Distributed Model Serving for Real-time Opinion Detection

Paul Pinter
TU Wien
Vienna, Austria
paul.pinter@student.tuwien.ac.at

Andrea Morichetta
Schahram Dustdar
Distributed Systems Group, TU Wien
Vienna, Austria
firstname.lastname@dsg.tuwien.ac.at

***Abstract*—The rapid evolution of the Web has revolutionized communication, enabling individuals to seek advice and share opinions on diverse subjects. However, this freedom has given rise to deceptive practices, such as manipulating product or business ratings through misleading reviews. While recent years have shown significant progress in opinion-based spam detection, the practical deployment of such systems remains a challenge, especially on modern distributed and heterogeneous platforms like the Web. Data distribution plays an essential role, as there is a need to collect as much information as possible from different sources. This paper addresses this gap by exploring the design challenges of distributed systems tailored for opinion spam detection. We evaluate three datasets, implement an accessible classification service, and test its performance on three distinct distributed system architectures. Our findings indicate the significant influence of certain features on classification performance and demonstrate the advantages of the asynchronous batch processing system over other architectures.**

***Index Terms*—Distributed Intelligence, Web characterization, Opinion spam detection, Distributed inference, Machine learning**

I. Introduction

In recent years, much effort has been put into developing counter-offensive work on different levels. Machine Learning (ML) has become a leading solution for big-scale problems. The advantages are several: little human intervention, precision, and the possibility to work in an automated fashion in different locations. A vivid example of this is the field of opinion-based spam detection, one of the most prominent demonstrations of Internet misuse. A 2021 study [1] analyzed streams of fake reviews and found that the average rating drops as soon as the flow of fake reviews stops. This finding indicates that customers are tricked into buying something they do not want or that does not reflect their standards due to a high rating. False or misleading claims also hurt the review hosts. Companies have taken action to deal with fake reviews to protect the integrity of platforms. Trip Advisor reported [2] having deleted 1 billion reviews in 2020. Furthermore, Amazon says [3] to have spent over 500 million dollars and employed 8 000 people to find unwanted reviews. Not only companies but also governments recognize how harmful fake reviews are. In 2021, the Competition and Markets Authority (CMA) of the UK government launched [4] an investigation of platforms that show reviews of goods and services. Based on the investigation results, the UK government has announced [5] to make posting fake reviews illegal. In addition, the CMA plans to fine companies if they do not take reasonable steps to check whether reviews are genuine or not. That customers pay attention to star ratings also shows survey [6] from Bright Local. The survey reports that only three percent of people would engage with a business with an average below a 3-star rating. This behavior incentivizes bad actors to post negative reviews on the competition and boost their products or services with positive reviews. Opinion-based spam detection aims to find fake reviews and users with malicious intent and exclude them from the platform.

Despite the excellent results in solving main problems for opinion detection [7], the focus on operationalizing it on distributed systems is missing [8]. Much of the production effort goes into model serving, representing the industry's 90% of output costs [9]. However, the discussion about this topic in the organization of the Internet, specifically in opinion spam detection, is still overlooked. Distributed ML solutions, like Tensorflow serving, offer guarantees for handling the execution of the models taking care of the resource management and scheduling. However, these tools are generic and leave the users with many implementation responsibilities and challenges [10]. In particular, specific models have some specific requirements for where and how the data should be collected and used for the prediction. Therefore, we aim to provide a magnifying lens to gain insights into the aspect of bringing models into production.

We follow a structured approach, showing the entire process of opinion spam detection, starting from feature engineering to model selection and operationalization. Furthermore, we contribute by proposing three possible architectures. We provide an analysis showing the advantages and disadvantages of each methodology. We evaluate it both in terms of computational performance and the effects of the model execution. We release the code, making everything reproducible and transparent. In summary, this work contributes to the field with the following:

(i) We propose a model training controller that performs the various ML training steps and makes the best model accessible for other applications.
(ii) Contextually, we show through experiments which feature category delivers the most impact on the classification performance.

2642-6587/24/$31.00 ©2024 IEEE
DOI 10.1109/SOSE62363.2024.00014

(iii) We design and make publicly available [1] three distributed system designs that use different communication methods.
(iv) We collect performance metrics to inspect how each system behaves under static and dynamic scenarios.

II. Background and Motivation

The current state of the Internet is witnessing unprecedented scale, dimension, and pervasiveness [11]. This evolving scenario poses severe challenges to who manages Internet services and those who use them. This situation also involves leading platforms; for example, Wikipedia has grown exponentially in the last few years, leading to problems in managing it and involving its users [12]. These challenges extend to every service, network, or business involving user interaction and to every person participating in that; nowadays, we have to face threats of fake news [13], fraudulent shops [14], and dishonest reviews. The rise in ML algorithms over the past two decades can be attributed to improved data availability, enhanced hardware, and superior technologies [15]. Though this progress has transformed the web, it has hidden particular obstacles. Deploying ML systems in production is intricate, with complexities surrounding automation and accuracy [16]. MLOps, which merges DevOps techniques with ML production, offers potential solutions across the model lifecycle [17], [18]. Still, every use case comes with its specific challenges. In generic solutions and models, the operational aspect of applied ML in production is frequently overlooked [16].

The serving perspective. The significance of model serving for enterprise ML has gained researchers' attention, especially in real-time contexts like the Web [9]. Various approaches have been proposed to improve system performance and reduce costs. While many center their studies around image recognition, our research tries to focus on another essential aspect of the Web, i.e., user interaction and feedback. We target the prevalent issue of fake reviews, focusing on the operationalization and system perspective, presenting the challenges from data representation to model serving.

Opinion spam detection. Research on opinion-based spam detection has grown over the past two decades to ensure online service credibility. Early works by Jindal et al. [19], [20] focused on identifying suspicious review patterns. Since then, advancements have spanned from data quality to sophisticated linguistic and behavioral features, as well as graph-based methodologies [21], [22], [23]. Although these studies optimize algorithms for better threat detection, they often neglect clarity on how to deploy these solutions in production. Only some works touch upon operationalizing fake-reviews detection models, an area we delve deeply into. Our research emphasizes reproducibility and the intricacies associated with model serving for opinion spam detection.

Distribution Aspect The mainstream adaptation of generative AI has resulted in an increase in artificial-generated content, such as reviews. Distributed systems are essential for opinion spam detection because they can scale horizontally, allowing for processing large volumes of data. By dividing the workload across multiple servers or nodes, these systems can handle the increase of data without compromising on performance or speed. Also distribution enhances the resilience. By decentralizing the operations, these services are less vulnerable to single points of failure. In the event of a server failure, other nodes can take over the load, ensuring continuous system operation. This robustness is crucial for maintaining the integrity of online services and the trust of their users. With the increasing demand for real-time responses in online interactions, distributed systems offer the capability to analyze and detect spam reviews instantly. This is important in dynamic environments where user feedback can influence immediate purchasing decisions of products and services. Online platforms often serve a global audience, making it necessary to manage data across different regions while complying with local data regulations. Strategically placed services in various geographical locations to reduce latency, enhance response times, and adhere to legal constraints regarding data sovereignty.

III. Methodology

A. Dataset

Dataset choice is essential. Conventional scalability tests focus on volume [24], [25]. In contrast, AI classification studies highlight dataset structure [26], [27]. Both dataset volume and quality are essential to emulate realistic high-workload scenarios for classification in distributed systems. We subsequently assess opinion-spam datasets to fit our criteria, emphasizing their construction and labeling. We consider three datasets. The HSpam14 dataset [28], collected by Sedhai and Sun, consisted of 20 million tweets from Twitter, now known as X. Web crawlers were used to fetch tweets containing popular hashtags from a preselected list. After excluding all non-English tweets, about 14 million remained, hence the name HSpam14. The tweets were grouped into clusters based on similarity and labeled according to the following process: 1) Tweets suggesting adult content, get-rich-quick schemes, or promotions were marked with specific keywords. 2) A subset in each cluster containing such tweets was manually labeled. Labels were propagated using a k-nearest neighbors algorithm (kNN) in clusters that had a high percentage of closely related tweets, identified as either ham or spam. 3) All tweets were grouped by user, clustered based on similarity, and a subset of each cluster containing at least ten tweets was manually labeled. 4) All tweets containing links to external resources were grouped and a subset of them was manually labeled. 5) Ham detection employed a classifier that was trained solely on positive and unlabeled tweets (PU). 6) The remaining tweets were annotated using an Expectation-Maximization Algorithm (EM). The dataset includes the tweet ID, label, and labeling step, with the labeling step numbered from 1 to 6 to indicate the stage at which each tweet was labeled.

The 3-Domain-Dataset [29], consists of reviews categorized into three distinct groups based on the type of service reviewed

[1] https://github.com/paulpinter/SOSE-2024-opinion-detection

or the reviewer's level of expertise. The services reviewed include hotels, restaurants, and medical services, while the reviewers are classified as experts, customers, or crowd workers. All crowd workers were recruited via Amazon's Mechanical Turk (AMT) crowdsourcing platform. The term "experts" refers to professionals within the specified service domains. It is assumed that all genuine reviews are submitted by customers who have actually visited the locations they reviewed. Reviews sourced from websites like Trip Advisor or Yelp must be in English, five-star rated, over 150 characters, and from users who have reviewed previously. Spam reviews were generated by either experts or crowd workers. While experts wrote reviews based on their familiarity with the service, crowd workers authored reviews without having visited the locations.

The Yelp ZIP dataset [23], compiled by Rayana and Akoglu, aggregates popular restaurant reviews from across the US, organized by zip codes. This dataset was created by executing search requests with query parameters that limited results to restaurants located within specific zip codes. Each search result included lists of both recommended and not recommended reviews. Subsequently, reviews from the recommended list were labeled as 'ham', whereas those from the not recommended list were tagged as 'spam'. This process was repeated using previously unused zip codes until approximately 600,000 reviews had been collected. Yelp's recommendation algorithm is reputed for detecting unwanted behavior, confirmed by manual checks.

Among the datasets, HSpam14 has the most expansive volume. However, this dataset references external content, necessitating additional steps to re-fetch this data. Moreover, tweets labeled as spam are more likely to have been block by the platform, since spamming activities are against Twitter's policies. On the other hand, the Yelp ZIP and 3-Domain datasets offer explicit data points. Yelp ZIP is suitable for stress tests with possible data expansion, but its imbalanced labels can compromise model quality. In contrast, the 3-Domain set is balanced but less flexible for workload tests. Yelp's labeling process has been validated [30], while the 3-Domain set ensures spam accuracy. The labeling process of HSpam14 section relies on the assumption that most tweets are not spam. HSpam14's labeling might be skewed by its base assumptions. Given these considerations, Yelp ZIP best meets our study's requirements.

B. Features

Previous works have represented the Product-Review-User relation as a graph $G(V, E)$. [31], [32], [33], [34]. Here, we introduce the notation for the construction of each feature. Let there be two different node types U and P such that:

$$V = U \cup P \tag{1}$$

where $U = \{u_1, \ldots, u_i, \ldots, u_n\}$ and $1 \leq i \leq n$ are the users and $P = \{p_1, \ldots, p_i, \ldots, p_m\}$ with $1 \leq j \leq m$ the product. Each review written by a user is represented as an edge:

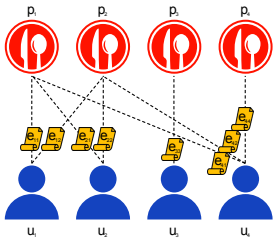


Fig. 1: Product-Review-User relation as a graph G(V,E)

$$E = \{\{u_i, p_j\} \mid u_i \in U, p_j \in P\} \tag{2}$$

where $e_{ij} \in E$ denotes a review written by user $u_i \in U$ for $1 \leq i \leq n$ and product p_j with $1 \leq j \leq m$. Figure 1 shows an example of a review graph with four users and four products.

We process the data with the $r(e_{ij}), d(e_{ij})$ and the $t(e_{ij})$ functions. Beginning with $r(e_{ij})$, each rating $r \in R = \{1, 2, 3, 4, 5\}$ is obtained by:

$$r(e_{ij}) : \begin{cases} E \to R \\ e_{ij} \mapsto r_{ij} \end{cases} \tag{3}$$

with $u_i \in U$ for $1 \leq i \leq n$ and product p_j with $1 \leq j \leq m$. Additionally a text $t \in T$ is gained via the $t(e_{ij})$ function:

$$t(e_{ij}) : \begin{cases} E \to T \\ e_{ij} \mapsto t_{ij} \end{cases} \tag{4}$$

with $u_i \in U$ for $1 \leq i \leq n$ and product p_j with $1 \leq j \leq m$. Every text t_{ij} is later broken down into sentences, words, characters, and eventually into a frequency-term-matrix (check in Appendix A). Next, the creation date is mapped with:

$$d(e_{ij}) : \begin{cases} E \to D \\ e_{ij} \mapsto d_{ij} \end{cases} \tag{5}$$

where is $u_i \in U$ for $1 \leq i \leq n$ and product p_j with $1 \leq j \leq m$. The creation date is ordered. A date $d_{ab} \in D$ is before $d_{xy} \in D$ if and only if $d_{ab} < d_{xy}$. A creation date d_{ab} is after d_{xy} when $d_{ab} > d_{xy}$ holds. If $d_{ab} = d_{xy}$ is true then the date refers to the same day.

In the same manner the difference of $d_{ab} - d_{xy}$ is the number of days of the two dates apart if $d_{ab} < d_{xy}$, 0 if $d_{ab} = d_{xy}$ and the negative number of days apart otherwise.

Ultimately, we use [35] star $*$ notation. E_{i*} is the set of all reviews written by user $u_i \in U$ for $1 \leq i \leq n$. E_{*j} corresponds to all reviews of a product $p_j \in P$ with $1 \leq j \leq m$.

In the Appendix A each feature will be constructed with this notation. The features were derived from SPEAGLE [23]. Table I summarizes the features. The first column reports the feature code, followed by a description and the reference of the feature characteristic. The third column reports whether a feature is linguistic or behavioral. Indeed, a feature can not be both. Linguistic features are calculated from the review text. Behavior features take ratings or dates as input. The

TABLE I: An overview of the features used to train the models

Feature	Description	B/L	I/D
ACS	Average content similarity [32], [35]	L	D
ALW	Average review length in words [36]	L	D
ARD	Average rating deviation [35], [36]	B	D
BRT	Burstiness [32], [35],[36]	B	D
ERD	Entropy of rating distribution [23]	B	D
ETG	Entropy of temporal gaps [23]	B	D
IPO	Is positive review [37]	B	I
ISR	Is singleton review [23]	B	I
MCS	Maximum content similarity [37],[36]	L	I
MNR	Maximum number of reviews written in a day [37],[36]	B	D
PRD	Product rating deviation [38]	B	D
RAC	Ratio of ALL-capitals words [38]	L	I
RBD	Rank by date [20]	B	D
RCL	Ratio of capital letters [38]	L	I
RES	Ratio of exclamation sentences containing '!' [38]	L	I
RLW	Review length in words [38]	L	I
RNR	Ratio of negative reviews [36]	B	D
RPP	Ratio of 1st person pronouns [38]	L	I
RPR	Ratio of positive reviews [36]	B	D
RSW	Ratio of subjective words [38]	L	I
WRD	Weighted rating deviation [32]	B	D

last column tells if a feature is *independent or dependent*. A feature is independent if a single review determines its value. In contrast, dependent features rely on user, product, and user-product information.

C. Architecture

This section elaborates on our architecture. Initially, we delve into the service responsible for model management. We then illustrate three system designs for serving spam detection models based on opinions.

The Model Creation Service ingests a labeled review dataset and produces a trained classifier. Except for the database, this is the only other component that remains unchanged across all architectural designs. After receiving an unlabeled review, the model creation service extracts the features from the reviews. Since some features are dependent on other's, the service needs to consider the whole dataset and cannot extract features from the review itself. As depicted in Figure 2, the process commences by calculating a feature matrix as per §III-B. This matrix trains several classifiers, from which the one with the highest F_1 score is selected. The F_1 score was chosen as the ranking metric because it provides a balanced measure of a model's recall and precision, particularly in cases where class distributions are imbalanced. Precision is crucial in spam detection because it measures the accuracy of the spam predictions that the model makes, which directly affects the user experience by ensuring that legitimate messages are not incorrectly classified as spam. Conversely, recall is equally important because it assesses the model's ability to identify all actual spam messages, which is necessary for maintaining the usability of platforms. Handling imbalanced distributions is relevant in spam detection, as well as in our selected Yelp ZIP dataset, which suffers from imbalance. In scenarios where the negative class significantly outnumbers the positive class, traditional accuracy might not provide a true picture of a model's effectiveness. A classifier could perform well by predicting the majority class, but still fail to catch a substantial number of spam messages. By focusing on both false positives and false negatives, it ensures that both types of errors are

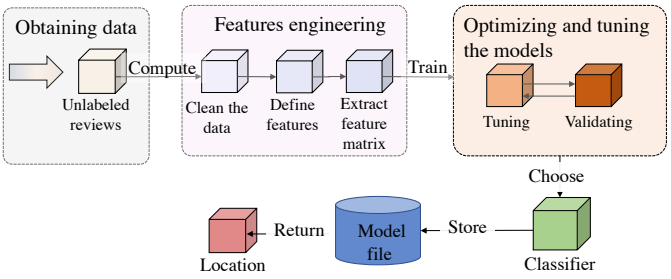


Fig. 2: Flow of the model creation service.

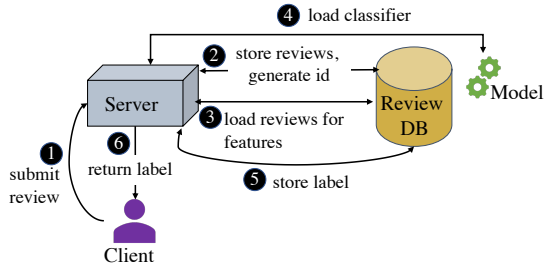
minimized. Ultimately the chosen classifier is serialized, and its path is stored in a Relational Database (RDM).

A dedicated server was established for **synchronous communication** that classifies incoming reviews as spam or ham. Figure 3a showcases its architecture.The synchronous server ensures that each request is processed in the order it is received, and each response is generated after the processing of the current request is fully complete. This model is beneficial primarily for its simplicity and ease of debugging, making it an excellent choice for environments where sequential processing of data is critical. Each review is handled one at a time, which simplifies the logic of transaction handling and error tracking. This method reduces the complexities associated with concurrent data access and resource locking, which are common in asynchronous systems. However, while the synchronous model is advantageous for simplicity and straightforward operation, it inherently limits the system's throughput and scalability. This limitation arises because the server must complete processing the current review before it can start another, potentially leading to increased response times and reduced overall system performance.

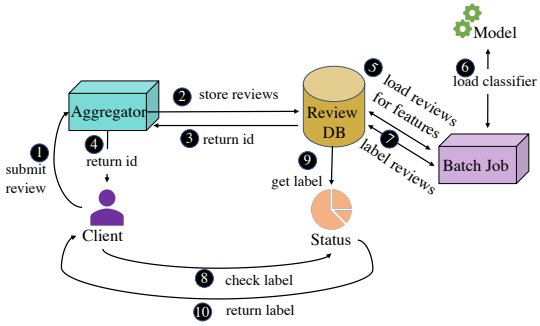
The method of feature computation for this service differs from the classifier service; it computes a feature vector for the incoming review. Dependent features are categorized as user, product, or user-product. User features depend on all reviews written by a user. In the same manner, reviews that are product features depend on all reviews written for a product. Since user and product features depend on previously written reviews, a subset of all stored reviews must be loaded into the request context. Finally, user-product features are a combination of both mentioned subsets. The combined subset gets created by a two-step process. First, select all reviews of a user. Subsequently, fetch all product reviews of the selected reviews. Combining these subsets allows a feature vector to be deduced, which is then passed to the classifier. Once labeled, a response is dispatched to the client.

We developed two systems to explore **asynchronous communication**. The *batch* system periodically polls the data source, while the *queue* system employs a distributed message queue. Unlike synchronous communication, where clients receive an instant classification, the asynchronous methods provide a status URL to check the review's labeling status.

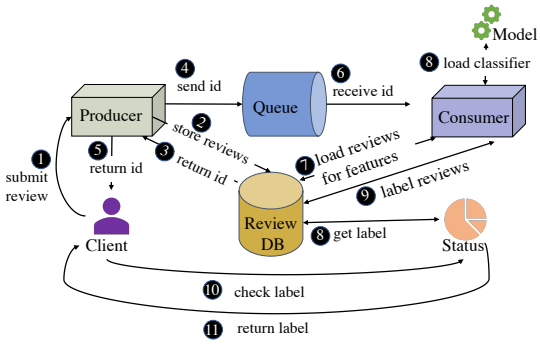
Figure 3b outlines the *batch* system's architecture. Unlike prior models, this architecture integrates a batch job and status



(a) Synchronous system architecture for spam detection.



(b) Batch system's architecture for spam detection



(c) Queue system's architecture for spam detection

Fig. 3: Asynchronous communication designs.

module. Reviews are submitted to the aggregator, stored, and later fetched by the batch service, which processes and stores the classifier's prediction. In the synch and the queue systems, each review or data input is processed instantaneously, often leading to a sparse feature set for new entities. This can result in less accurate predictions due to the lack of historical data. Conversely, the batch processing system collects and stores reviews in an aggregated manner. This allows for a more substantial compilation of data to be processed in a single batch. For bots who may submit multiple reviews in a short period or new products that receive a sudden increase of reviews, this system is capable of integrating this fresh information directly into the feature vectors during the next batch processing cycle.

As depicted in Figure 3c, the *queue* system comprises three server components. The producer receives and forwards reviews to a queue. A consumer then retrieves, classifies, and saves the reviews. This consumer can batch-process reviews by collectively computing features. While the batch system consolidates and processes reviews collectively at designated intervals, allowing comprehensive feature vector updates, the queue system processes data inputs continuously and independently. This can lead to scenarios where two related reviews are handled by different consumers. When it occurs, there is a potential for informational loss as each consumer processes data independently without immediate access to concurrent submissions that might be related.

While we cannot provide specific recommendations for query optimizations for a general architecture, we want to highlight that optimizing the querying process for these related reviews is crucial for enhancing server performance. Given that the computation of feature vectors necessitates loading a subset of all stored reviews into the request context, the efficiency with which this data is retrieved and processed impacts the responsiveness and scalability of the service. Furthermore, the complexity of data retrieval escalates with the activeness of the user or the ranking of a product. For instance, a bot who writes numerous reviews or a product that is highly popular and thus reviewed frequently, poses a greater challenge in terms of computational load. With more information available, the potential combinatorial explosion in the user-product review graph necessitates more sophisticated query optimization techniques to maintain performance.

IV. Evaluation

A. Model Selection

We assess three classification algorithms: naive Bayes (NB), logistic regression (LOG), and support vector machine (SVM) on three Yelp datasets: ZIP, CHI [36], and NYC [23]. We adopted Gaussian naive Bayes due to non-discrete features in Table III-B. Naive Bayes is frequently employed in the analysis of large datasets due to its rapid convergence properties. Initial tests on another variant, complement naive Bayes, were halted because of poor performance. Similar to NB, LOG can process both discrete and binary data. Additionally, LOG typically exhibits greater resilience to outliers. Support Vector Machines were selected for their capability to manage high-dimensional feature spaces. However, SVMs do not converge quickly on large datasets. In our experiments they were limited to a linear kernel due to computational constraints.

B. Classifier Performance

Using the *model creation service*, we applied grid search with F_1 scoring and 10-fold cross-validation for hyperparameter optimization. Training-test split was 4:1, and cross-validation was from the training set. Final performance was based on test set predictions. Table II contrasts linguistic and behavioral features. NB consistently achieved top accuracy and precision, while LOG stood out in F_1 score. Linguistic features performed best on Yelp CHI, whereas behavioral features were superior on the other two datasets, with a 0.05 point average advantage. Table III highlights the comparison between independent and dependent features. NB led in accuracy and precision, LOG in F_1 score, and SVMs in recall. Dependent

TABLE II: Performance of classifiers trained with linguistic or behavioral features

Classifier	Split	Dataset	Accuracy	F_1	Precision	Recall	ROC AUC
NB	L	CHI	**0.792**	0.32	**0.287**	0.363	0.611
NB	B	CHI	0.679	0.387	0.261	0.753	0.71
NB	L	NYC	0.863	0.142	**0.195**	0.112	0.53
NB	B	NYC	**0.867**	0.126	0.191	0.094	0.525
NB	L	ZIP	0.831	0.165	0.238	0.126	0.532
NB	B	ZIP	**0.838**	0.228	**0.309**	0.181	0.56
LOG	L	CHI	0.658	**0.39**	0.257	**0.809**	0.722
LOG	B	CHI	0.681	0.388	0.262	0.748	0.71
LOG	L	NYC	0.506	0.227	0.135	0.713	0.598
LOG	B	NYC	0.585	**0.277**	0.169	0.782	0.672
LOG	L	ZIP	0.572	0.275	0.177	0.612	0.589
LOG	B	ZIP	0.619	**0.354**	0.228	0.786	**0.69**
SVM	L	CHI	0.657	0.388	0.256	0.805	0.72
SVM	B	CHI	0.64	**0.39**	0.253	0.853	**0.73**
SVM	L	NYC	0.501	0.226	0.134	0.717	0.597
SVM	B	NYC	0.576	0.274	0.166	**0.785**	**0.669**
SVM	L	ZIP	0.572	0.274	0.177	0.609	0.588
SVM	B	ZIP	0.613	0.351	0.226	**0.79**	0.688

TABLE III: Performance of classifiers trained with independent or dependent features

Classifier	Split	Dataset	Accuracy	F_1	Precision	Recall	ROC AUC
NB	CHI	I	**0.832**	0.202	**0.281**	0.157	0.547
NB	CHI	D	0.678	0.392	0.263	0.772	0.717
NB	NYC	I	0.863	0.142	**0.195**	0.111	0.53
NB	NYC	D	**0.867**	0.122	0.185	0.091	0.523
NB	ZIP	I	**0.84**	0.136	0.24	0.095	0.524
NB	ZIP	D	0.82	0.274	**0.295**	0.255	0.581
LOG	CHI	I	0.693	0.388	0.266	0.722	0.706
LOG	CHI	D	0.661	**0.403**	0.264	0.85	0.74
LOG	NYC	I	0.687	0.235	0.156	0.471	0.591
LOG	NYC	D	0.574	**0.276**	0.167	0.795	0.672
LOG	ZIP	I	0.539	0.272	0.172	0.65	0.586
LOG	ZIP	D	0.614	**0.352**	0.226	0.791	**0.689**
SVM	CHI	I	0.692	0.388	0.265	0.723	0.705
SVM	CHI	D	0.659	**0.403**	0.264	**0.855**	**0.742**
SVM	NYC	I	0.538	0.23	0.139	0.679	0.6
SVM	NYC	D	0.563	0.271	0.164	**0.798**	**0.668**
SVM	ZIP	I	0.538	0.272	0.172	0.651	0.586
SVM	ZIP	D	0.606	0.349	0.223	**0.795**	0.686

features surpassed independent ones by an average of 0.07 points in most metrics except accuracy. The final experiment (shown in Table IV) indicates that NB was superior in accuracy and precision, while LOG excelled in F_1 and ROC AUC scores. SVM had the highest recall. The Yelp CHI dataset rendered the best scores for all classifiers, despite being the smallest dataset, implying that dataset size was not directly linked to classifier performance.

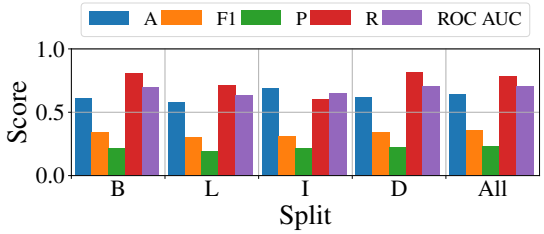


Fig. 4: Performance across different feature splits

Figure 4 consolidates the experiment results, illustrating the impact of different features on classifier performance. It indicates that dependent features significantly influence the results.

V. Distributed Serving Performance

Experiments showed that the dependent feature split's performance closely mirrors the all-classifier's. Recognizing the potency of dependent features, subsequent tests centered on the subset size's effects in terms of time and resources. Both were tracked across SYNC, BATCH, and QUEUE systems, as detailed in section III-C. Given the asynchronous nature of the latter two systems, request time was gauged from storage to labeling. In each scenario, 3,000 reviews were sequentially dispatched via an HTTP client to every system.

a) Implementation

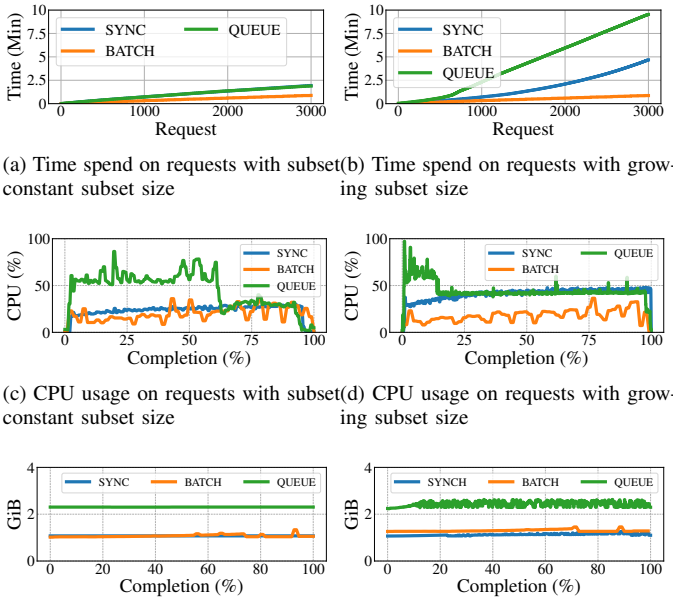
Components were stored in a PostgreSQL database and scripted in Python 3.7. The HTTP handler used Flask, while the message queue employed Apache Kafka. Classifiers were built with sci-kit learn and serialized using the job lib API. Every test utilized a Mac mini with an M1 processor and 16 GB of RAM, monitoring resources via Docker.

b) Constant subsets

Every review here had a unique user and restaurant, keeping the subset size constant at one. This made dependent features effectively independent, representing optimal single-review processing. Figure 5a depicts request times: the batch system finished in 53 seconds, the QUEUE in 113, and the synch system in 166. CPU usage (Figure 5c) averaged 45% for

TABLE IV: Performance of classifiers trained with all features

Classifier	Dataset	Accuracy	F_1	Precision	Recall	ROC AUC
NB	CHI	**0.683**	0.393	0.265	0.761	0.716
NB	NYC	**0.823**	0.227	**0.205**	0.255	0.571
NB	ZIP	**0.815**	0.278	**0.288**	0.268	0.583
LOG	CHI	0.68	**0.405**	**0.27**	0.807	**0.734**
LOG	NYC	0.617	**0.291**	0.179	0.772	**0.686**
LOG	ZIP	0.64	**0.364**	0.238	0.778	**0.698**
SVM	CHI	0.676	0.404	0.268	**0.813**	**0.734**
SVM	NYC	0.608	0.288	0.176	**0.776**	0.683
SVM	ZIP	0.634	0.361	0.235	**0.78**	0.696



(a) Time spend on requests with subset constant subset size

(b) Time spend on requests with growing subset size

(c) CPU usage on requests with subset constant subset size

(d) CPU usage on requests with growing subset size

(e) Memory usage on requests with subset constant subset size

(f) Memory usage on requests with growing subset size

Fig. 5: Time, CPU, and Memory performance for both constant and growing subset size.

the QUEUE, 23% for the synch system, and 20% for the batch system. Memory consumption was consistent across tests (Figure 5e).

c) Growing subsets

Here, every review came from the same user but for different products. This scenario highlighted the BATCH system's efficiency. Figure 5b reveals the BATCH system was notably faster, labeling reviews in 52 seconds. In contrast, the SYNC and QUEUE systems took 281 and 572 seconds, respectively. While the BATCH system also consumed fewer CPU resources, memory requirements were relatively stable across all systems.

VI. DISCUSSION

Classifiers performance: The hyperparameters were optimized for the highest F_1 score due to their efficacy on imbalanced datasets. Naive Bayes consistently showed the best accuracy and precision, logistic regression led in F_1 score, and SVMs in the recall. Classifiers trained on only review and linguistic features underperformed compared to those trained on behavioral, dependent, and all features. The all-classifier delivered the top F_1 and combined score, closely followed by the dependent feature-split, despite it using nine fewer features. Grouping feature vectors streamline processing, but parallelizing this introduces delays, potentially allowing malicious actions to persist. Non-deterministic outcomes arise when data distribution causes reviews to reside in separate databases. **Distributed serving:** In scalability, the QUEUE model outperforms due to its flexible producer-consumer pairing, while the synchronous model struggles with resource utilization. Additional safeguards are needed in the batch job service to avoid redundant review filtering. The BATCH system boasts superior stability, whereas the QUEUE system's queue is its vulnerability, although this can be mitigated [39]. The SYNC system's longer client-server connections increase its susceptibility to errors. **Systems evaluation:** The SYNC and QUEUE systems showed comparable performance in constant subsets, but the QUEUE lagged in linear growing subsets. Both these systems face a completion penalty with larger subset sizes, not seen in the BATCH system. CPU resource consumption during spikes was highest in the QUEUE system. Interestingly, increasing subset size did not affect average memory needs but did introduce variability. Overall, the BATCH system outperformed both in speed and efficiency.

General Applicability: The architectures discussed in this paper, while evaluated within the context of distributed spam detection, have broader implications for distributed computing tasks. The adaptability and performance of these systems suggest potential applications beyond spam detection to other classification tasks that require robust, scalable solutions across distributed environments. This could include real-time data processing for various types of network security, fraud detection, and dynamic information filtering across platforms, which are critical in today's data-intensive scenarios. The insights gained from this study could guide the design and implementation of distributed systems for a wide range of applications, where data volume, velocity, and variety are major considerations.

VII. CONCLUSION

This work provides an explicit definition of 21 features using a graph-based approach. Although the classifier did not reach the performance of the latest research, the chosen path was ideal for measuring the significance of distinguished feature categories. Multiple classification techniques with different datasets showed that dependent features have the most impact on classification performance. The model creation service was born out of the necessity to automate the creation, updates and comparison of multiple classifiers. By reasonable assumption, this component is flexible enough to distribute any model. Out of three architectures, the BATCH system completed the workload faster while also using fewer resources. All three systems are classifier agnostic and are combinable to negate their downsides. Given these strengths, the designs are suited to serve as a blueprint for future implementations of opinion based spam detection on distributed systems.

VIII. ACKNOWLEDGMENT

This work is funded by the HORIZON Research and Innovation Action 101135576 INTEND "Intent-based data operation in the computing continuum".

REFERENCES

[1] S. He *et al.*, "The market for fake reviews," *Marketing Science*, 2022.
[2] T. Advisor. (2021) Review transparency report. [Online]. Available: https://www.tripadvisor.co.uk/TransparencyReport2021
[3] J. Keegan. (2020) Is this amazon review bullsh*t? [Online]. Available: https://themarkup.org/the-breakdown/2020/07/21/how-to-spot-fake-amazon-product-reviews
[4] GOV.UK. (2020) Cma investigates misleading online reviews. [Online]. Available: https://www.gov.uk/government/news/cma-investigates-misleading-online-reviews
[5] ——. (2022) New rules to protect consumers' hard-earned cash. [Online]. Available: https://www.gov.uk/government/news/new-rules-to-protect-consumers-hard-earned-cash
[6] J. Pitman. (2022) Local consumer review survey 2022. [Online]. Available: https://www.brightlocal.com/research/local-consumer-review-survey/
[7] Y. Dou *et al.*, "Enhancing graph neural network-based fraud detectors against camouflaged fraudsters," in *Proc. ACM CIKM*, 2020.
[8] C. Zhang *et al.*, "MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving," in *Proc. USENIX ATC*, 2019.
[9] K. Park *et al.*, "End-to-end optimization of machine learning prediction queries," *arXiv preprint arXiv:2206.00136*, 2022.
[10] P. Barham *et al.*, "Pathways: Asynchronous distributed dataflow for ml," *Proc. Machine Learning and Systems*, vol. 4, pp. 430–449, 2022.
[11] S. Kemp. (2022) Digital 2022: Global overview report. [Online]. Available: https://datareportal.com/reports/digital-2022-global-overview-report
[12] Wikipedia. (2022) Wikipedia:why is wikipedia losing contributors - thinking about remedies. [Online]. Available: https://en.wikipedia.org/wiki/Wikipedia:Why_is_Wikipedia_losing_contributors_-_Thinking_about_remedies
[13] V. Pérez-Rosas *et al.*, "Automatic detection of fake news," *arXiv preprint arXiv:1708.07104*, 2017.
[14] L. Beltzung *et al.*, "Real-time detection of fake-shops through machine learning," in *Proc. IEEE BigData*, 2020.
[15] A. D'Alconzo *et al.*, "A survey on big data for network traffic monitoring and analysis," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 3, 2019.
[16] D. Sculley *et al.*, "Hidden technical debt in machine learning systems," *Advances in neural information processing systems*, vol. 28, 2015.

[17] Google. (2020) Mlops: Continuous delivery and automation pipelines in machine learning. [Online]. Available: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

[18] G. Symeonidis *et al.*, "Mlops-definitions, tools and challenges," *CoRR*, vol. abs/2201.00162, 2022.

[19] N. Jindal and B. Liu, "Opinion spam and analysis," in *Proc. WSDM*, 2008.

[20] N. Jindal *et al.*, "Finding unusual review patterns using unexpected rules," in *Proc. ACM CIKM*.

[21] M. Ott *et al.*, "Finding Deceptive Opinion Spam by Any Stretch of the Imagination," in *Proc. ACL*, 2011.

[22] Q. Xu and H. Zhao, "Using Deep Linguistic Features for Finding Deceptive Opinion Spam," 2012, pp. 1341–1350.

[23] S. Rayana and L. Akoglu, "Collective Opinion Spam Detection: Bridging Review Networks and Metadata," in *Proc. ACM KDD*, 2015.

[24] A. Al-Said Ahmad and P. Andras, "Measuring and Testing the Scalability of Cloud-based Software Services," 2018.

[25] D. Yang and P. Thiengburanathum, "Scalability and Robustness Testing for Open Source Web Crawlers," in *2021 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunication Engineering*, 2021, pp. 197–201.

[26] M. Gong *et al.*, "An Attention-Based Unsupervised Adversarial Model for Movie Review Spam Detection," vol. PP, pp. 1–1, 2020.

[27] J. Li *et al.*, "Fusion Convolutional Attention Network for Opinion Spam Detection," in *ICONIP*.

[28] S. Sedhai and A. Sun, "HSpam14: A Collection of 14 Million Tweets for Hashtag-Oriented Spam Research," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '15. Association for Computing Machinery, 2015, pp. 223–232. [Online]. Available: https://doi.org/10.1145/2766462.2767701

[29] J. Li *et al.*, "Towards a General Rule for Identifying Deceptive Opinion Spam," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2014, pp. 1566–1576. [Online]. Available: https://aclanthology.org/P14-1147

[30] M. Luca and G. Zervas, "Fake It Till You Make It: Reputation, Competition, and Yelp Review Fraud," 2013.

[31] L. Akoglu *et al.*, "Opinion fraud detection in online reviews by network effects," pp. 2–11, 2013.

[32] G. Fei *et al.*, "Exploiting Burstiness in Reviews for Review Spammer Detection," vol. 7, no. 1, pp. 175–184. [Online]. Available: https://ojs.aaai.org/index.php/ICWSM/article/view/14400

[33] S. Shehnepoor *et al.*, "NetSpam: A Network-Based Spam Detection Framework for Reviews in Online Social Media," *IEEE Trans. Inf. Forensics Security*, vol. PP, pp. 1–1, 2017.

[34] G. Wang *et al.*, "Review Graph Based Online Store Review Spammer Detection," in *Proc. IEEE ICDM*, 2011.

[35] E.-P. Lim *et al.*, "Detecting product review spammers using rating behaviors," in *Proceedings of the 19th ACM International Conference on Information and Knowledge Management - CIKM '10*. ACM Press, 2010, p. 939. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1871437.1871557

[36] A. Mukherjee *et al.*, "What Yelp Fake Review Filter Might Be Doing?" vol. 7, no. 1, pp. 409–418, 2013. [Online]. Available: https://ojs.aaai.org/index.php/ICWSM/article/view/14389

[37] ——, "Spotting opinion spammers using behavioral footprints," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13. Association for Computing Machinery, 2013, pp. 632–640. [Online]. Available: https://doi.org/10.1145/2487575.2487580

[38] F. Li *et al.*, "Learning to Identify Review Spam." 2011, pp. 2488–2493.

[39] M. Rostanski *et al.*, "Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ," in *2014 Federated Conference on Computer Science and Information Systems*, 2014, pp. 879–884.

[40] A. Esuli and F. Sebastiani, "SentiWordNet: A Publicly Available Lexical Resource for Opinion Mining," 2006.

APPENDIX

A. Word analysis

As mentioned in III-B, reviews will be broken down into minor elements. Let the $w(t_{ij})$ with $u_i \in U$ and $p_j \in P$ return the set of words of a given text $t_{ij} \in T$. The review length in words is then determined by

$$x_{\text{RLW}}(e_{ij}) = |\{w \mid w \in w(t(e_{ij}))\}| \tag{6}$$

where $e_{ij} \in E$. Accordingly, the average review length written by a user is

$$x_{\text{ALW}}(u_i) = \frac{1}{|E_{i*}|} \sum_{e_{ij} \in E_{i*}} |w(e_{ij})| \tag{7}$$

where $u_i \in U$. Despite the absolute and the average length, words with specific properties are observed. For instance, the ratio of subjective terms is given by

$$x_{\text{RSW}}(e_{ij}) = \frac{|\{w \mid w \in w(t(e_{ij})), w \in SU\}|}{x_{\text{RLW}}(e_{ij})} \tag{8}$$

$e_{ij} \in E$ and SU are the set of subjective terms. The collection of subjective words was used by SENTIWORDNET[40]. In contrast, the ratio of objective terms is given by

$$x_{\text{ROW}}(e_{ij}) = \frac{|\{w \mid w \in w(t(e_{ij})), w \in OW\}|}{x_{\text{RLW}}(e_{ij})} \tag{9}$$

with $e_{ij} \in E$. Naturally SENTIWORDNET also contains a list of objective terms OW. From a grammatical point of view, the ratio of first-person pronouns is of interest:

$$x_{\text{RPP}}(e_{ij}) = \frac{|\{w \mid w \in w(t(e_{ij})), w \in PP\}|}{x_{\text{RLW}}(e_{ij})} \tag{10}$$

where $e_{ij} \in E$. In order to determine first-person pronouns, each review gets POS-tagged, filtered by pronouns, and ultimately compared with the list of first-person pronouns.

B. Character analysis

A more granular approach than word-based analysis is character-based analysis. As a start, let CW be the set of all capital words. The ratio of words that contain only upper case letters is given by:

$$x_{\text{RAC}}(e_{ij}) = \frac{|\{w \mid w \in w(t(e_{ij})), w \cap CW = w\}|}{x_{\text{RLW}}(e_{ij})} \tag{11}$$

with $e_{ij} \in E$. Similarly let CL be the set of capital letters and $c(t_{ij})$ with $u_i \in U$ and $p_j \in P$ return a list of characters of a given text $t_{ij} \in T$ such that the ratio of capital letters is given by

$$x_{\text{RCL}}(e_{ij}) = \frac{|\{c \mid c \in c(t(e_{ij})), c \in CL\}|}{|\{c \mid c \in c(t(e_{ij}))\}|} \tag{12}$$

with $e_{ij} \in E$. Finally, let $s(t_{ij})$ with $u_i \in U$ and $p_j \in P$ return a set of sentences of a given text $t_{ij} \in T$ such that each sentence s is an ordered list of characters that represent the

sentence s. The sentence model allows checking whether an exclamation mark is in a sentence.

$$x_{\text{RES}}(e_{ij}) = \frac{|\{s \mid s \in s((e(t_{ij})), s \cap \{\text{'!'}\} \neq \{\}\}|}{|\{s \mid s \in s(t(e_{ij}))\}|} \tag{13}$$

with $e_{ij} \in E$.

C. Content analysis

Most spammers post similar reviews [36]. In this work, the likeness of the two reviews is determined with the cosine similarity function. First, a frequency matrix is constructed with a Bag-Of-Word model such that:

$$B_{ij,kl} = bow(t(e_{ij}) \uplus bow(t(e_{kl})) \tag{14}$$

where $bow(w(e_{ij}))$ returns a word frequency vector, $\uplus$ is the disjoint union and $B_{ij,kl}$ the term frequency matrix of $t_{ij}, t_{kl} \in T$. Next, the cosine similarity of the two reviews is given by:

$$sim(B_{ij,kl}) = \frac{\mathbf{B_{ij}} \cdot \mathbf{B_{kl}}}{\|\mathbf{B_{ij}}\|\|\mathbf{B_{kl}}\|} \tag{15}$$

where $\cdot$ is the dot product $\mathbf{B_{ij}}, \mathbf{B_{kl}}$ frequency term vectors and $\|\mathbf{x}\|$ the euclidian norm. Accordingly, the average content similarity is given by:

$$x_{\text{ACS}}(e_{ij}) = \frac{1}{|E_{i*}|} \sum_{e_{kl} \in E_{i*}} sim(B_{ij,kl}) \tag{16}$$

with $e_{ij} \in E$. In addition to x_{ACS} the maximum content similarty is

$$x_{\text{MCS}}(e_{ij}) = \max_{e_{kl} \in E_{i*}} sim(B_{ij,kl}) \tag{17}$$

with $e_{ij} \in E$.

D. Is singleton review

Behavioral Features are excerpts of the users' actions. As an example, the feature $x_{\text{ISR}}(u_{ij})$ flags users that have submitted only one review:

$$x_{\text{ISR}}(u_{ij}) = \begin{cases} 0 & \text{if } |E_{i*}| \neq 1 \\ 1 & \text{otherwise} \end{cases} \tag{18}$$

where $u_{ij} \in U$. Singletons are harder to judge whether their behavior is deceptive or not as they have not produced much data for a spam detection algorithm.

E. Is positive review

Succeeding with x_{IPO}, the rating is transformed into a binary feature that separates positive reviews from the rest by:

$$x_{\text{IPO}}(e_{ij}) = \begin{cases} 0 & \text{if } r(e_{ij}) > 3 \\ 1 & \text{otherwise} \end{cases} \tag{19}$$

with $e_{ij} \in E$. x_{IPO} converts a rating into a thumps-up and thumps-down model.

F. Ratio of positive reviews

A spammer's goal is to boost a range of products with positive reviews. Therefore capturing the ratio of positive reviews might identify spammers that operate with that goal in mind. Initially, the proportion of a given rating $r \in R$ by a user $u_i \in U$ is:

$$rr(u_i, r) = \frac{|\{e_{ij} \mid e_{ij} \in E_{i*}, r(e_{ij}) = r\}|}{|E_{i*}|} \tag{20}$$

With equation 20 the ratio of positive is calculated by:

$$x_{\text{RPR}}(u_i) = rr(u_i, 4) + rr(u_i, 5) \tag{21}$$

Where a positive review refers to a rating higher than 3.

G. Ratio of negative reviews

In contrast to x_{RPR}, the goal of a spammer might be to discourage customers from buying from the competition. With equation 29, the ratio of negative reviews is constructed by:

$$x_{\text{RNR}}(u_i) = rr(u_i, 1) + rr(u_i, 2) \tag{22}$$

where $u_i \in U$ and a negative rating $r_{ij} \in \{1, 2\}$.

H. Product rating deviation

Influencing the average product rating might affect customers' buying decisions and future ratings. It is therefore of interest to measure the absolute rating deviation of a given product $p_j \in P$:

$$x_{\text{PRD}}(e_{ij}) = |r(e_{ij}) - \frac{1}{|E_{*j}|} \sum_{e_{kj} \in E_{*j}} r(e_{kj})| \tag{23}$$

with $e_{ij} \in E$.

I. Average rating deviation

Elaborating on x_{PRD} the rating deviation of a selected user is

$$x_{\text{ARD}}(u_i) = \frac{1}{|E_{i*}|} \sum_{e_{ij} \in E_{i*}} x_{\text{PRD}}(e_{ij}) \tag{24}$$

with $u_i \in U$. x_{ARD} explains how much the user's opinion differs on average from the user's reviewed products.

J. Weighted rating deviation

Equation 28 serves as an indicator of a review's impact, and equation 23 describes the deviation of the user's opinion on the average of a single product. Both notions are combined in the weighted rating deviation.

Initially, the weights are defined as:

$$w(e_{ij}) = \frac{1}{x_{\text{RBD}}(e_{ij})^{\alpha}} \tag{25}$$

With $\alpha = 1.5$ that is utilized as a decay rate. Next, the weighted rating deviation is given by:

$$x_{\text{WRD}}(u_i) = \frac{\sum_{e_{ij} \in E_{i*}} x_{\text{PRD}}(e_{ij}) w(e_{ij})}{\sum_{e_{ij} \in E_{i*}} w(e_{ij})} \tag{26}$$

with $u_i \in U$. x_{WRD} measures how much a user $u_i \in U$ tends to differ on average ratings of sparsely reviewed products.

K. Entropy of rating distribution

Simple bot implementations do repeated actions over a given set of products. To catch this pattern, the entropy of rating distribution gets calculated:

$$x_{\text{ERD}}(u_i) = -\sum_{r=1}^{5} rr(u_i, r) \log_2 rr(u_i, r) \tag{27}$$

with $u_i \in U$. The entropy rating distribution measures the user's rating *sophistication*. If x_{ERD} is low, then u_i tends to give the same rating on each review.

L. Rank by date

A single review significantly affects the product's perception if only a tiny amount of other reviews are present. The review's $e_{ij} \in E$ rank of a product $p_j \in P$ by date is given by:

$$x_{\text{RBD}}(e_{ij}) = 1 + \sum_{e_{kj} \in E_{*j}} [d(e_{kj}) \leq d(e_{ij})] \tag{28}$$

with $e_{ij} \in E$. The rank helps differentiate between reviews that were dropped in a bucket for a highly reviewed product and the reviews that significantly shaped the average rating.

M. Maximum number of reviews per day

As spammers are more effective by writing more reviews, it is in the spammer's interest to write many reviews daily. Therefore, the maximum number of reviews written per day is captured.

First, given a review and a day, the number of reviews written on that day is counted by:

$$nr(e_{ij}) = \sum_{e_{il} \in E_{i*}} [d(e_{il}) = d(e_{ij})] \tag{29}$$

with $e_{ij} \in E$. Consequently, the maximum number of reviews that a user has written on a day is given if

$$x_{\text{MNR}}(u_i) = \max_{e_{ij} \in E_{i*}} nr(e_{ij}) \tag{30}$$

with $u_i \in U$.

N. Entropy of temporal gaps

Similar to the idea of x_{ERD}, the entropy of temporal gaps detects unvarying behavior. The set of reviews that a user has written after a specific review is constructed by:

$$D_{e_{ij} \leq E_{i*}} = \{e_{il} \mid e_{il} \in E_{i*}, d(e_{il}) \leq d(e_{ij}), e_{il} \neq e_{ij}\} \tag{31}$$

with $e_{ij} \in E$. With $D_{e_{ij} \leq E_{i*}}$ the a temporal gap is given by:

$$gap(e_{ij}) = \begin{cases} 0 & \text{if } D_{e_{ij} \leq E_{i*}} = \{\} \\ \min\limits_{e_{il} \in D_{e_{ij} \leq E_{i*}}} d(e_{ij}) - (e_{il}) & \text{otherwise} \end{cases} \tag{32}$$

with $e_{ij} \in E$. Naturally, the entropy of temporal gaps of a given user $u_i \in U$ are defined as:

$$x_{\text{ETG}}(u_i) = -\sum_{e_{ij} \in E_{i*}} gap(e_{ij}) \log_2 gap(e_{ij}) \tag{33}$$

with $u_{ij} \in U$. If x_{ETG} is low the user u_i posts reviews in a constant time interval.

O. Burstiness

Spammers often submit reviews in a short time frame and then become inactive[32].

Consequently, short-term members get flagged by:

$$x_{\text{BRT}}(u_i) = \begin{cases} 0 & \text{if } \min\limits_{e_{ij} \in E_{i*}} d(e_{ij}) - \max\limits_{e_{ij} \in E_{i*}} d(e_{ij}) > \tau \\ 1 - \frac{\min\limits_{e_{ij} \in E_{i*}} d(e_{ij}) - \max\limits_{e_{ij} \in E_{i*}} d(e_{ij})}{\tau} & \text{otherwise} \end{cases} \tag{34}$$

where $\tau = 28$ and $u_i \in U$. If a user has posted a review after 28 days u_i the first review, the user is a long-term member.