models	Datasets	
	Etsy	Amazon
Our model	87.5%	90.2%
fastText	87.4%	90.1%
Ensemble	88.1%	90.6%
SVM	84.8%	89.6%
DecisionTree	78.4%	77.9%
RandomForest	87.3%	89.2%
BERT	90.1%	91.2%
DistilBERT	89.4%	91.0%

recommendation. Such a large group of trees are able to provide a discrete representation that is similar in complexity to the non-linear higher-order continuous representation of neural networks on these datasets. The ensemble model described in the previous section performs better than all models (except BERT models) on both datasets. Our transparent model and the ensemble with fastText is able to provide a more deterministic reasoning of the text classification process with better accuracy than all models described here. We discuss the execution times of all models next in the section IV-D.

D. Execution Performances

In this section, we compare the execution performances of our model first with fastText highlighting the nuances, and then we provide a comprehensive comparison with all models. We focus on the time taken for training and inference on the datasets.

1) Comparison with fastText: Table VII shows the execution times of both models. For our method, the preprocessing time is shown separately. In the case of fastText, it is included in both the training and test times. Our method has a faster test phase than fastText by $2.9\times$ on Etsy and $2.3\times$ on Amazon. Our model's amortized per input inference latency is approximately 0.01 ms for both datasets. Though fastText's inference time also qualifies for real-time prediction with approximately 0.02 ms. Our model reduces the daily batch prediction (categorization of all products) time by more than half.

TABLE VII: Execution time comparison.

Time (s)	Etsy		Amazon	
	Our	fastText	Our	fastText
Preprocessing	17.4	-	80.6	-
Training	1.2	77.9	6.4	324.8
Test	2.8	8.1	14.6	33.7

Even if we include the preprocessing time, our model's training is still significantly faster by $4.2\times$ on Etsy and $3.7\times$ on Amazon than fastText. Such faster training times enable frequent model refreshes (model re-training) which can accommodate newly listed products and new taxonomy for categorization which in our case takes only a few minutes. Frequent model refreshes are also beneficial for applying customized engineering to datasets for reducing mispredictions and fraud detection. Even with real-world data where there are hundreds of millions of listings, our method can scale to perform daily model refreshes.

2) Performances of various models: Table VIII shows the test and training times of all models in different units. Test times for all models except BERT models show that they are suitable for real-time inferencing on a per-input basis. Even the reduction in time by DistilBERT is not sufficient for real-time prediction. Despite their high accuracy, this is the primary drawback of Large Language Models (LLMs) which we had discussed before. The inference time is also not scalable for a daily batch prediction on hundreds of millions of products.

TABLE VIII: Execution times of different baselines.

Models	Test Time		Training Time	
	Etsy	Amazon	Etsy	Amazon
Our model	2.8 secs	14.6 secs	18.6 secs	28.8 secs
fastText	8.1 secs	33.7 secs	1.3 mins	3.1 mins
SVM	3.2 secs	17.7 secs	2.6 mins	2.7 mins
DecisionTree	3.3 secs	18.2 secs	1.3 hrs	2.2 hrs
RandomForest	14.3 secs	49.1 secs	15.9 mins	1.5 hrs
BERT	52 mins	1.1 hrs	29.3 hrs	34.1 hrs
DistilBERT	30.3 mins	36.1 mins	15.7 hrs	18.4 hrs

The training times of BERT take more than a day. Although this is mitigated by DistilBERT which reduces the time taken by half, there is some drop in accuracy as shown in Table VI. The scaling on very big datasets is still a concern. Such training times can hamper frequent model refreshes.

In fact, the increase in inference and training times offsets the difference in accuracy for practical applications. While LLMs come with heavy resource usage, the accuracies of CPU-based models such as our method and Random Forest can be improved by ensemble or domain-specific techniques. SVM's implementation *LinearSVC* scales quite well for large-scale classification as discussed in section II.

Decision Tree by design follows sequential execution which is difficult to parallelize. The tree is grown node by node depending on the criteria that direct the inputs towards more homogeneous groups. Hence the training times run into hours and are difficult to scale. Conversely, though a Random Forest is a collection of Decision Trees, it can be trained in parallel by different threads. Its training time is limited by the tree taking the largest time. On the contrary, both Decision Tree's and Random Forest's inferences can be done in parallel so the prediction is fast even if individual inference is sequential. Using Gini impurity instead of entropy can exacerbate the computational expense. Thus, both tree model's inference times can scale for very big datasets whereas their training will not scale.

E. Level 2 categories

We show the accuracy and execution times for the second-level categories on Etsy in Table IX for 80-20 split of the data. Our model is approximately 1% more accurate to fastText despite the drop in both their accuracies. In spite of the large of number categories (199) at the second level, the training time of fastText is not significantly higher than its training time at the first level. This is due to its hierarchical softmax layer which allows more parallel weight updates for each classifier (One-vs-all). This is beneficial because there are fewer weight updates per class at the second level as each category contains fewer listings compared to the first level. Our model's training scales linearly and there are no parameter updates or training epochs. Thus, we achieve a substantial speed of 6.3× over fastText in training time. Although the gap in both model's inference times is now smaller, it can be easily improved with a fine-grained implementation of our model's inference. This will enable threads to parallelly match the test listing with all categories.

TABLE IX: Second-level classification (Etsy $k = 199$).

Metric	Etsy	
	Our	fastText
Accuracy	81.67%	80.73%
Preprocessing time(s)	17.3	-
Training time (s)	2.3	123.6
Test time (s)	13.6	14.7

F. Results on Smaller Vocabulary

In this section, we present additional results with a smaller word vocabulary (370K words) containing only alphabetic characters. We also remove words with frequency ≤ 3 and > 100, and remove all non-alphabetic characters from the datasets. We show results on a sampled dataset on Etsy to better understand our methodology. The edge weights are modified to give higher weight to contiguous words, and unseen words are mapped as in Section III-I. We give performance results for the analysis method described in Section III-J. Also, we investigate the interpretability of our model by looking at sentences that are mispredicted, which consist of recognizable English words, unlike the previous vocabulary.

1) Analysis of graph properties: We evaluate each case mentioned in Section III-J for various graph properties on the Etsy dataset mentioned above with an 80-20 train-test split. Table X shows the percentage of listings in Etsy falling in each case. The last row shows the accuracy obtained when using each property as the first rule. For best accuracy, most of the sentences should belong to cases 1 and 4, with minimal sentences in cases 2 and 3. Case 5 sentences require classification using a second rule to break the ties. We can see that the edges matched and the number of triangles performed the best even though they have a larger percentage for case 4 than connected components and largest clique sizes. While the number of triangles is a better choice than edges matched, it is more expensive than simply checking edge matches.

2) Interpretability: As discussed in section III-H, we analyze an example of misprediction in the Etsy dataset. Given the sentence "indulge poodle recycled tee shirt back pack" belonging to category "Bags and Purses", our model mispredicts it as "Clothing". The product describes a *back pack* make of *recycled tee shirt* and the intuitive tendency for

TABLE X: Case analysis of different graph properties with percentage of test data for Etsy. EdMat=Edges matched, CC=Connected Components, LCS=Largest Clique Size, NumTri=Number of Triangles.

Cases	% Test Data			
	EdMat	CC	LCS	NumTri
Case 1	28.69	17.68	28.69	28.69
Case 2	1.67	5.52	1.67	1.67
Case 3	10.61	1.44	6.92	10.72
Case 4	49.26	2.35	41.1	49.82
Case 5	11.19	76.96	24.11	10.29
Accuracy	82.12	2.06	78.22	82.26

misprediction is evident. Table XI lists the 10 and 8 co-occurrences matches with *Clothing* and *Bags and Purses*, respectively. Even though the true category has an additional co-occurrence in row 5, the missing rows 1, 7 and 8 result in the misprediction. If we remove the words *poodle* and *tee*, which are more clothing related design terms than bags, the sentence is correctly classified as *Bags and Purses* with 6 co-occurrences.

TABLE XI: An example sentence and the co-occurrences matched by our model to categories Clothing and Bags(and Purses).

Sl. no	**Clothing**		**Bags**	
1	poodle	shirt	-	-
2	recycled	tee	recycled	tee
3	recycled	shirt	recycled	shirt
4	recycled	back	recycled	back
5	-	-	recycled	pack
6	tee	shirt	tee	shirt
7	tee	back	-	-
8	tee	pack	-	-
9	shirt	back	shirt	back
10	shirt	pack	shirt	pack
11	back	pack	back	pack

V. Conclusions and Future Work

We devised a simple sentence classification model based on word co-occurrence graphs and analysed its performance on e-commerce datasets for the product categorization task. The model achieves better accuracy than the popular model fastText and is faster for both training and inference. The performance gap is similar across multiple variations and splits for the e-commerce datasets Etsy and Amazon. fastText's use of subword information leads to good accuracy, especially on the smaller length inputs and it can distinguish product titles of similar categories. We overcome the limitations of our model by performing an ensemble of both our model and fastText based on specific scenarios. This ensemble provides better accuracy than all non-LLM models under consideration and is very close to DistilBERT's performance. Our model is adept for real-time inferencing and will scale well for 100× larger datasets. Its training time is in minutes allowing daily model refresh.

In the future, we will focus on enriching input tokens through contextual and semantic connections with other tokens from training vocabulary. This could help our model perform on par with LLMs like BERT, especially on smaller length inputs and smaller categories. We aim to extend our approach to other sentence classification tasks and datasets such as sentiment analysis of tweets, classification of question banks, and larger text data. We plan to do a comprehensive analysis with smaller LLMs that are faster, while also improving the running time of our model by implementing fine-grained parallelization (especially useful for a larger number of categories).

References

[1] Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. Challenges and applications of large language models, 2023.

[2] Aakash Sharma, Vivek M. Bhasi, Sonali Singh, Rishabh Jain, Jashwant Raj Gunasekaran, Subrata Mitra, Mahmut Taylan Kandemir, George Kesidis, and Chita R. Das. Stash: A comprehensive stall-centric characterization of public cloud vms for distributed deep learning. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, pages 1–12, 2023.

[3] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In D. Song, M. Carbin, and T. Chen, editors, *Proceedings of Machine Learning and Systems*, volume 5, pages 606–624. Curan, 2023.

[4] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2246–2251, Online, July 2020. Association for Computational Linguistics.

[5] Nicholas J. Belkin, B. G. Michell, and D. G. Kuehner. Representation of texts for information retrieval. In *Proc. 18th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 147–148, 1980.

[6] François Rousseau and Michalis Vazirgiannis. Graph-of-Word and TW-IDF: New approach to ad hoc IR. In *Proc. 22nd ACM Int'l. Conf. on Information & Knowledge Management (CIKM)*, page 59–68, 2013.

[7] Samer Hassan, Rada Mihalcea, and Carmen Banea. Random walk term weighting for improved text classification. *International Journal of Semantic Computing*, 1(04):421–439, 2007.

[8] Roi Blanco and Christina Lioma. Random walk term weighting for information retrieval. In *Proc. 30th Annual Int'l. ACM Conf. on Research and Development in Information Retrieval (SIGIR)*, page 829–830, 2007.

[9] Rada Mihalcea and Paul Tarau. TextRank: Bringing order into text. In *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, pages 404–411, 2004.

[10] Esteban Castillo, Ofelia Cervantes, Darnes Vilariño, and David Báez. UDLAP at SemEval-2016 task 4: Sentiment quantification using a graph based representation. In *Proc. 10th Int'l. Workshop on Semantic Evaluation (SemEval-2016)*, pages 109–114, 2016.

[11] Jianyi Liu, Jinghua Wang, and Cong Wang. A text network representation model. In *Proc. 5th Int'l. Conf. on Fuzzy Systems and Knowledge Discovery*, pages 150–154, 2008.

[12] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. In *Proc. 8th Int'l. Joint Conf. on Natural Language Processing (Volume 1: Long Papers)*, pages 253–263, 2017.

[13] A. Hassan and A. Mahmood. Convolutional recurrent deep learning model for sentence classification. *IEEE Access*, 6:13949–13957, 2018.

[14] Yoon Kim. Convolutional neural networks for sentence classification. In *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

[15] Armand Joulin, Édouard Grave, Piotr Bojanowski, and Tomáš Mikolov. Bag of tricks for efficient text classification. In *Proc. 15th Conf. of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, 2017.

[16] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

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

[18] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations, 2017.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[20] Ashwin Geet d'Sa, Irina Illina, and Dominique Fohr. Bert and fasttext embeddings for automatic detection of toxic speech. In *2020 International Multi-Conference on:"Organization of Knowledge and Advanced Technologies"(OCTA)*, pages 1–5. IEEE, 2020.

[21] Ashirbad Mishra, Soumik Dey, Jinyu Zhao, Marshall Wu, Binbin Li, and Kamesh Madduri. Graphite: A graph-based extreme multi-label short text classifier for keyphrase recommendation. In *Frontiers in Artificial Intelligence and Applications*. IOS Press, October 2024.

[22] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, jun 2008.

[23] Zornitsa Kozareva. 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*, pages 1329–1333, 2015.

[24] Ali Cevahir and Koji Murakami. Large-scale multi-class and hierarchical product categorization for an e-commerce giant. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 525–535, 2016.

[25] Vivek Gupta, Harish Karnick, Ashendra Bansal, and Pradhuman Jhala. Product classification in e-commerce using distributional semantics. *arXiv preprint arXiv:1606.06083*, 2016.

[26] Dan Shen, Jean David Ruvini, Manas Somaiya, and Neel Sundaresan. Item categorization in the e-commerce domain. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1921–1924, 2011.

[27] Hongshen Chen, Jiashu Zhao, and Dawei Yin. Fine-grained product categorization in e-commerce. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 2349–2352, New York, NY, USA, 2019. Association for Computing Machinery.

[28] Eli Cortez, Mauro Rojas Herrera, Altigran S. da Silva, Edleno S. de Moura, and Marden Neubert. Lightweight methods for large-scale product categorization. *Journal of the American Society for Information Science and Technology*, 62(9):1839–1848, 2011.

[29] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2(1–2):1–135, January 2008.

[30] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proc. 10th ACM SIGKDD Int'l. Conf. on Knowledge Discovery and Data Mining*, KDD '04, page 168–177, New York, NY, USA, 2004.

[31] Soo-Min Kim and Eduard Hovy. Determining the sentiment of opinions. In *Proceedings of the 20th International Conference on Computational Linguistics*, COLING '04, page 1367–es, USA, 2004. Association for Computational Linguistics.

[32] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, Hong Kong, China, November 2019. Association for Computational Linguistics.

[33] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[34] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[35] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.