UNIVERSITÀ DI PISA

Department of Information Engineering
Artificial Intelligence and Data Engineering course
Data mining and Machine Learning

Spoiler Detector

Application developed by
Lorenzo Massagli, Francesca Pezzuti

Academic Year 2021/2022

1 Introduction

The *SpoilerDetector* application is a simple application where users can search movies, read reviews written by other users and write reviews about movies. Since some reviews may contain spoilers (a description of an important plot development), we want to ensure that when the application shows a review to the user, if it may contain a spoiler, it should show a warning message to the user asking whether if (s)he confirms (s)he wants to read the full content of the review and only after his/her consent, it should show the full content.

2 Design

In this section is described the design of the application in terms of main actors, functional and non-functional requirements, data model and system architecture.

2.1 Requirements

2.1.1 Main Actors

The actors of *SpoilerDetector* are:

- unlogged user
- logged user
- admin
- classifier

2.1.2 Functional requirements

- through *registration* a unlogged user can become a registered user
- a registered user can login and become a logged user
- a logged user can logout from *SpoilerDetector*
- a logged user can browse movies
- a logged user can visit a movie's page
- a logged user can read the reviews about a movie
- a logged user can write a new review about a movie
- a logged user can edit or delete its reviews
- a logged user can edit their personal information
- an admin user can delete reviews, movies and users
- the classifier predicts the spoiler attribute of new reviews

2.1.3 Non functional requirements

- The application must provide a great quality of service in terms of high availability and low latency.
- The user's usernames must be unique.
- The application shall be user-friendly, since a user shall interact with an intuitive graphical interface.
- The code shall be readable and easy to maintain.

2.2 Use case diagram

In Figure 1 it is shown the use case diagram of our application.

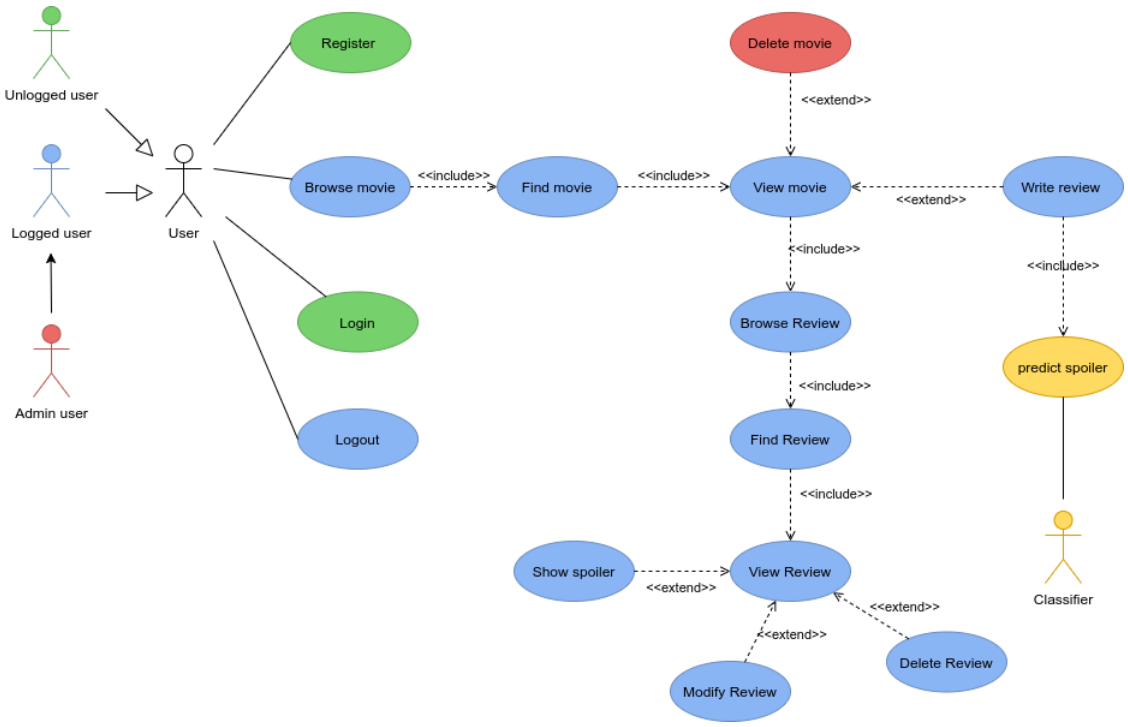


Figure 1: Use case diagram of *SpoilerDetector*

2.3 Class diagram

Figure 2 shows the UML class diagram of *SpoilerDetector*.

The generalization of *logged user* and *admin* into user is solved by adding an attribute *is_admin* into the entity *User*.

- each user can write 0 or more reviews
- each review is related to only one movie

2.4 Data Model

We decided to use a DocumentDB to store all the information about users, movies and reviews. Our database has two collections: *users* and *movies*.

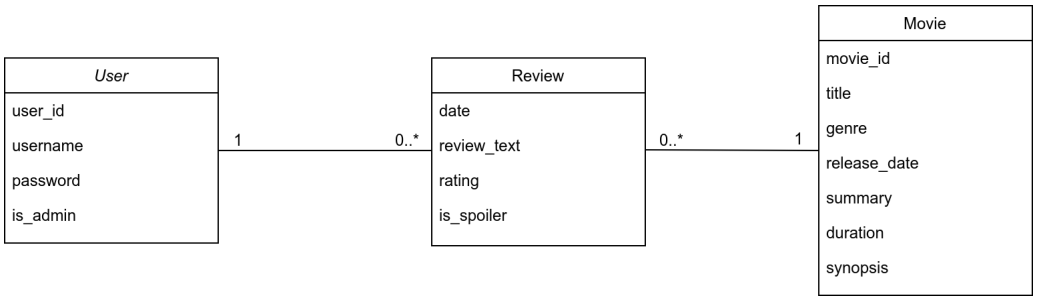


Figure 2: Class diagram of *SpoilerDetector*

2.4.1 Example of user's document

```
{
"_id": {"$oid": "62080e64a44c99e53aff21cb"},
 "username": "humorousPorpoise7",
 "password": "j0FH&aaP",
 "reviews": [
    {"_id": {"$oid":"62080e64a44c99e53aff1b68"},
     "title": "8 1/2",
     "review_date": "26 November 1998",
     "review_text": "8 1/2 has been known to bring a
        tear to my eye....",
     "rating": 10,
     "is_spoiler": false
    },
    {"title": "I, Tonya",
     "review_date": "17 February 2022",
     "review_text": "Very cool movie!",
     "rating": 7,
     "is_spoiler": false}],
  "is_admin": 0
}
```

2.4.2 Example of movie's document

```
{"_id": {"$oid":"62080e64a44c99e53aff1908"},
 "title": "Old School",
 "genre": ["Comedy"],
 "release_date": "2003-02-21",
 "summary": "Mitch, Frank and Beanie are ...",
 "synopsis": "Attorney Mitch Martin comes back ...",
```

```
 "duration": "1h 28min",
 "cover_url": "https://i.imgur.com/R7KOk45.png",
 "reviews":[
        {"_id": {"$oid":"62080e64a44c99e53aff1fa2"},
        "username": "sadPie6",
        "review_date": "22 February 2003",
        "review_text": "Very cool movie!",
        "rating": 8,
        "is_spoiler": false
        }]
}
```

2.5 System Architecture

In this section we want to describe the system architecture and which languages and tools we have used to realize it. The application has been realized in Java and it communicates with two different components:

- MongoDB database: A NoSQL database, document oriented, used to store all the informations about users, films and reviews.
- Flask web-application: A micro web framework written in Python. It has been used to manage RESTful request used to make machine learning model predictions on new reviews. The machine learning model contained in the Flusk web application has been trained, evaluated and deployed using Sklearn library which is a free software machine learning library for the Python programming language.

The system architecture can be seen in the figure 3.

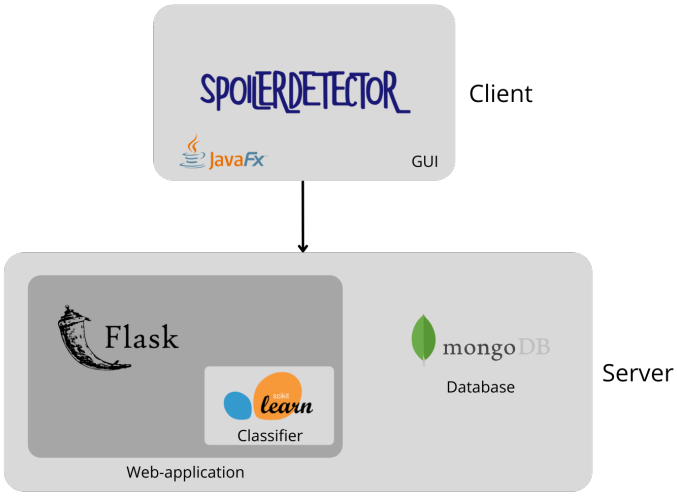


Figure 3: System Architecture

3 Machine Learning

This section describes the Machine Learning part of our application. The aim of the usage of a classifier in the application is to preserve users from reading spoilers by predicting for each new review if the review contains a spoiler or not, storing the information in the database and whenever the review is shown to the user, showing a popup to advise the user that the text may contain some spoilers and ask her/him if they want to read the full content or not.
The entire Machine Learning process has been developed in *Python* using the *Jupyter Notebook* tool.

3.1 Introduction

The machine learning part includes all the classical stages of KDD, which are:

- Data pre-processing: The goal is prepare the data for the mining operations. It is divided in: Data Cleaning, Data Integration, Data Reduction, Data Transformation and Discretization.
- Attribute selection: The goal is to decide which are the attributes that are relevant for the model.
- Model Learning and Validation: The goal is to train and evaluate the Machine Learning model.

The machine learning model used in our application is a binary classifier which takes as input the text of a review and produces as output a boolean value which describes if a review is spoiler (true) or not spoiler (false). Since the input is a "String", we had to perform some text mining techniques before applying the classifier.

3.2 Dataset

The dataset used for the training and evalution of the model has been taken from *Kaggle* and can be found with its name that is *IMDB Spoiler Dataset* or at the following link IMDB Spoiler Dataset. The dataset contains both movie details and their reviews but for the machine learning part we used only the reviews dataset.
Each review's record is described by seven attributes which are:

- ReviewDate: Timestamp of the review;
- MovieID: Unique ID of the movie;
- UserID: Unique ID of the user;
- Rating: Rate given to the film by the user;
- ReviewText: Text of the user's review;

- ReviewSummary: Text that summarizes the review text;
- isSpoiler: Ground truth of the classification problem.

The total number of reviews' in the dataset is 573913.

3.3 Data analysis

Before the preprocessing phase, we have performed some data analysis studies to better understand how the dataset was composed and how to mine information from it.

Class distribution First of all, we analyzed how the two classes (spoiler, not-spoiler) are distributed in the dataset. As we can notice from the histogram shown in the Figure 4, the dataset is a little imbalanced due to the difference of percentages in the class distribution.

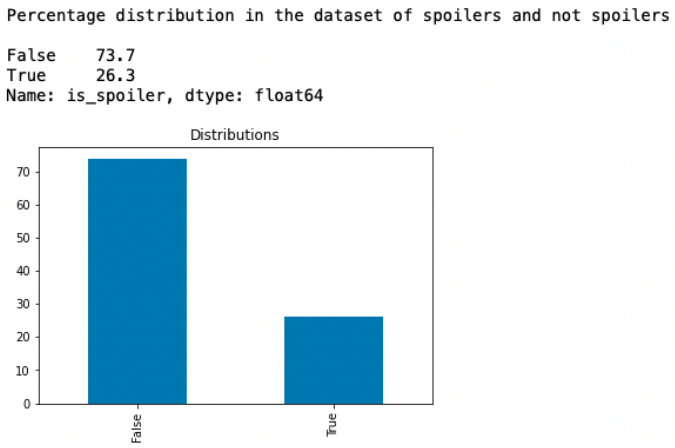
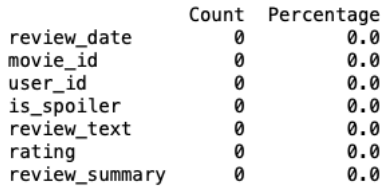


Figure 4: Dataset Distribution

Missing values We have analyzed if in the dataset were present some missing values to handle and, as we can see from Figure 5, we didn't found any.



	Count	Percentage
review_date	0	0.0
movie_id	0	0.0
user_id	0	0.0
is_spoiler	0	0.0
review_text	0	0.0
rating	0	0.0
review_summary	0	0.0

Figure 5: Number and Percentage of null values in the dataset

Spoiler distribution among different levels of rating Later, we have analyzed if the review's rating was an interesting factor in determining if a review is a spoiler or not. In order to study it, we computed the percentage of spoilers for each level of rating compared to the total number of reviews for the same level of rating. As shown in Figure 6, we found that with the increase of the rating value the percentage of spoilers decreases a little.

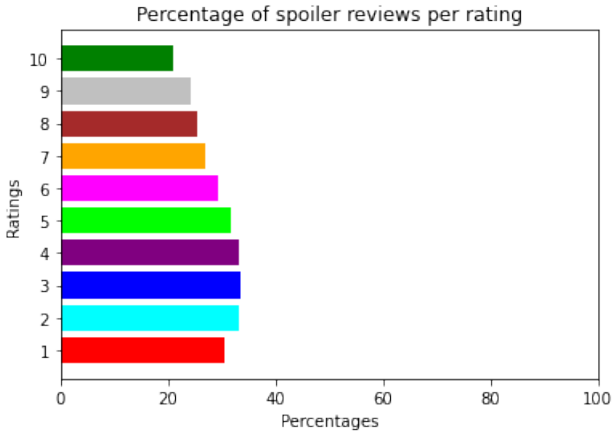


Figure 6: Percentage of spoilers by rating in the dataset

Spoiler distribution and selected words In this subsection, we checked if being a spoiler was related somehow to the presence of some classic spoiler words in the text of the review (we took as an example the words "spoiler", "die" and "win"). In particular we counted the number of spoiler (and not) reviews containing the selected word and we related it to the total number of spoiler reviews. The results that we obtained are shown in the pie charts in Figures 7, 8 and 9. On the left of each figure, it's represented the distribution of the classes in respect to the total number of reviews containing the word, while on the right it's represented the distribution of the reviews which contain the specified word in respect to the total number of spoiler reviews.

Examining reviews' length As last study, we checked if the length of the reviews is a discriminant for the spoiler label and, as shown in Figure 10, the average the number of words used in a spoiler review is higher than the one of a not spoiler review.

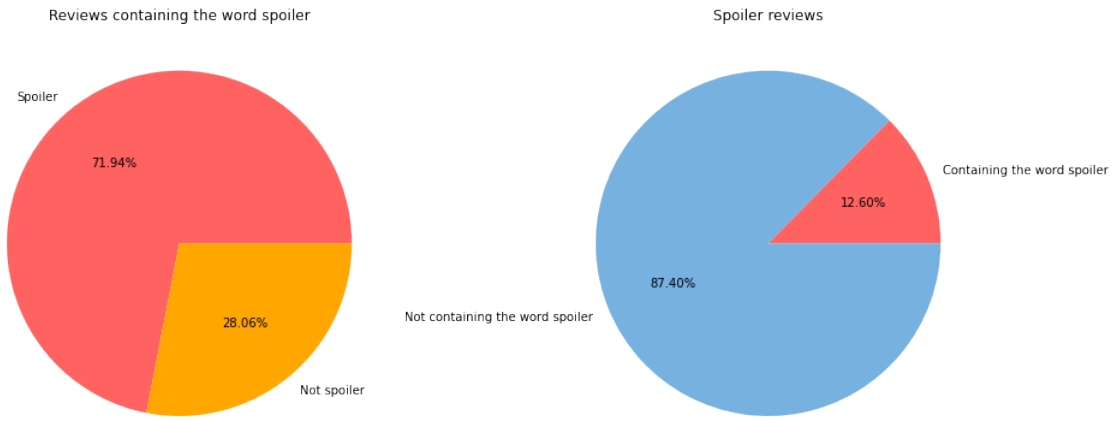


Figure 7: Spoiler distribution in relation to word "Spoiler"

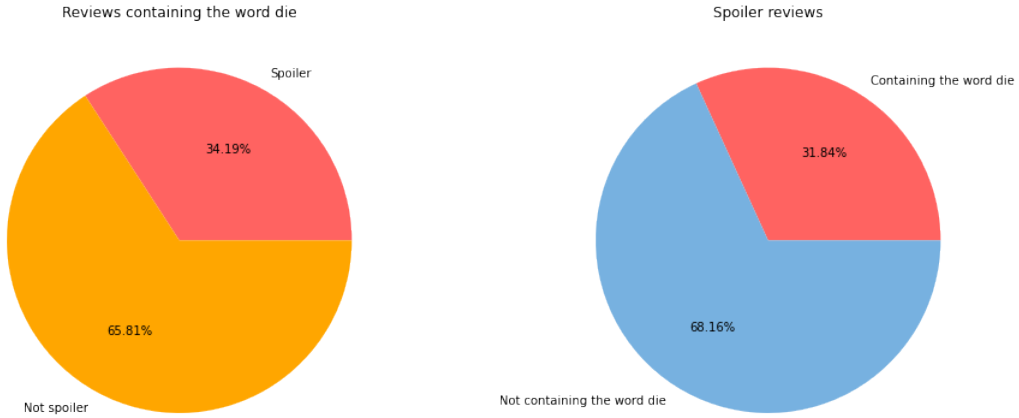


Figure 8: Spoiler distribution in relation to word "Die"

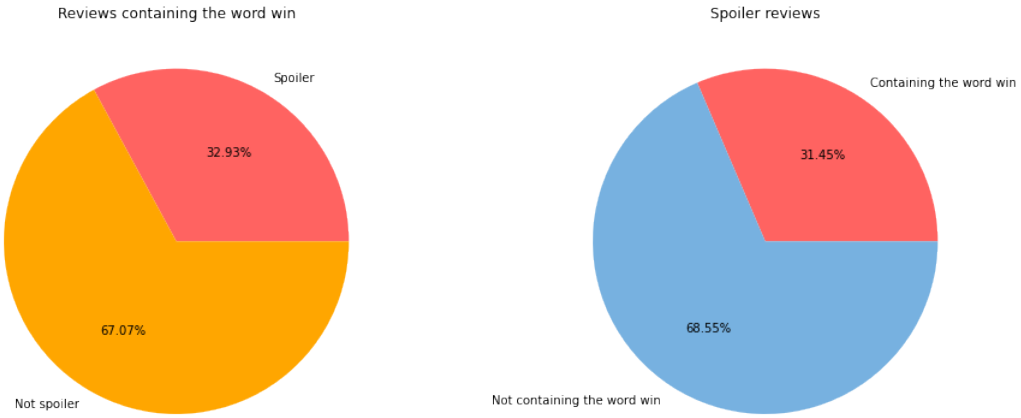


Figure 9: Spoiler distribution in relation to word "Win"

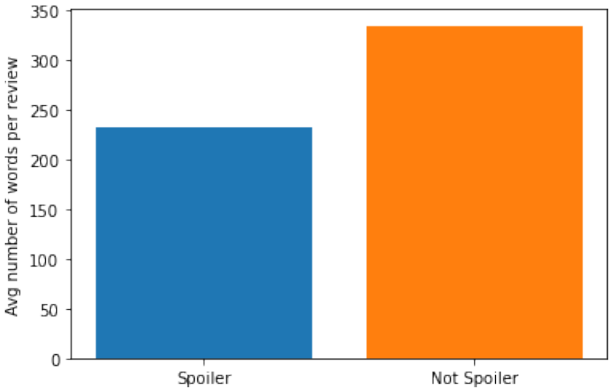


Figure 10: Average number of words per class

3.4 Preprocessing

The preprocessing stage has the aim to clean up the review text from words and noise.
The text cleaning functions applied to each review's text are:

- Remove links: this function removes links from the text.
- Remove accented characters: this function removes accented characters from the text.
- Reducing incorrect characters repetition: this function reduces the incorrect repetition to max then two repetitions.
- Expand contractions: this function expands the contractions of the words in the text.
- Remove stopwords: this function removes the stopwords from the text.
- Remove MeaningLess Words: this function removes the words with no meaning in order to reduce the noise present in the text.
- Stemming: this operation is used to stem some words in the review texts in order to obtain better performances of the classifier.

In order to test if the stemming (which has an high computational cost) really improves the model's performances (with english words, stemming may be not useful), we used two type of cleaned text reviews: one were was applied the stemming and one without the stemmed words and we compared the results obtained from both the kind of cleaned texts.

3.4.1 Outdated review, train set and test set

The datatime range of the reviews in the dataset was very large, in particular in the dataset are present reviews written between 1998 and 2018. Since the classifier goal is to detect the new spoiler reviews, we decided to consider reviews older than 2007 as outdated and with the same reasoning, we divided the last year reviews of the dataset (year 2017), dividing them into K (12) folds based on months, as the evaluation set for the final classification model, obtaining 45.631 reviews for the test set. The training set is composed with reviews between 2008 and 2016 included, obtaining a total of 288.351 instances.

3.5 Model Evaluation & Selection

In the model evaluation and selection phase, we performed all the operation needed to evaluate and to select the model which guarantees better performances. We used Sklearn and Imblearn libraries to do these operations and to handle the dataset distribution. Since our problem is the classification of a binary class, we analyzed the performances of three types of classification models:

```
--- Initial text example ---
http://google.com this IS treee okk test fòr text cleaning, isn't it  dsiadohaspidnpas anomalies ?

--- Lower function ---
http://google.com this is treee okk test fòr text cleaning, isn't it  dsiadohaspidnpas anomalies ?

--- Remove_Links function ---
 this is treee okk test fòr text cleaning, isn't it  dsiadohaspidnpas anomalies ?

--- Remove_Links function ---
this is treee okk test fòr text cleaning, is not it dsiadohaspidnpas anomalies ?

--- Remove_Accented_Characters function ---
this is treee okk test for text cleaning, is not it dsiadohaspidnpas anomalies ?

--- Remove_Special_Characters function ---
this is treee okk test for text cleaning  is not it dsiadohaspidnpas anomalies

--- Reduce_Incorrect_Character_Repeatation function ---
this is tree okk test for text cleaning is not it dsiadohaspidnpas anomalies

--- Remove stopwords_meaningLessWord and apply stemming function ---
tree test text clean anomali

--- Remove stopwords_meaningLessWord function -> Don't apply stemming ---
tree test text cleaning anomaly
```

Figure 11: Text preprocessing example

- Naive Bayes Multinomial
- Support Vector Machine
- Logistic Regression

To compare the three models we applied a K-fold cross validation.

3.5.1 Compare models

We compared the models in different ways to understand the best parameters, distribution and models to use.

Select best models using the dataset distribution First of all, we compared the models using the same distribution as the dataset one with the aim of doing a first comparison of the models and decide on which of them focus more to change the parameters and to perform an analysis of the distribution.

As we can see from the table 1 and from Figure 12, the multinomial model works really bad in terms of precision and recall in respect to the other two classifiers, so we decided to perform the analysis changing parameters, distribution of the classes and applying stemming only to the SVM and Logistic models, which appeared to have similar precisions and recalls and so were more interesting to investigate.

Compare SVM and Logistic on different parameters and class distributions We compared the SVM and Logistic models changing different combinations of the parameters of the classifiers:

Model	AvgPrecision	AvgRecall
Multinomial	0.62	0.13
SVM	0.56	0.36
Logistic	0.60	0.34

Table 1: Comparison of the models using the same distribution as the dataset

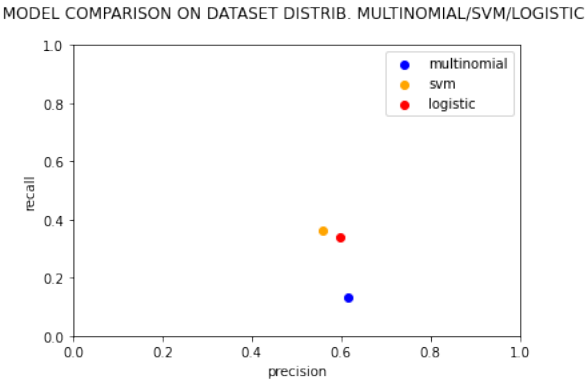


Figure 12: Comparison of the models using the same distribution as the dataset

- Ngram-Range: we used different ngram-ranges of the count vectorizer. The ngram-range specifies the lower and upper boundary of the range of n-values for different word n-grams or char n-grams to be extracted. In particular, we used ngram-range equal to (1,2) and (2,2). For example an ngram-range of (1, 1) means only unigrams, (1, 2) means unigrams and bigrams, and (2, 2) means only bigrams.

- Class Distributions: we used different class distributions for the training dataset by applying the *RandomUnderSampler* to undersample the major class. The RandomUnderSample changes only the major class (if not specified as parameter to change the minor class). We have used distributions equal to 50/50, 60/40 and the same distribution of the dataset.

- Stemming: For each ngram-range and distribution case, we performed experiments with both stemmed text and not-stemmed text in order to check if the stemming really improves the classifier performances.

The complete results that we obtained are reported in the table 2. Then we filtered the results keeping only the most relevant in terms of precision and recall (we kept those having an average precision and average recall higher than 0.45) and we reported them in the tables 3 and 4 and in the plot in the figure 13.

Since in our application the recall measure is more relevant in respect of the precision since it is more important to classify a true spoiler review as spoiler than classifying well spoiler reviews, we decided to get more in details with the Logistic model since it gives the best results in terms of recall and gives acceptable results in terms of precision instead of study more the SVM which gives a result with a lower recall.

	SVM		Logistic	
Parameters Combination	AvgPrecision	AvgRecall	AvgPrecision	AvgRecall
ngram1-1-sample5050	0.47	0.60	0.49	0.61
ngram1-1-sample5050-stem	0.49	0.61	0.49	0.61
ngram1-1-sample6040	0.52	0.45	0.55	0.44
ngram1-1-sample6040-stem	0.53	0.45	0.55	0.44
ngram1-1-sampledataset	0.56	0.36	0.60	0.34
ngram1-1-sampledataset-stem	0.57	0.35	0.60	0.34
ngram1-2-sample5050	0.47	0.63	0.50	0.65
ngram1-2-sample5050-stem	0.47	0.63	0.50	0.64
ngram1-2-sample6040	0.52	0.50	0.56	0.48
ngram1-2-sample6040-stem	0.52	0.50	0.56	0.47
ngram1-2-sampledataset	0.56	0.42	0.61	0.37
ngram1-2-sampledataset-stem	0.56	0.42	0.61	0.37

Table 2: SVM and Logistic Regression scores on different parameters combination

Parameters	AvgPrecision	AvgRecall
ngram1-1-sample5050	0.47	0.60
ngram1-1-sample6040	0.52	0.45
ngram1-2-sample5050	0.47	0.63
ngram1-2-sample6040	0.52	0.50

Table 3: Relevant results of SVM model

Parameters	AvgPrecision	AvgRecall
ngram1-1-sample5050	0.49	0.61
ngram1-2-sample5050	0.49	0.65
ngram1-2-sample6040	0.56	0.47

Table 4: Relevant results of Logistic Regression model

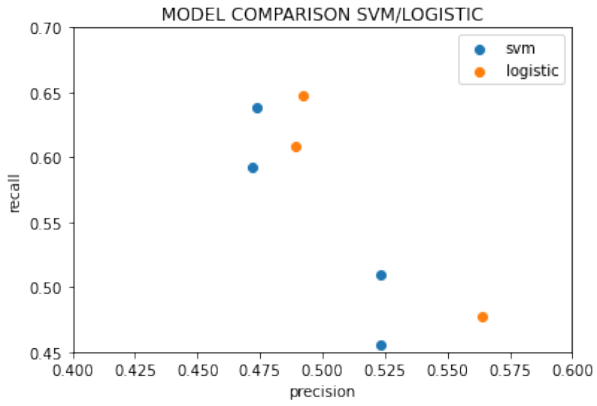


Figure 13: Comparison between SVM and Logistic

Logistic Regression model selection As explained previously, we analyzed the best results only for logistic model and decided that the best set of parameters for our model is the one with parameters *ngram=(1,2), sample=(50/50)* since, as shown in Figure 14, it gives an high recall and an acceptable precision.

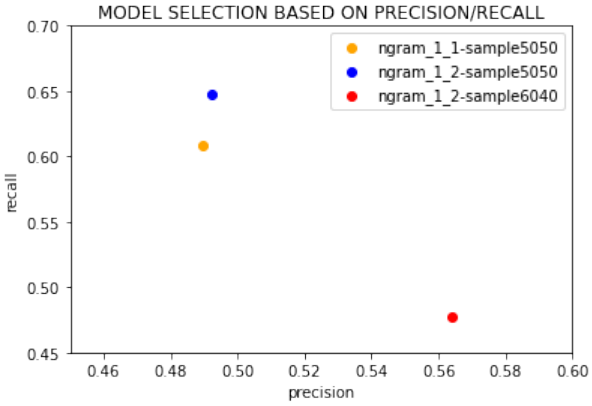


Figure 14: Model Selection Logistic Regression

4 Model Deploy

In this section we will analyze how we have trained, evaluated and deployed the final model (the one used by the Java application).

4.1 Model training

As we have seen in the section 3.5.1, we have selected as final model the *Logistic Regression* with the following parameters:

- Ngram-range: (1,2)
- Distribution: 50/50
- Stemming: False

The fit was performed on all the training dataset.

4.2 Model evalutation

The test of the classifier was performed using the test set extracted during the Data Preprocessing phase. As explained in section 3.4.1, the test set is composed by all the latest reviews of the original dataset (January 2017, December 2017) and we divided them into K-fold based on the month and we used the K-fold cross-validation to evaluate our final model.
The class distribution and the number of instances for each K-fold is show in the figure 15.

We have computed the precision and the recall obtained for each fold, and we reported the results in the table 5. As we can see, we have obtained acceptable results in terms of precision and recall.

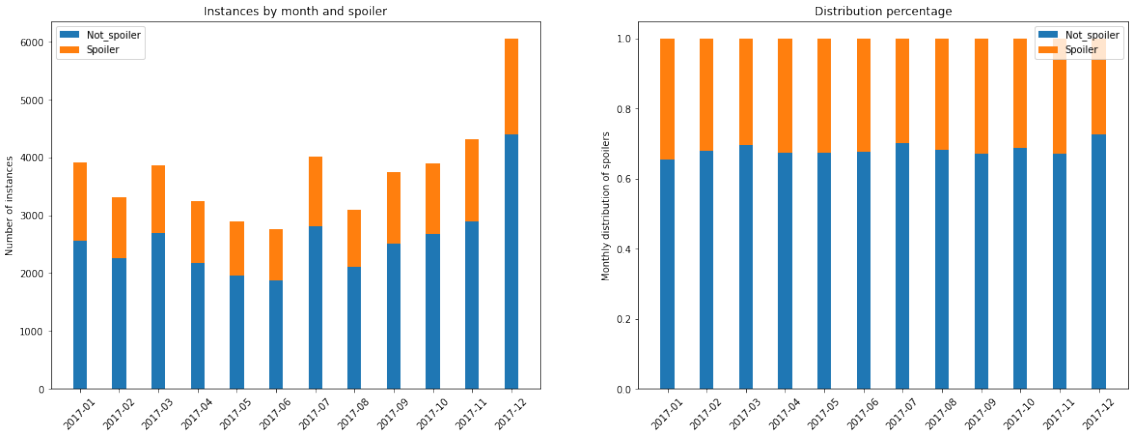


Figure 15: Test set number of instances and class distribution

Timestamp	Precision	Recall
2017-01	0.53	0.63
2017-02	0.52	0.65
2017-03	0.50	0.62
2017-04	0.52	0.64
2017-05	0.52	0.64
2017-06	0.50	0.59
2017-07	0.49	0.62
2017-08	0.50	0.62
2017-09	0.53	0.60
2017-10	0.52	0.60
2017-11	0.53	0.61
2017-12	0.50	0.62

Table 5: Precision and Recall measured on test set

4.2.1 Model Deploy

To deploy the model in order to use it in our Java application, we used the *joblib* library that is a set of tools which provides lightweight pipelining in Python and allows model saving in a way that it can operate on objects with large NumPy arrays/data as a backend with many parameters.

The model has been saved in the pickle format (.pkl). A PKL file is pickled to save space when being stored or transferred over a network then is unpickled and loaded back into program memory during runtime.

4.3 Conclusions

In conclusion, the results that we obtained are acceptable but not optimal so we want to propose some possible ways of proceeding in order to improve the accuracy of the model.

One possibility that may improve the performances of the classifier can be to take in consideration the synopsis of the movie to which the review is referred; this possibility could improve the model metrics because some words can be spoiler for some films but may be not spoiler for other movies.

Another possibility is to consider the rating of the review since, as we have seen in Figure 6, there may be a little correlation between the increase of rating and the decrease of spoiler reviews.

The last possibility that we want to propose is to try some other combination of the parameters of the Count Vectorizer, TF-IDF and models. We have only tried some combinations and some ranges of values for the parameters but there a lot of other possibilities. For what concerns this solution, an important consideration is that more parameters the training has to test, more time it will require in order to complete the training. One possible available tool that can be used to perform a K-fold cross validation test using a huge number of parameters is the *Grid Search*.

All we have said is to emphasize that a model can be improved by exploring different possibilities and analyzing the latter. The results we have obtained are only a small part of them.