UNIVERSITÀ DI PISA

MSc in Artificial Intelligence and Data Engineering

Data Mining and Machine Learning

Fake reviews detection on Amazon dataset

Gianmaria Saggini

A.A. 2023/2024

Contents

1 Introduction

In the digital age, online reviews play a crucial role in shaping consumer decisions and influencing market trends. Among the ecommerce platforms, Amazon stands out as a major player, with billions of customer reviews contributing to the purchasing process.

User-oriented online reviews serve as the second most reliable source of product information, after recommendations from family and friends. Moreover, 80% of customers change their purchasing decisions after reading negative reviews, while 87% approve their decisions after reading positive reviews.

However, the integrity of these reviews is increasingly under scrutiny due to the rising prevalence of fake reviews. For instance, 60% of reviews for top 10 electronic products on Amazon are fake. Fake reviews can not only manipulate product rankings, but more widely reduce consumer trust in online reviews.

The objective of this project is to develop a robust framework for detecting and mitigating fake reviews on Amazon, using traditional methods of text vectorization and more advanced methods such as word embeddings.

2 Methodology

2.1 Dataset

I used a labeled Amazon dataset containing 21000 reviews, half fake and half genuine. Labeling was done by Amazon. Features present in the dataset are the following:

- **Label**. *label1* means fake, while *label2* means genuine. Labeling was conducted by the Amazon filtering algorithm embedded in the Amazon website.
- **Rating**. Review rating left by the customer.
- **Verified purchase**. Whether the customer that wrote the review really purchased the product. A multitude of studies reported that verified_purchase plays a fundamental role in the identification of fake reviews, as the majority of them are not verified.
- **Product category**. The dataset contains 30 different product categories, each of them consisting of 700 reviews (half fake and half genuine).
- **Product id**.
- **Product title**.
- **Review title**.
- **Review text**.

Considering that the dataset was created by Amazon, there are no null or missing values.

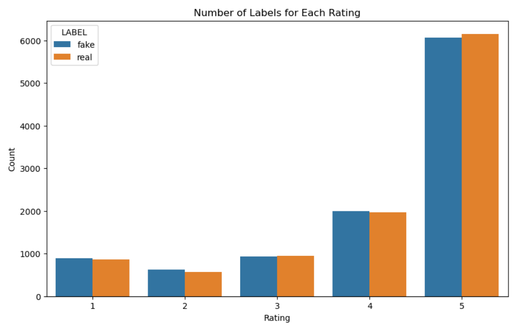
Another unlabeled dataset, still from Amazon, was used to train a Word2Vec model. This dataset contains approximately half a billion of reviews, from 1996 to 2023. Further details regarding the training process and implementation of the Word2Vec model can be found in the **Word Embeddings** section.

2.2 Exploratory Data Analysis (EDA)

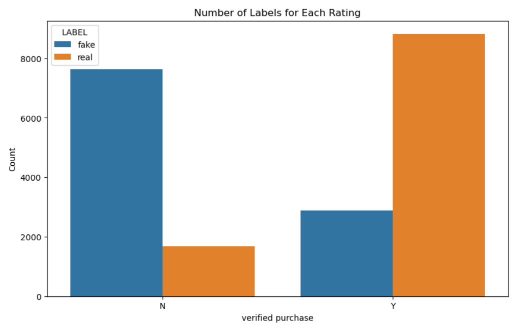
Firstly, I analyzed the distribution of fake and real reviews in various attributes. This can help to decide which features to use for classification.

In contrast to researches, I did not include the review rating in the features. In fact, as we can see in Figure 1a, review rating doesn't help in recognizing fake reviews. Whereas according to researches, verified purchase help enormously in this task. Figure 1b shows how fake reviews are mostly of unverified purchases.

Based on other studies, real reviews are lengthier than fake ones, since spammers don't want to devote too much time to writing. However, I wanted to test this hypothesis before inserting the length of the review as a feature. I plotted the frequency of words for fake and real reviews, and the same but for review length,



(a) Number of fake and real reviews for ratings.

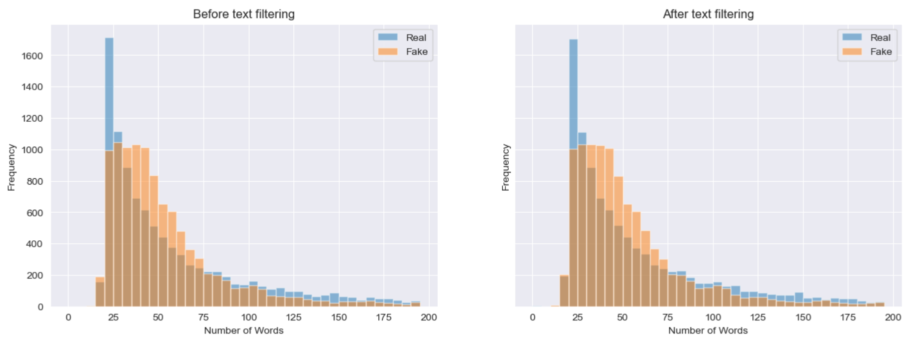


(b) Number of fake and real reviews for verified purchases and not.

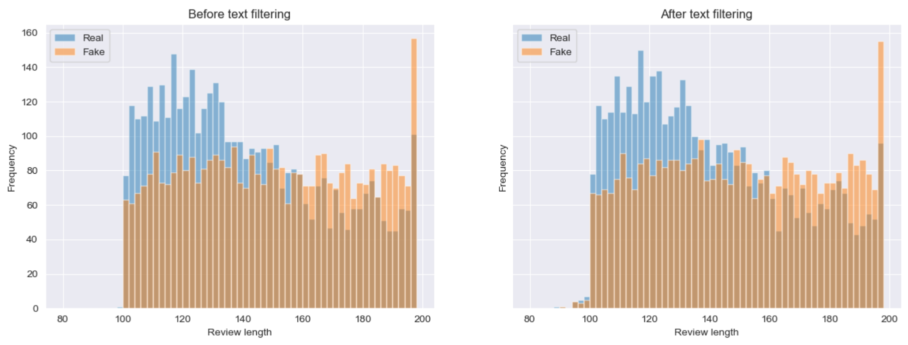
both before and after text filtering (remove URLs and HTML tags). As we can see in Figure 2a and Figure 2b, there isn't a significant difference before and after text filtering.

Most importantly, the hypothesis suggested by other studies is only partially true. As shown in Figure 2a, it's true that on the average fake reviews are shorter, but this is false for the median. In fact, most of real reviews are about 20 words. Similarly, in Figure 2b, fake reviews are distributed evenly whereas real ones are on average shorter.

Because there isn't much difference between the distributions of real and fake reviews, I decided to omit these features in the model evaluation.



(a) Number of fake and real reviews for ratings.



(b) Number of fake and real reviews for verified purchases and not.

Figure 2: Text features before and after text parsing.

Lastly, I plotted a correlation matrix of those four features, shown in Figure 3. It is noticeable that there is no difference before and after text filter, and also between number of words and review length.

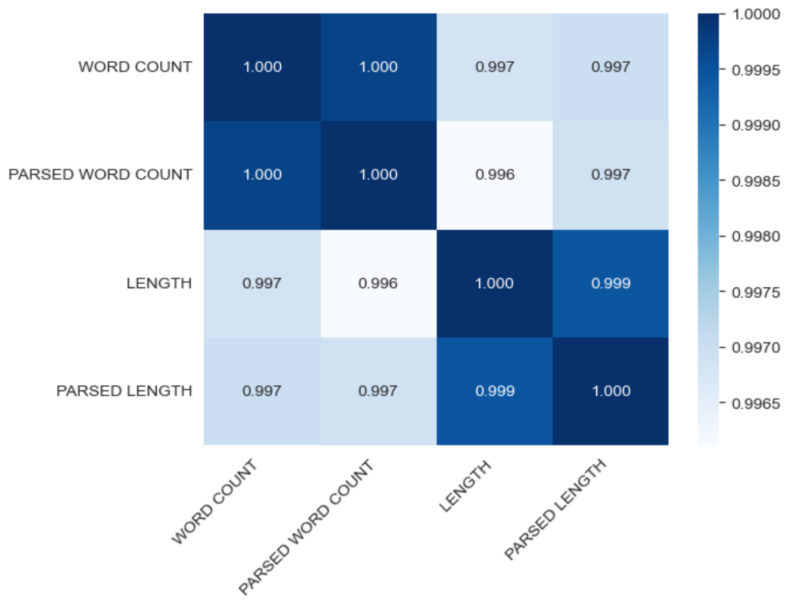


Figure 3: Correlation matrix

2.3 Text preprocessing

The preprocessing phase is divided in the following steps:

- **Filtering**. Text is converted to lowercase. Moreover, URLs and HTML tags are removed.
- **Tokenization**. Using *RegexpTokenizer* from nltk, the text is divided into tokens that contain only letters.
- **Stopword removal**. Stopwords are removed using the nltk english stopwords list.
- **Stemming**. After tokenization and stopword removal, each token is passed through the Snowball Stemmer, which reduces words to their root form. A stemmer was used instead of a lemmatizer for simplicity and computational speed. For product reviews like Amazon reviews, which often contain informal or non-standard language, stemming tends to be a good enough approximation, while also being faster to implement.

To ensure coherence, these steps are applied to each text used, both from the labeled and unlabeled datasets.

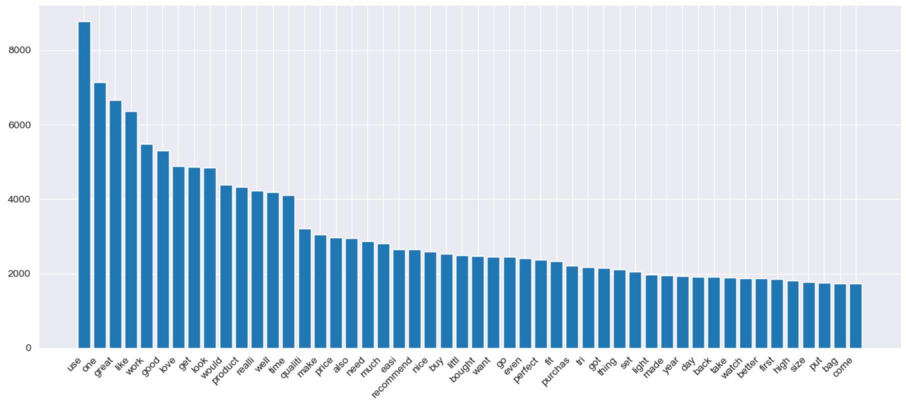
2.4 Feature extraction

Both Bag of Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF) were used to vectorize the reviews.

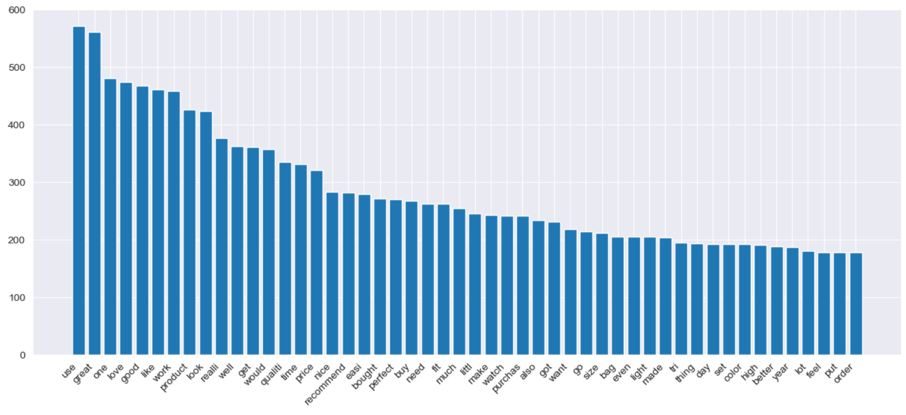
BoW was used to create a matrix of word frequencies across the entire corpus of reviews. Common words such as *"use"*, *"one"*, *"great"* appear frequently, reflecting the general nature of product-related discussions in the dataset. However, it may overemphasize common terms that do not carry significant discriminative power between genuine and fake reviews.

To address the limitations of BoW, the TF-IDF approach was also employed. The Term Frequency (TF) component measures how frequently a word appears in a review, while the Inverse Document Frequency (IDF) down-weights words that appear across many reviews, emphasizing words that are more unique to specific reviews. This helps to reduce the impact of overly common terms that might not be useful.

Figure 4a shows the fifty most frequent words in the BoW vectorization, while Figure 4b shows the fifty most frequent words in TF-IDF.



(a) top 50 words in Bag of Words.



(b) top 50 words in TF-IDF.

Figure 4: Most frequent words for both vectorizers.

2.5 Word embeddings

In this project, word embeddings were generated using the Skip-Gram variant of Word2Vec model. Word2Vec is a shallow neural network architecture designed to capture semantic relationships between words by learning distributed representations. That is, it maps a word into a vector such that similarity in the word representation is retained.

The model was trained on the unlabeled dataset of half a billion reviews. Firstly, the dataset was preprocessed, removing reviews after 2018. This was done because the main labeled dataset was made in 2018. Afterward, a file with a sentence per line was created. To speed up computation, the *gensim* library offer the possibility to parallelize the work. To do so, it needs as input a file with sentences already tokenized per line. In this manner, the file is not loaded entirely in memory.

Among the multitude of hyperparameters, there are a few most important: window, min_count, vector_size. From previous studies, the optimal window and vector_size are 5 and 100 respectively. The min_count is the minimum frequency of a word to be taken into consideration. To decide it, I plotted the frequency of the 1000 most frequent words, as shown in Figure 5. I decided to try both with 500000 and 700000 as min_count values, and see after the impact of this parameter.

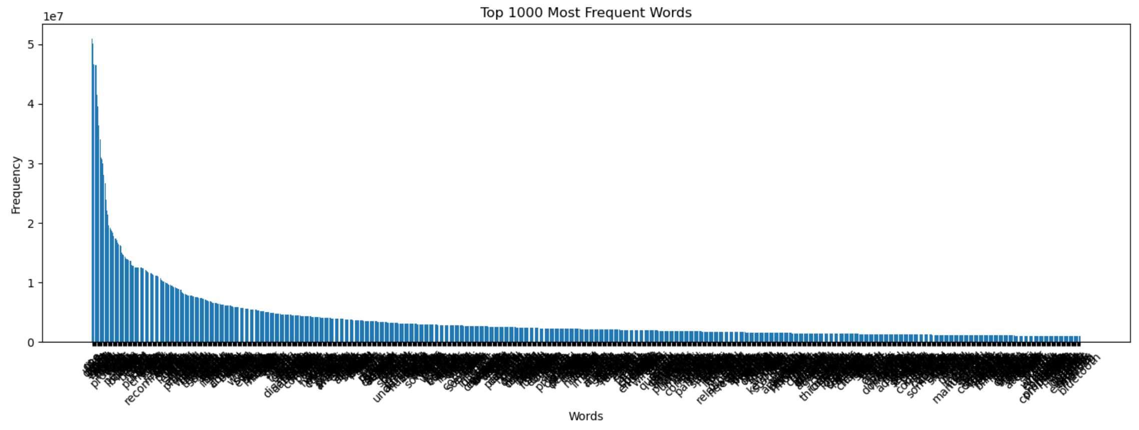
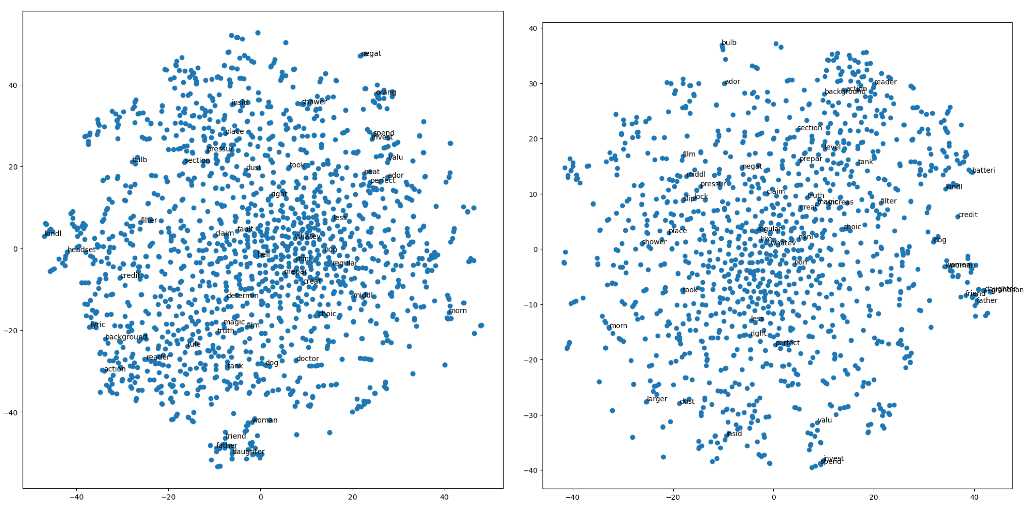


Figure 5: 1000 most frequent words in tokenized unlabeled dataset.

In Figure 6 are shown the visualizations of the two models by reducing dimensionality of the words to 2 dimensions using tSNE (t-distributed stochastic neighbor embedding).



(a) min_count=500000 (b) min_count=700000

Figure 6: Projection onto 2 dimensions of word embeddings.

3 Experimental results

According to researches, **Random Forest (RF)** is the most performant model. In addition to that, I also evaluated **linear SVC (SVC)**, which had excellent results, and also scales better with larger datasets.

For the sake of computation efficiency, I selected 12 main features combinations for comparison. Each combination has BoW or TF-IDF as a vectorization. Then, I wanted to explore the effect of verified purchase and word embeddings on the results. WE1 stands for the model with *min_count=700000*, while WE2 for *min_count=500000*. The set of feature matrices is shown in Table 1.

Feature matrix	Abbreviation
Bag of Words	BoW
Bag of Words, verified purchase	BoW+VP
Bag of Words, word embeddings 1	BoW+WE1
Bag of Words, word embeddings 2	BoW+WE2
Bag of Words, word embeddings 1, verified purchases	BoW+WE1+VP
Bag of Words, word embeddings 2, verified purchases	BoW+WE2+VP
TF-IDF	TF-IDF
TF-IDF, verified purchase	TF-IDF+VP
TF-IDF, word embeddings 1	TF-IDF+WE1
TF-IDF, word embeddings 2	TF-IDF+WE2
TF-IDF, word embeddings 1, verified purchases	TF-IDF+WE1+VP
TF-IDF, word embeddings 2, verified purchases	TF-IDF+WE2+VP

Table 1: Used features combinations to evaluate models

3.1 Hypertuning

Nested cross-validation was used, using *StratifiedKFold* for both the inner and outer loop. In the inner loop I used 5-fold validation, whereas in the outer one I decided to use 10-fold.

For each fold, **HalvingGridSearch** was applied. HalvingGridSearch introduces an iterative approach where it begins by evaluating a large number of hyperparameter combinations on a smaller subset of the data. After this initial round, it "halves" the number of combinations, keeping only the best-performing ones, and trains the remaining combinations on a larger subset of the data. This is done iteratively until only one combination remain.

The hyperparameters used are resumed in Table 2.

Model	Hyperparameter	Values
SVC	C	[0.5, 1, 1.5]
	max_iter	[2000, 4000]
	loss	[hinge, squared_hinge]
RF	n_estimators	[800, 1000, 1200]
	criterion	[entropy, log_loss]
	max_depth	[20, 40, 60, None]
	bootstrap	[False, True]

Table 2: Used features combinations to evaluate models

3.2 Results

Results are shown in Table 3. For readability, not all 24 tests are reported, but only the most interesting. All results can be found in the *results.txt* file.

As we can see, Random Forest always performed better than linear SVC. As expected, verified purchase increment noticeably the accuracy of the models. TF-IDF seems to be slightly better than Bag of Words. Word embeddings 1 (min_count=700000) didn't seem to be always better than word embeddings 2 (min_count=500000).

3.3 Statistical comparison

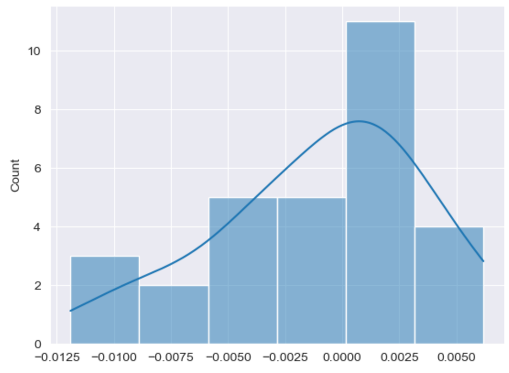
To assess which model is the best, a statistical comparison should be conducted between each pair of models. To have a significant comparison, I should perform more than one 10-fold cross-validation for each test. Since it is computationally heavy, I decided to compare only the two models. The most interesting comparison in my opinion was the one between TF-IDF+VP+WE1 and TF-IDF+VP both with Random Forest, to see if word embeddings contribute to the accuracy. To make the comparison more significant, I performed three 10-fold cross-validation.

Features	Model	Accuracy	F1 score	AUC
BoW	RF	63.81%	66.04%	70.17%
	SVC	61.71%	63.62%	66.32%
BoW+VP	RF	80.53%	79.22%	**87.10**%
	SVC	79.25%	78.97%	84.58%
BoW+WE1	RF	63.51%	65.13%	68.72%
	SVC	62.25%	64.07%	66.96%
BoW+WE2	RF	63.12%	64.85%	68.71%
	SVC	62.29%	64.13%	67.03%
BoW+VP+WE1	RF	80.45%	79.68%	86.51%
	SVC	78.87%	78.63%	84.43%
BoW+VP+WE2	RF	80.54%	79.73%	86.66%
	SVC	78.65%	78.44%	84.24%
TF-IDF	RF	64.60%	65.91%	70.93%
	SVC	64.17%	64.37%	69.58%
TF-IDF+VP	RF	80.68%	79.36%	86.94%
	SVC	80.62%	**80.27**%	86.40%
TF-IDF+WE1	RF	63.95%	64.79%	69.75%
	SVC	64.06%	64.31%	69.67%
TF-IDF+WE2	RF	63.88%	64.97%	69.88%
	SVC	64.08%	64.37%	69.66%
TF-IDF+VP+WE1	RF	**80.75**%	80.11%	86.81%
	SVC	80.54%	80.23%	86.44%
TF-IDF+VP+WE2	RF	80.74%	80.10%	86.83%
	SVC	80.39%	80.07%	86.43%

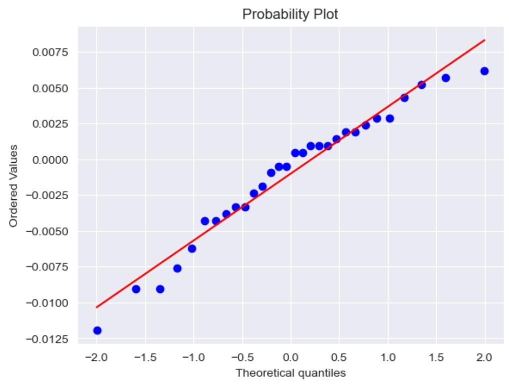
Table 3: Results obtained.

First, I verified if the distribution was normal, using Q-Q plots and histograms, as shown in Figure 7. As we can see, the differences didn't seem to be normally distributed, so I opted for a Wilcoxon signed-rank test.

The Wilcoxon test showed a p-value of 0.44, thus the Null hypothesis cannot be rejected. This means that word embeddings don't bring a tangible contribute to the model. Further improvements and tuning of the Word2Vec model could be done to see if there would be an improvement in performances.



(a) Histogram

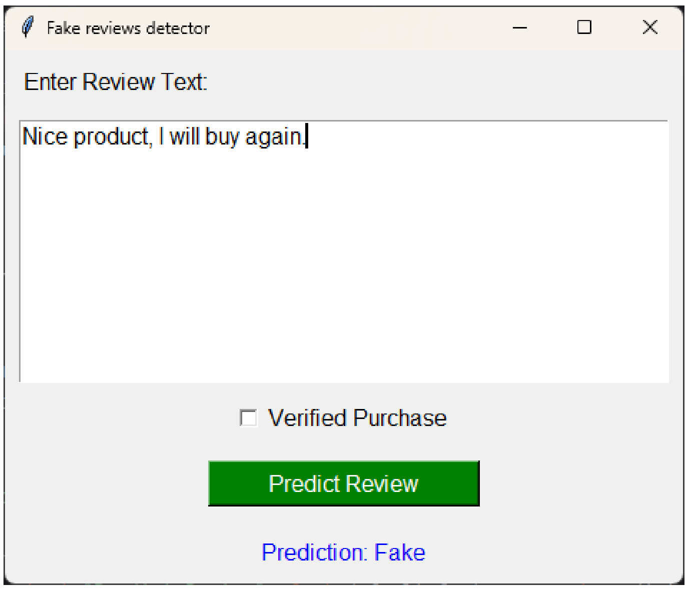


(b) Q-Q plot.

Figure 7: Normality distribution tests.

4 Interface

To test the model with the best accuracy, I created a simple interface in Python using the tkinter library. The interface allows writing a review and decide if the purchase is verified or not. Clicking on *Predict review*, the interface first generates the word embeddings and then calls the predict method on the saved model. The graphic interface is shown in Figure 8.


Fake reviews detector

Enter Review Text:

Nice product, I will buy again

☐ Verified Purchase

Prediction: Fake

Figure 8: Graphical interface.