UNIVERSITÀ DI PISA

Artificial Intelligence and Data Engineering

Business and Project Management

Amazon Reviews Classifier

Project Documentation

AUTHOR:
Leonardo Bargiotti

Academic Year: 2022/2023

Contents

List of Figures

1 — Introduction

Unstructured data in the form of text: chats, emails, social media, survey responses is present everywhere today. Text can be a rich source of information, but it can be hard to extract insights from it. Text classification is one of the important task in supervised machine learning (ML). It is a process of assigning tags/categories to documents helping us to analyze automatically text in a cost-effective manner. It is one of the fundamental tasks in Natural Language Processing with broad applications such as sentiment-analysis, spam-detection, topic-labeling, intent-detection etc.

This project exploits text mining in order to automatically categorize news articles to their right topic (for example sport, business, world and sci/tech). This application could be used to classify text of other subject (not only news), depending on which dataset is given in input. [1]

1.1 Goals

The aim of this paper is to explain the choices and the strategies adopted on the project and development of **Reviews Classifier**. In order to accomplish it, the first step is perform preprocess for having a suitable dataset and then it is used several classifiers, to determinate which predicts the right class.

1.2 Initial Dataset

In order to realize this application is used this dataset: **Amazon Reviews** [2]

The origianl dataset is composed by 3600000 rows in training set and 400000 in testset. It is balanced and it has two classes *1, 2*, class 1 is the negative and class 2 is the positive. It has three columns: one relative to the class of the review, the second is the title of the review and the last one is the review. For this application the title is not used, but it can be set as *Column Text* in configuration file.

The original dataset is very large, so it is deicided to make a new one smaller than the original. In partcicular the new has 200000 rows in training set and 100000 in testset.

[1] GitHub repository for the project: `https://github.com/leobargiotti/amazon_reviews_classifier`

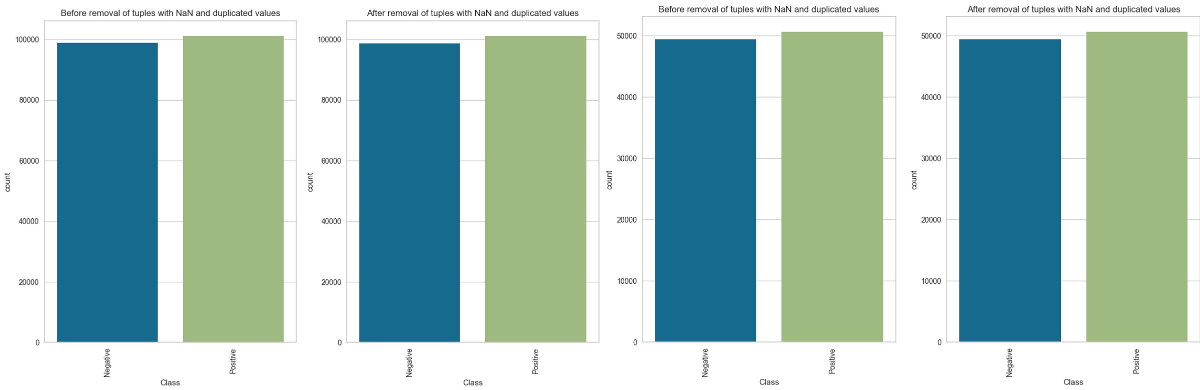
[2] Link for Dataset from Kaggle: `https://www.kaggle.com/datasets/kritanjalijain/amazon-reviews?select=train.csv`

2 — Preprocessing

Before building models, is necessary preprocess dataset in order to remove first of all missing values, duplicates and then clean the text. The entire process is shown in this chapter.

2.1 Removing Duplicates and Missing Values

The first thing is remove duplicate rows and those contain missing values. In the images below is possible to see that after removing values the new datasets have the same class distribution and the number of elements of each class are quiet the same that original.



(a) Training Set **(b)** Test Set

Figure 1: Class Distribution

2.2 Text Preprocess

In order to perform computational tasks on text, is need to convert the language of text into a language that the machine can understand. In particular text cleaning is composed by following steps:

- Normalization
- Stemming
- Lemmatization

2.2.1 Normalization

One of the key steps in processing language data is to remove noise so that the machine can more easily detect the patterns in the data. Text data contains a lot of noise, this takes the form of special characters (such as URLs, HTML tags, diacritics, extra white spaces), punctuation and numbers. Additionally, it is also important to apply some attention, if text includes both upper case and lower case versions of the same words then the computer will see these as different entities. To avoid this problem is enough transform all words in text to lowercase. The python library used to implement this steps is *texthero*[3]. Another important phase is removing stop-words, it is list of generic words for example *'i', 'you', 'a', 'the', 'he', 'which' etc.* for the English vocabulary. The list of stop-words used is the default in *nltk library*[4]. There are another feature

[3] Link for Texthero library: `https://texthero.org`
[4] Link for Nltk library: `https://www.nltk.org`

only available for English text, is to write abbreviations in their long forms and slangs in to the correct form, using *contractions library*[5].

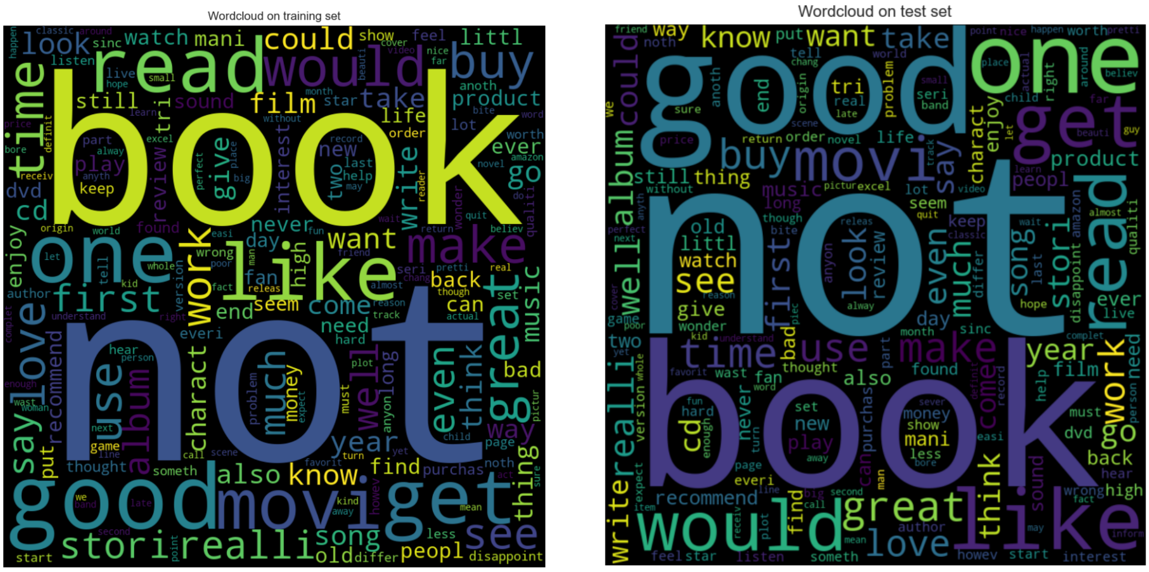
2.2.2 Stemming

Stemming is the process of reducing words to their root form. For example, the words *'rain', 'raining' and 'rained'* have very similar, and in many cases, the same meaning. The process of stemming will reduce these to the root form of *'rain'*. This is a way to reduce noise and the dimensionality of data. To implement this process is used *texthero library*, used in the previous step.

2.2.3 Lemmatization

The goal of lemmatization is the same as for stemming, in that it aims to reduce words to their root form. However, stemming is known to be a fairly crude method of doing this. Lemmatization, on the other hand, is a tool that performs full morphological analysis to more accurately find the root. To implement this process is used *simplemma library*[6].

After apply this process to the original datasets these are the results:



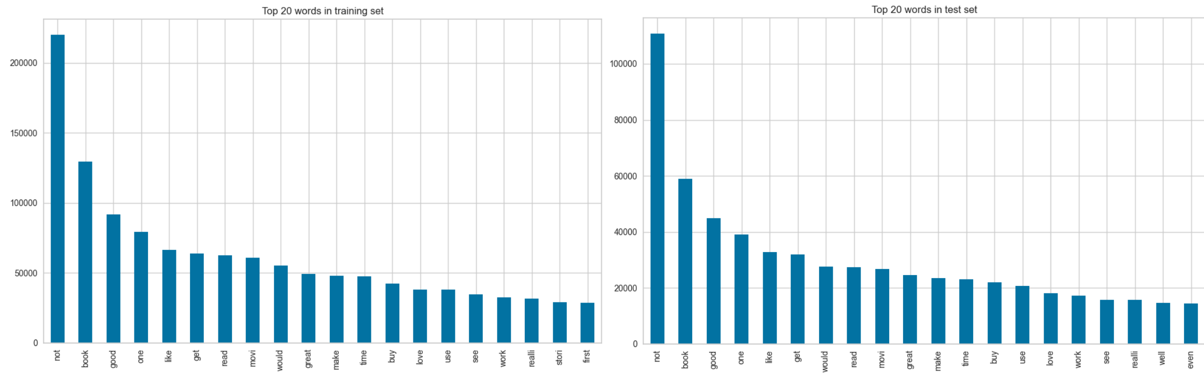
(a) Training Set

(b) Test Set

Figure 2: Wordcloud

[5]Link for Contractions library: `https://libraries.io/pypi/contractions/0.1.73`

[6]Link for Simplemma library: `https://libraries.io/pypi/simplemma`



(a) Training Set **(b)** Test Set

Figure 3: Top 20 Words on Dataset


```
Before Preprocess:
The first rows of training set are:
0    I'm reading a lot of reviews saying that this ...
1    This soundtrack is my favorite music of all ti...
2    I truly like this soundtrack and I enjoy video...
3    If you've played the game, you know how divine...
4    I am quite sure any of you actually taking the...
Name: Text_Complete, dtype: object

The first rows of training set are:
0    2
1    2
2    2
3    2
4    2
Name: Class, dtype: int64

The first rows of test set are:
0    Despite the fact that I have only played a sma...
1    I bought this charger in Jul 2003 and it worke...
2    Check out Maha Energy's website. Their Powerex...
3    Reviewed quite a bit of the combo players and ...
4    I also began having the incorrect disc problem...
Name: Text_Complete, dtype: object

The first rows of test set are:
0    2
1    1
2    2
3    2
4    1
Name: Class, dtype: int64

The class distribution on training set is
2    101166
1     98834
Name: Class, dtype: int64

The class distribution on test set is
2    50579
1    49421
Name: Class, dtype: int64

The numbers of Nan value on training set are 0

The numbers of Nan value on test set are 0

The numbers of duplicate elements on training set are 133

The numbers of duplicate elements on test set are 11

There are 200000 rows in the training set

There are 100000 rows in the test set
```

```
After Preprocess:
The first rows of training set are:
0    read lot review say good game soundtrack figur...
1    soundtrack favorit music time hand intens sad ...
2    truli like soundtrack enjoy video game music p...
3    play game know divin music everi singl song te...
4    quit sure actual take time read play game less...
Name: Text_Complete, dtype: object

The first rows of training set are:
0    2
1    2
2    2
3    2
4    2
Name: Class, dtype: int64

The first rows of test set are:
0    despit fact play small portion game music hear...
1    buy charger jul work ok design nice conveni ho...
2    check maha energi websit powerex mh c f charge...
3    review quit bite combo player hesit due unfavo...
4    also begin incorrect disc problem read vcr sti...
Name: Text_Complete, dtype: object

The first rows of test set are:
0    2
1    1
2    2
3    2
4    1
Name: Class, dtype: int64

The class distribution on training set is
2    101133
1     98725
Name: Class, dtype: int64

The class distribution on test set is
2    50576
1    49412
Name: Class, dtype: int64

The numbers of Nan value on training set are 0

The numbers of Nan value on test set are 0

The numbers of duplicate elements on training set are 0

The numbers of duplicate elements on test set are 0

There are 199858 rows in the training set

There are 99988 rows in the test set
```

Figure 4: Before and After Preprocess

3 — Classification

After the preprocessing phase, the dataset is ready to be used to learn classification models that will be used in the final application. In this chapter are discussed the chosen strategies relative to classification phase. In this project is used to different approach to classify, the first is the traditional and the other one is using pre-trained NLI models as a ready-made *zero-shot sequence* classifiers.

Before apply traditional classifiers is necessary to transform text into vector of real numbers, which is the format that ML models support. The process to convert text data into numerical data/vector, is called vectorization.

3.1 Vectorization

The solution used to implement vectorization is **Term Frequency-Inverse Document Frequencies** *(TF-IDF)* [7]. It is a numerical statistic that's intended to reflect how important a word is to a document. Words that get repeated too often don't overpower less frequent but important words. It is composed by two parts:

1. **Term Frequency** *(TF)*. It can be understood as a normalized frequency score and it is always ≤ 1. It is calculated via the following formula:

$$TF = \frac{Frequency\,of\,word\,in\,a\,document}{Total\,number\,of\,words\,in\,that\,document}$$

2. **Inverse Document Frequency** *(IDF)*, but before is necessary make sense of *DF – Document Frequency*. It's given by the following formula:

$$DF\,(word) = \frac{Number\,of\,documents\,with\,word\,in\,it}{Total\,number\,of\,documents}$$

It measures the proportion of documents that contain a certain word. *IDF* is the reciprocal of the Document Frequency, and the final IDF score comes out of the following formula:

$$IDF\,(word) = \log\left(\frac{Total\,number\,of\,documents}{Number\,of\,documents\,with\,word\,in\,it}\right)$$

The intuition behind it is that the more common a word is across all documents, the lesser its importance is for the current document. A logarithm is taken to dampen the effect of IDF in the final calculation. The final *TF-IDF* score comes out to be:

$$TF - IDF = TF \cdot IDF$$

The higher the score and more important that word is. Basically, the value of a word increases proportionally to count in the document, but it is inversely proportional to the frequency of the word in the corpus.

3.2 Traitional Classifiers

This section is about the training of the machine learning models on the vectorized dataset. To minimize lengthy re-training and allow you to share, commit, and re-load pre-trained machine learning models is used *Dill library* [8], that is a useful Python tool that allows to save ML

[7] Link for TfidfVectorizer library: `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`

[8] Link for Dill library: `hhttps://libraries.io/pypi/dill`

models. Every times that application starts, it controls if models are present in *models_saved* directory. If a model is present, it is being loaded otherwise it is being trained and saved in *models_saved* folder.

To find the optimal parameters from the chosen classifiers is performed a technique called *GridSearchCV* [9]. The performance of a model significantly depends on the value of hyperparameters. There is no way to know in advance the best values for hyperparameters so ideally, is necessary to try all possible values to know the optimal values. Doing this manually could take a considerable amount of time and resources and thus a solution is use GridSearchCV to automate the tuning of hyperparameters.

In this application the chosen classifier is:

- *Logistic Regression*[10]

3.2.1 Hyperparameters

This section is focused on GridSearchCV and hyperparameters of its classifiers. The parameters of the estimator used are optimized by cross-validated using *StratifiedKFold*[11] over a parameter grid. The parameters grid for the classifier is:

- *Logistic Regression*
  - C : [100, 75, 50, 25 15, 10, 5, 3, 1, 0.1, 0.05, 0.01]
  - solver: $['liblinear', 'newton-cg']$

Here are reported the best parameters, best score and time to the train the classifier:

- *Logistic Regression*
  - best parameters
    - C: 3
    - solver: 'liblinear'
  - best score: 0.888
  - time: 828.418 seconds

The time to fit all classifiers is calculated using *Apple M1* processor.

[9] Link for GridSearchCV library: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`

[10] Link for Logistic Regression library: `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

[11] Link for StratifiedKFold library: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html`

3.3 Pre-trained Classifiers

In this section is specified one NLI model:

- *MoritzLaurer/DeBERTa-v3-base-mnli-fever-anli*[12]

The "DeBERTa V3" model represents an advancement in natural language understanding (NLU) achieved through training on diverse datasets, including MultiNLI, Fever-NLI, and Adversarial-NLI (ANLI), encompassing 763,913 NLI hypothesis-premise pairs. Built upon the DeBERTa-v3-base from Microsoft, this model surpasses the performance of many large models on the ANLI benchmark. DeBERTa, incorporating disentangled attention and an enhanced mask decoder, outperforms BERT and RoBERTa models across numerous NLU tasks using 80GB of training data. The V3 variant further enhances efficiency through ELECTRA-Style pre-training with Gradient Disentangled Embedding Sharing. Compared to its predecessor, DeBERTa V3 exhibits significant performance improvements on downstream tasks. The DeBERTa V3 base model features 12 layers, a hidden size of 768, and 86 million backbone parameters. The vocabulary includes 128,000 tokens, contributing to a total of 98 million parameters in the Embedding layer. Trained on 160GB of data, this model reflects the culmination of advancements detailed in the associated research paper. For additional technical details and updates, refer to the official repository.

[12]Link for HuggingFace library: `https://huggingface.co/MoritzLaurer/DeBERTa-v3-base-mnli-fever-anlil`

4 — Results

One of the main goals of this project was to compare the classifiers chosen in order to verify which one had best performances. To test the traditional classifier is used the entire test set and for the pre-trained models is used a subset composed by the first 1000 rows, beacuse the time complexity for the entire or a large dataset is too much great. In this chapter there is the description of the results obtained using these following statistics:

- *Classification Report*[13]
- *Confusion Matrix*[14]

4.1 Classification Report

Logistic Regression				
	Precision	Recall	F1-Score	Support
Negative	0.89	0.89	0.89	49412
Positive	0.89	0.90	0.89	50576
Accuracy			0.89	99988
Macro Avg	0.89	0.89	0.89	99988
Weighted Avg	0.89	0.89	0.89	99988

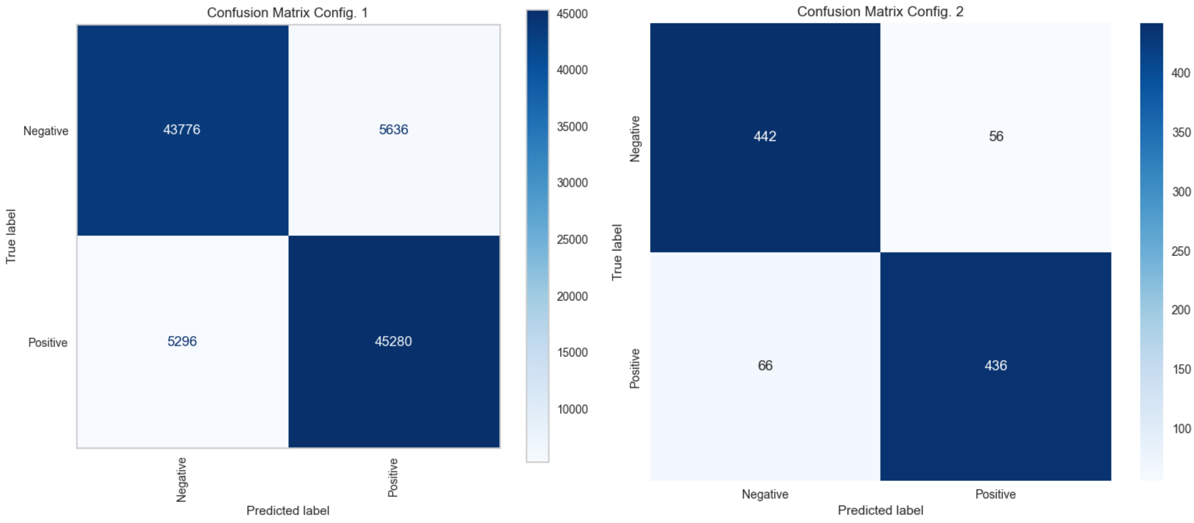
Final Training Accuracy: 94.47% Model Accuracy: 89.07%

DeBERTa				
	Precision	Recall	F1-Score	Support
Negative	0.87	0.89	0.88	498
Positive	0.89	0.87	0.88	502
Accuracy			0.88	1000
Macro Avg	0.88	0.88	0.88	1000
Weighted Avg	0.88	0.88	0.88	1000

[13]Link for Classification Report library: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html`

[14]Link for ConfusionMatrixDisplay library: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html`

4.2 Confusion Matrix



(a) Logistic Regression **(b)** DeBERTa

Figure 5: Confusion Matrix

5 — Application

Reviews Classifier is a simple application based on text mining in order to classify reviews. In *main.py* file is specified which classifier are used and their name, that are shown in the Application Home. It uses a configuration file *config.ini* to store some information about the configuration setting.

5.1 How to Install

Before use the application is necessary to install all library used using this command: *pip install -r requirements.txt*. After installing all libraries, the command to start application is *python ./src/main.py*. The application is tested with *python 3.8*.

5.2 How to classify text

Every user can operate with the applicative as they open it and insert into the textbox a news to classify.

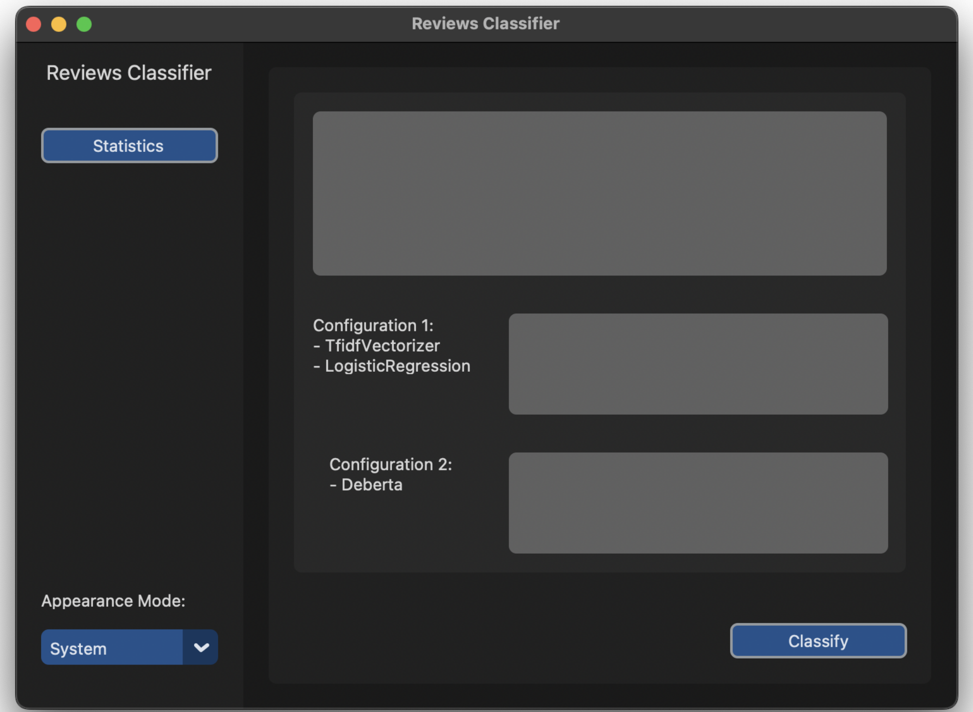


Figure 6: Home Application

1. *TextBox*: Input text to classify
2. *Prediction of classifiers*: Output text displays class predicted, by the first classifier, and its probability
3. *Classify*: Button to classify text inserted in *TextBox*
4. *Statistics*: Button to see all statistics on dataset and classifiers

5. *Appearance*: Option menu to change the appearance of the application (Dark, Light and System)

5.3 Display statistics

The image below describes all statistics in the application, each buttons display the corresponding statistics.

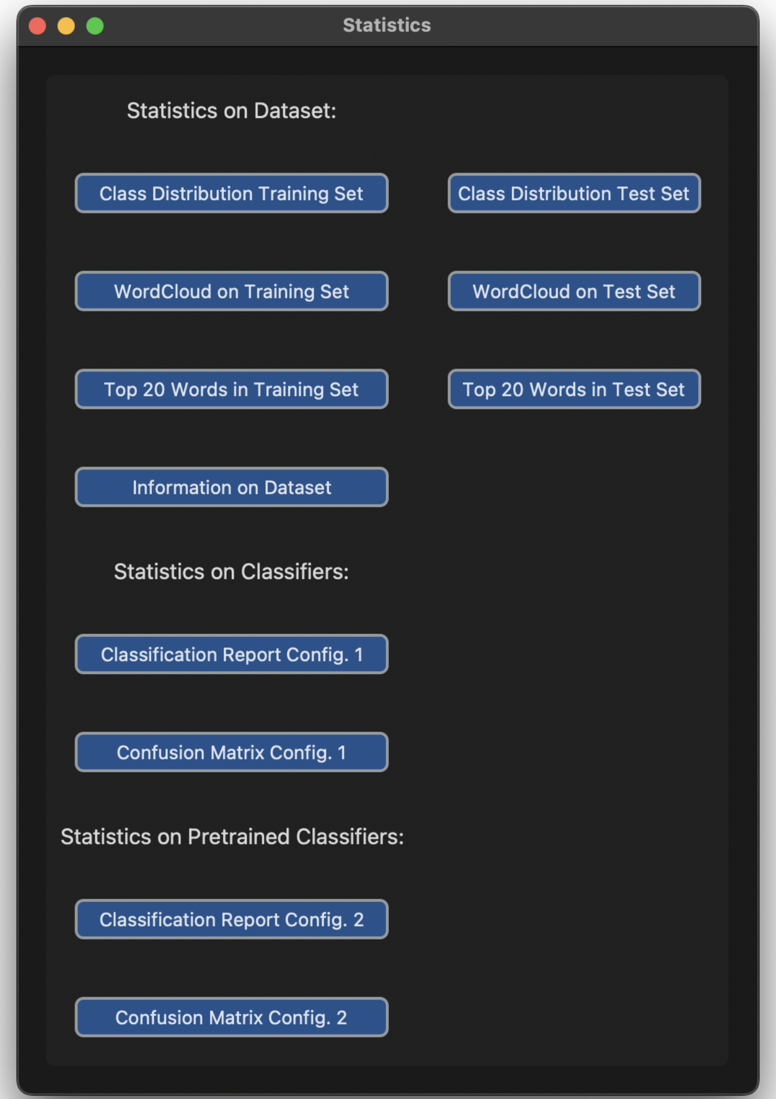
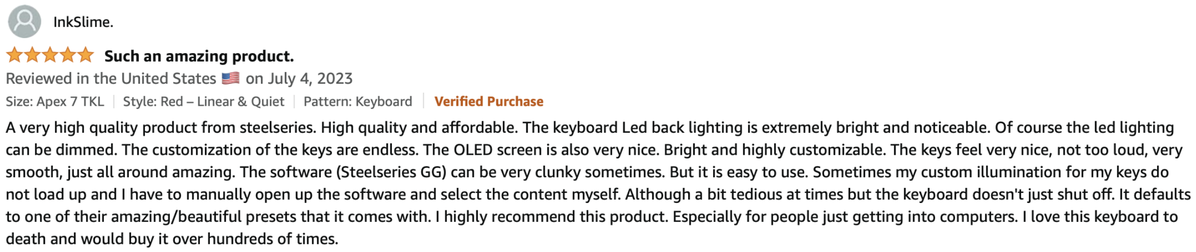


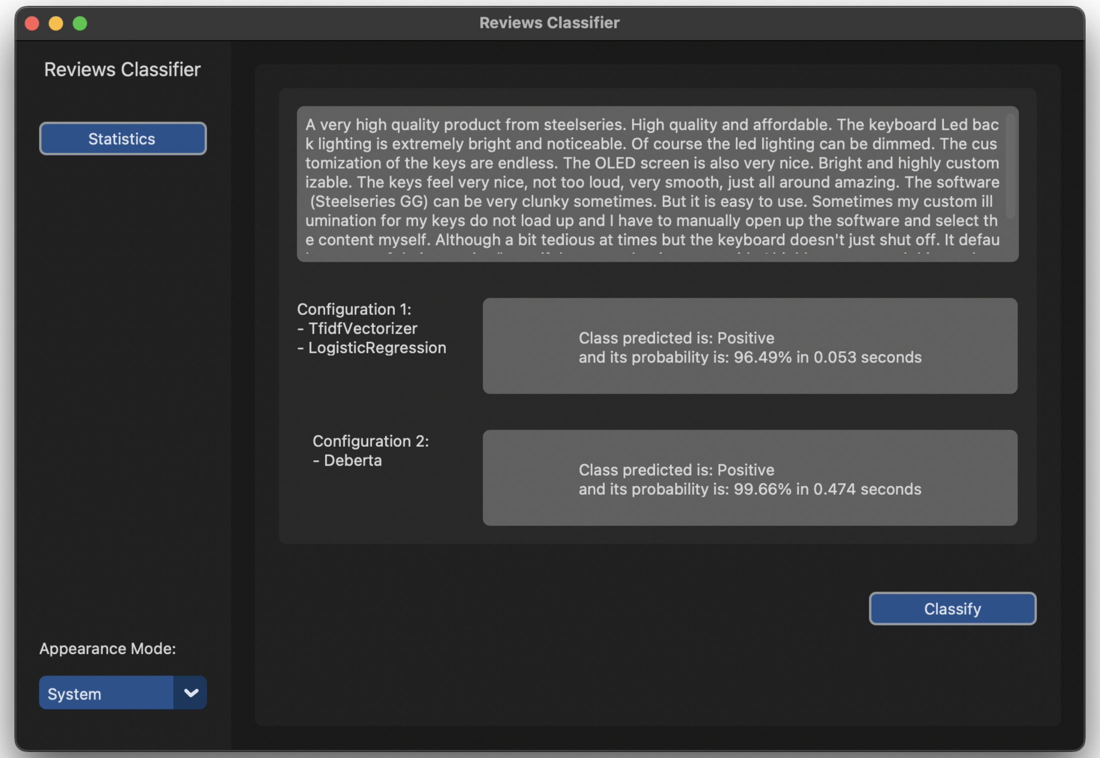
Figure 7: Statistics

5.4 Examples

Here are reported two example of Amazon reviews the first is positive and the second is negative.



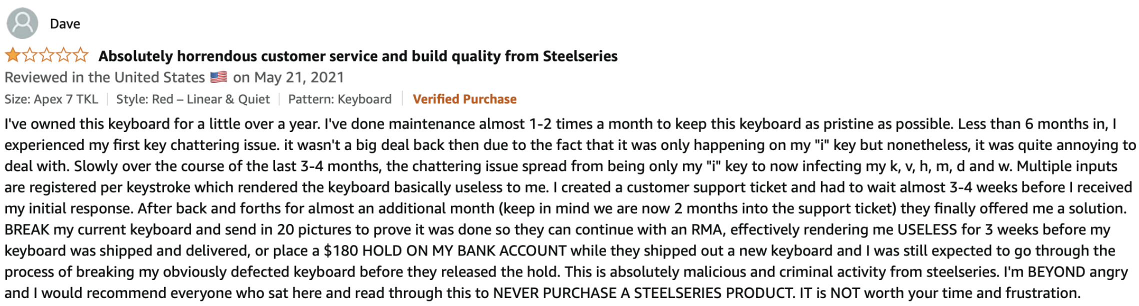
(a) Positive Amazon Review



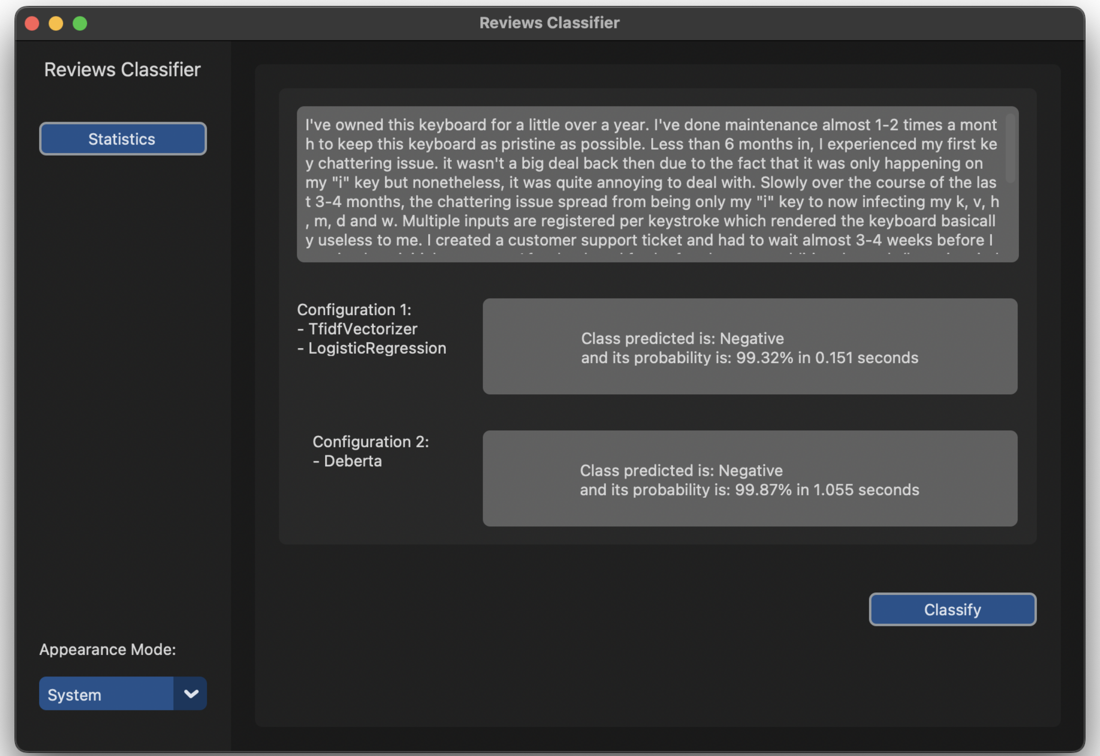
(b) Output Positive Amazon Review

Figure 8: Positive Example[17]

[17] Positive Amazon Review: `https://www.amazon.com/gp/customer-reviews/RGNEZISRST2D7/ref=cm_cr_getr_d_rvw_ttl?ie=UTF8&ASIN=B07TGQ7CNF`



(a) Negative Amazon Review



(b) Output Negative Amazon Review

Figure 9: Negative Example[18]

[18] Negative Amazon Review: `https://www.amazon.com/gp/customer-reviews/R3A53GGYD9NE07/ref=cm_cr_getr_d_rvw_ttl?ie=UTF8&ASIN=B07TGQ7CNF`

6 — Conclusion

This final chapter provides a comparison between the two models:

- **Time Complexity:** The pre-trained model exhibits significantly slower prediction times compared to traditional classifier.
- **Ease of Use:** Pre-trained model is more user-friendly than traditional classifiers because it eliminates the need for a training phase; it is ready to use.
- **Size:** The pre-trained model has a larger file size (380MB) compared to the normal classifier trained on the specific dataset (only 40MB).
- **Performance:** All two models demonstrate similar performance, with minimal differences observed between the two types (as detailed in the Results chapter).