uc3m | Universidad Carlos III de Madrid

Degree in Data Science and Engineering
2022-2023

Machine Learning Applications

Final project

Jorge Parreño - 100429982
Sven Bulach - 100502905
Pierre Gourdou - 100502587
Madrid, May 7^{th} 2023

CONTENTS

1. DATASET

The IMDb reviews dataset is a large collection of movie reviews that has been used extensively in natural language processing (NLP) research and machine learning applications. The dataset includes a total of 50,000 reviews, with an equal number of positive and negative reviews. The reviews were sourced from the popular movie database website, IMDb.

Each review is stored as a separate text file and covers a wide range of movie genres, including drama, comedy, action, and horror, among others. The reviews were written by a diverse group of users, including professional critics and casual moviegoers.

The dataset has already been preprocessed and tokenized, with each word represented by a unique integer identifier. This preprocessing step helps to standardize the data and improve model performance. However, it is also possible to obtain the unprocessed or untokenized text for each review, depending on the version of the dataset.

The dataset has been widely used for sentiment analysis and other NLP tasks, including text classification and language modeling. It is also frequently used as a benchmark for evaluating new models and algorithms in the field.

In addition to its use in machine learning and NLP research, the dataset has also been analyzed to gain insights into movie preferences and trends among viewers. For example, researchers have used the dataset to analyze the language and sentiment used in reviews of popular movies.

Taking this dataset as inspiration we chose to the create a web scraping script to build our own version of it, containing reviews and ratings, the same as the one that can be downloaded but composed of instances collected through scraping. The ratings distribution of the scraped dataset is the following one,

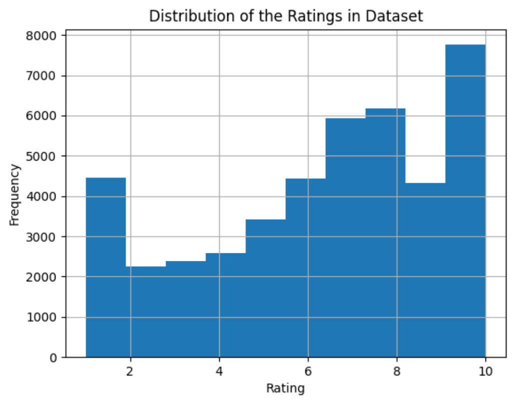

Fig. 1.1. Rating distribution

2. TEXT PREPROCESSING AND VECTORIZATION

Preprocessing

As we built a dataset through web scraping, this involved the following preprocessing steps, removal of alphanumeric characters, extra spaces, HTML tags, urls, tokenitzation and removal of stop words and lemmatization.

Vectorizations

We considered a wide array of different vectorization techniques of the ones covered in the course, in particular bag of words, term frequency-inverse document frequency and variations of word2vec.

BoW and TFIDF

Bag-of-Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF) are commonly used methods for representing text data in machine learning. In BoW, a document is represented as a vector of word counts, where each dimension corresponds to a unique word in the vocabulary. For example, if the vocabulary contains three words "dog", "cat", and "bird", and the document contains the text "dog cat cat bird", then the BoW vector would be [1, 2, 1].
However, BoW has a few limitations. It does not take into account the relative importance of different words, and it treats each word as equally important. Additionally, it does not handle out-of-vocabulary words, or words that were not seen during training.
To address these issues, TF-IDF can be used. In TF-IDF, the importance of each word is weighted by its frequency in the document and its inverse frequency in the corpus. The TF-IDF weight of a word i in a document j is given by:

$$\text{tf-idf}(i, j) = \text{tf}(i, j) \times \text{idf}(i) \tag{2.1}$$

where tf(i, j) is the frequency of word i in document j, and idf(i) is the inverse document frequency of word i, given by:

$$\text{idf}(i) = \log \frac{N}{df(i)} \tag{2.2}$$

where N is the total number of documents in the corpus, and $df(i)$ is the number of documents that contain word i. The logarithm is used to dampen the effect of very rare words that may have high IDF values.

Word2Vec

Word2Vec is a popular algorithm for learning dense vector representations, also known as embeddings, of words from large text corpora. These embeddings capture semantic and syntactic relationships between words, and can be used as input to machine learning algorithms for various natural language processing tasks, such as language modeling, sentiment analysis, and machine translation.
Word2Vec can be trained using either the Continuous Bag-of-Words (CBOW) or Skip-gram model. In CBOW, the task is to predict the target word given its surrounding context words, while in Skip-gram, the task is to predict the context words given the target word. Both models use a neural network with a single hidden layer, where the input and output layers represent the words as one-hot vectors, and the hidden layer represents the dense vector representation, or embedding, of the words.
Word2Vec can be trained on n-gram models as well. In n-grams, the input is divided into n consecutive words. 1-gram models consider one word at a time, 2-gram models consider two consecutive words, and so on. A window of size w can be used to limit the number of words that are considered for the context in the n-gram model. For example, a 1-gram window of size 8 means that only the 8 words surrounding the target word will be considered for training the Word2Vec model.

Topic Modeling

In order to do topic modeling we used latent dirichlet allocation. LDA is a generative probabilistic model that is widely used for topic modeling in natural language processing and information retrieval. It is based on the assumption that a document can be represented as a mixture of topics, and each topic is represented as a probability distribution over words in the vocabulary.
The goal of LDA is to discover the underlying topics in a large corpus of documents and to estimate the probability distribution of each topic over the words in the vocabulary. To achieve this, LDA assumes that each document is generated by a mixture of topics, where each topic is a probability distribution over the words in the vocabulary. The topic proportions in each document are drawn from a Dirichlet distribution, which is a distribution over probability distributions. The word distribution of each topic is also drawn from a Dirichlet distribution.
The LDA algorithm works by inferring the posterior distribution of the latent variables given the observed data. This is done using variational inference or Gibbs sampling. The algorithm iteratively updates the estimates of the topic proportions and the word distributions of each topic until convergence. We used the gensim library to implement LDA and extract topics from our corpus of reviews. By choosing to extract 10 topics we can access to the distribution of those ten topics (i.e. the words contained in this topic as well as their weight in the distribution).

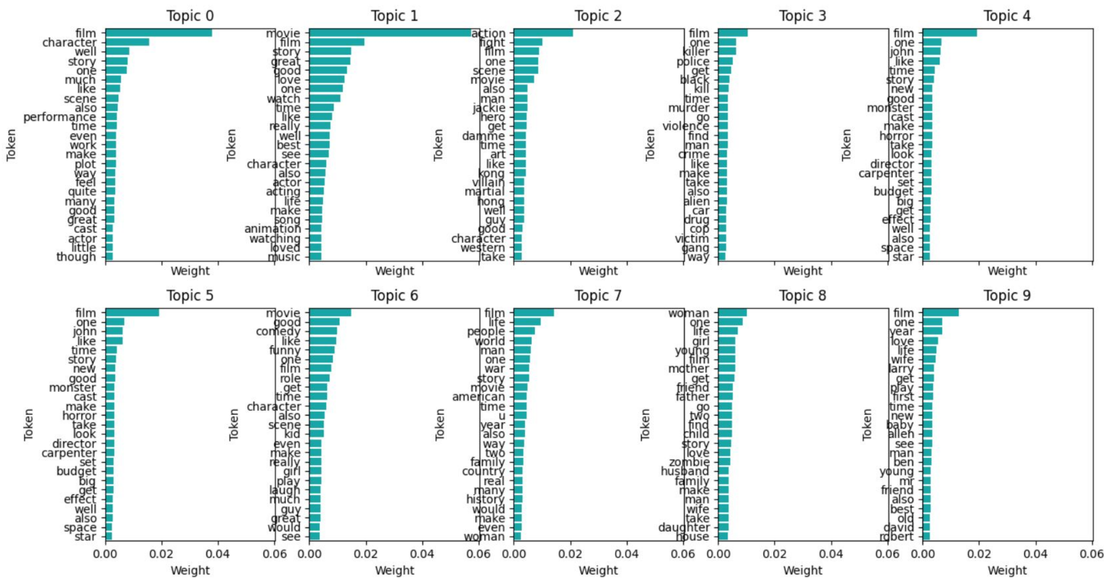

Fig. 2.1. Topic distribution

We can see by looking at the different topics we can see that some topics have very similar word distributions (for example topic 4 and 5). We could therefore conclude that it is unnecessary to keep those and only choose one instead. The goal is indeed to have the least amount of topics to describe the corpus so the hard task is to pick the most relevant ones. This is an hard task because the LDA algorithm will generate different topics given the different number of topics we set as a parameter (This can be seen on the dashboard). To have an idea of how many topics are needed to describe the corpus, we can plot the coherence of the LDA model given the number of chosen topics:

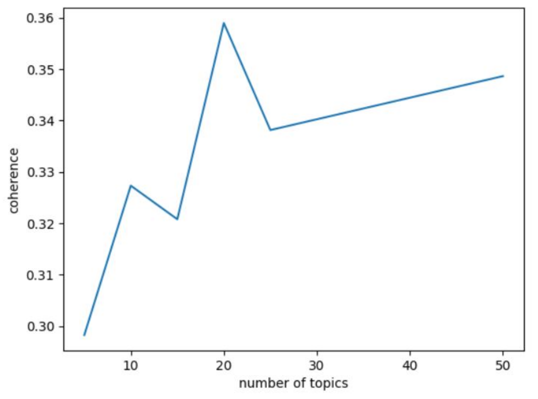

Fig. 2.2. Topic distribution

This graph shows that it is necessary to reach around 20 topics to have the best coherence but the coherence stays pretty low (around 0.36). To try to identify the 20 most relevant topics, it would have been interesting to compare the topics given by the classic gensim LDA model with one trained with Mallet, an extension of gensim. Unfortunately the Mallet package wasn't working at all.

3. MACHINE LEARNING MODEL

According to the requirements we chose to fulfill a classification task, in this section we tried combinations of different machine learning algorithms with the vectorizations of the texts performed in section I in order to perform sentiment analysis.

Machine learning algorithms

K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a non-parametric classification algorithm that is used for both regression and classification tasks. The algorithm is based on the idea that similar instances are close to each other in the feature space.
Given a new instance, KNN looks for the K nearest neighbors in the training set based on a distance metric (e.g., Euclidean distance). The algorithm then assigns the new instance to the class that is most common among its K nearest neighbors. The value of K is a hyperparameter that needs to be tuned for each dataset.

Support Vector Machines (SVM)

Support Vector Machines (SVMs) are a powerful class of supervised learning algorithms used for classification and regression tasks. In the case of classification, SVMs aim to find a hyperplane that separates the positive and negative examples with the largest margin. The examples that lie closest to the hyperplane are called support vectors, and are used to define the hyperplane.
One of the advantages of SVMs is that they can handle high-dimensional data and can learn complex decision boundaries.

Gradient Boosting

Gradient Boosting is an ensemble learning technique that combines multiple weak models, typically decision trees, into a strong model. The basic idea behind Gradient Boosting is to iteratively add models to the ensemble, with each model trying to improve on the mistakes of the previous models.
At each iteration, the Gradient Boosting algorithm fits a new model to the residuals of the previous models. The residuals are the differences between the true labels and the predicted labels of the previous models. The new model is then added to the ensemble with a weight that is determined by a learning rate parameter, which controls the contribution of each model to the final prediction.

Random Forest

Random Forest is an ensemble learning technique that combines multiple decision trees into a single model. The basic idea behind Random Forest is to train a set of decision trees on different subsamples of the training data, and then aggregate the predictions of the individual trees to make a final prediction.
Random Forest is typically used for classification and regression tasks. In the case of classification, the final prediction is made by taking the majority vote among the individual trees. In the case of regression, the final prediction is made by taking the average of the individual tree predictions.

Logistic Regression

Logistic Regression is a statistical learning technique used for classification tasks, where the goal is to predict a binary outcome based on a set of input features. The basic idea behind Logistic Regression is to model the probability of the positive class as a function of the input features.
In Logistic Regression, the predicted probability of the positive class is transformed using a logistic function, which maps the predicted probabilities to the range [0, 1].
Once the parameters have been estimated, the logistic regression model can be used to make predictions by computing the predicted probability of the positive class for each input example, and then applying a threshold to convert the probabilities to binary class labels.

Experiments

In the experiments we included different variations of the vectorizers explained in the first section. We employed *BoW*, combined with *TF-IDF* and variations of *Word2Vec*, modifying the n-gram size and the window size, in total 5 vectorizers with 8 machine learning algorithms which resulted in 40 combinations. We optimized every classifier on a with a randomized search and a given search space. The performance evaluated on all the 10 classes, was very bad so we concluded, that this task is too hard. This makes sense, since users have a very subjective perception and give different ratings for similar positive or negative written content. Therefore as a first experiment, we trained the classifiers on all the 10 different classes and evaluated them on only four and two classes respectively. Table 3.1 shows the results for the evaluation on four classes. Table 3.2 shows the results for the evaluation on two classes. As a next step, we took the 3 best performing classifier-vectorizer combinations and optimized them on a bigger search space. This resulted only in a very slightly increase of performance which can be seen in table 3.3.
Then we mapped our dataset with 10 classes to only four and two classes respectively and trained the three previously best performing classifiers on these new datasets again. Table

3.6 shows the results for the training on four classes and evaluation on four classes. Table 3.5 shows the results for the training on four classes and evaluation on two classes and table 5 shows the results for training on two classes and evaluation on two classes.

Vectorizer	ML Algorithm	Accuracy	Precision	Recall	F1-Score
BoW	logistic regression	0.570	0.571	0.570	0.555
BoW	gradient boosting	0.527	0.532	0.527	0.510
BoW	random forest	0.508	0.530	0.508	0.456
BoW	svm	0.563	0.569	0.563	0.541
BoW	logistic regression balanced	0.571	0.577	0.571	0.568
BoW	random forest balanced	0.535	0.536	0.535	0.519
BoW	svm balanced	0.579	0.574	0.579	0.574
BoW	knn	0.342	0.389	0.342	0.280
BoW and tfidf	logistic regression	0.568	0.563	0.568	0.562
BoW and tfidf	gradient boosting	0.532	0.530	0.532	0.517
BoW and tfidf	random forest	0.514	0.532	0.514	0.464
BoW and tfidf	svm	0.599	0.595	0.599	0.589
BoW and tfidf	logistic regression balanced	0.579	0.580	0.579	0.572
BoW and tfidf	random forest balanced	0.542	0.539	0.542	0.526
BoW and tfidf	svm balanced	0.599	0.595	0.599	0.589
BoW and tfidf	knn	0.412	0.437	0.412	0.386
Word2Vec1gram	logistic regression	0.585	0.579	0.585	0.576
Word2Vec1gram	gradient boosting	0.569	0.563	0.569	0.561
Word2Vec1gram	random forest	0.533	0.531	0.533	0.496
Word2Vec1gram	svm	0.589	0.584	0.589	0.582
Word2Vec1gram	logistic regression balanced	0.567	0.569	0.567	0.564
Word2Vec1gram	random forest balanced	0.535	0.528	0.535	0.507
Word2Vec1gram	svm balanced	0.579	0.574	0.574	0.579
Word2Vec1gram	knn	0.342	0.280	0.389	0.342
Word2Vec3gram	logistic regression	0.589	0.578	0.584	0.589
Word2Vec3gram	gradient boosting	0.567	0.559	0.561	0.567
Word2Vec3gram	random forest	0.549	0.522	0.544	0.549
Word2Vec3gram	svm	0.594	0.581	0.591	0.594
Word2Vec3gram	logistic regression balanced	0.573	0.572	0.577	0.573
Word2Vec3gram	random forest balanced	0.549	0.532	0.543	0.549
Word2Vec3gram	svm balanced	0.586	0.588	0.593	0.586
Word2Vec3gram	knn	0.513	0.506	0.507	0.513
Word2Vec1gram_w8	logistic regression	0.591	0.583	0.585	0.591
Word2Vec1gram_w8	gradient boosting	0.571	0.563	0.565	0.571
Word2Vec1gram_w8	random forest	0.542	0.508	0.544	0.542
Word2Vec1gram_w8	svm	0.594	0.578	0.591	0.594
Word2Vec1gram_w8	random forest balanced	0.544	0.522	0.539	0.544
Word2Vec1gram_w8	svm balanced	0.582	0.584	0.590	0.582
Word2Vec1gram_w8	knn	0.506	0.500	0.502	0.506

TABLE 3.1. RESULTS OF DIFFERENT VECTORIZERS AND ML

Vectorizer	ML Algorithm	Accuracy	Precision	Recall	F1-Score
BoW	logistic regression	0.815	0.807	0.815	0.815
BoW	gradient boosting	0.775	0.760	0.775	0.775
BoW	random forest	0.771	0.744	0.794	0.771
BoW	svm	0.800	0.786	0.808	0.800
BoW	logistic regression balanced	0.807	0.809	0.814	0.807
BoW	random forest balanced	0.798	0.795	0.794	0.798
BoW	svm balanced	0.815	0.812	0.813	0.815
BoW	knn	0.672	0.632	0.648	0.672
BoW and tfidf	logistic regression	0.813	0.809	0.811	0.813
BoW and tfidf	gradient boosting	0.777	0.763	0.776	0.777
BoW and tfidf	random forest	0.768	0.740	0.790	0.768
BoW and tfidf	svm	0.829	0.822	0.829	0.829
BoW and tfidf	logistic regression balanced	0.825	0.824	0.824	0.825
BoW and tfidf	random forest balanced	0.799	0.795	0.796	0.799
BoW and tfidf	svm balanced	0.829	0.826	0.829	0.829
BoW and tfidf	knn	0.707	0.680	0.695	0.707
Word2Vec1gram	logistic regression	0.832	0.829	0.830	0.832
Word2Vec1gram	gradient boosting	0.815	0.810	0.812	0.815
Word2Vec1gram	random forest	0.791	0.776	0.796	0.791
Word2Vec1gram	svm	0.829	0.826	0.827	0.829
Word2Vec1gram	logistic regression balanced	0.819	0.820	0.822	0.819
Word2Vec1gram	random forest balanced	0.796	0.786	0.796	0.796
Word2Vec1gram	svm balanced	0.821	0.824	0.831	0.821
Word2Vec1gram	KNN	0.771	0.767	0.766	0.771
Word2Vec3gram	Logistic Regression	0.834	0.830	0.833	0.834
Word2Vec3gram	Gradient Boosting	0.815	0.810	0.812	0.815
Word2Vec3gram	Random Forest	0.796	0.784	0.797	0.796
Word2Vec3gram	SVM	0.830	0.824	0.829	0.830
Word2Vec3gram	Logistic Regression balanced	0.817	0.819	0.824	0.817
Word2Vec3gram	Random Forest balanced	0.804	0.797	0.801	0.804
Word2Vec3gram	SVM balanced	0.824	0.827	0.833	0.824
Word2Vec3gram	KNN	0.774	0.769	0.769	0.774
Word2Vec1gram_w8	Logistic Regression	0.835	0.832	0.833	0.835
Word2Vec1gram_w8	Gradient Boosting	0.817	0.812	0.815	0.817
Word2Vec1gram_w8	Random Forest	0.798	0.785	0.801	0.798
Word2Vec1gram_w8	SVM	0.833	0.827	0.833	0.833
Word2Vec1gram_w8	Random Forest balanced	0.802	0.795	0.800	0.802
Word2Vec1gram_w8	SVM balanced	0.823	0.825	0.833	0.823
Word2Vec1gram_w8	KNN	0.774	0.770	0.770	0.774

TABLE 3.2. RESULTS OF DIFFERENT VECTORIZERS AND ML

Vectorizer	ML Algorithm	Accuracy	Precision	Recall	F1-Score
Word2Vec1gram	logistic regression	0.831	0.830	0.831	0.827
Word2Vec3gram	logistic regression	0.836	0.835	0.836	0.832
Word2Vec1gram_w8	logistic regression	0.834	0.832	0.834	0.830

TABLE 3.3. RESULTS OF LOGISTIC REGRESSION ALGORITHM WITH DIFFERENT VECTORIZERS TRAINED ON ALL CLASSES AND EVALUATED ON FOUR CLASSES.

Vectorizer	ML Algorithm	Accuracy	Precision	Recall	F1-Score
word2vec_1gram	Logistic Regression	0.604	0.605	0.603	0.599
word2vec_3gram	Logistic Regression	0.612	0.615	0.612	0.608
word2vec_1gram_w8	Logistic Regression	0.609	0.611	0.609	0.605

TABLE 3.4. RESULTS OF DIFFERENT VECTORIZERS AND ML ALGORITHMS TRAINED AND EVALUATED ON FOUR CLASSES.

Vectorizer	ML Algorithm	Accuracy	Precision	Recall	F1-Score
word2vec_1gram	logistic regression	0.8406	0.8389	0.8399	0.8372
word2vec_3gram	logistic regression	0.8414	0.8400	0.8400	0.8377
word2vec_1gram_w8	logistic regression	0.8432	0.8417	0.8417	0.8399

TABLE 3.5. RESULTS OF LOGISTIC REGRESSION TRAINED ON FOUR CLASSES AND EVALUATED ON TWO CLASSES.

Vectorizer	ML Algorithm	Accuracy	Precision	Recall	F1-Score
Logistic Regression	word2vec_1gram	0.8425	0.84	0.8406	0.8425
Logistic Regression	word2vec_3gram	0.8471	0.8446	0.8454	0.8471
Logistic Regression	word2vec_1gram_w8	0.8457	0.8434	0.8439	0.8457

TABLE 3.6. RESULTS OF LOGISTIC REGRESSION WITH DIFFERENT VECTORIZERS TRAINED AND EVALUATED ON TWO CLASSES.

4. IMPLEMENTATION OF A DASHBOARD

Dash is a Python framework for building web applications. In our project we used it to display the influence of the number of topics chosen in the LDA algorithm. The Dashboard indeed include an interactive button where the user can choose between 5,10 or 20 number of topics chosen and the 5 most important topics in each case will be displayed with their word distribution (Token and Weight).