Amazon Kindle Store Reviews

Federico Moiraghi [799735] - Pranav Kasela [846965] - Roberto Berlucchi [847939]

Abstract
The present Project is composed by two parts: in the first part it is provided a model to predict if the buyer of a e-book on Amazon's Kindle Store liked it or not according to the text of the comment; in the second part...

Contents

Introduction

The present Project is divided into several parts, starting from the same corpus (Amazon's Kindle Store reviews[1]), with different purposes. In the first part, the aim is to provide a good supervised model to predict if a user liked the book or not according to his or her review.

The entire Project[2] is made using Apache Hadoop[3] (for data pre-processing) and Apache Spark[4] (for the analysis) and is distributed as Dock image through DockHub[5]. Using a Big Data framework is required by the huge quantity of texts, hardly analyzable by a single machine without taking some precautions: this Project aims not only to provide a valid approach to all tasks but also to propose a scalable and efficient way to solve them. Distribution through Docker is considered necessary to simplify installation of dependencies: providing a virtual Operating System (in the specific case, Linux Ubuntu 16.04) with all programs already installed and configured is useful to focus the attention on the Natural Language Processing (NLP) pipeline instead of building the environment.

It is chosen Kindle Store corpus to avoid problems that physical products reviews have in common: a lot of reviews do not focus on the product itself but stress how it was delivered. Another reason to limit the analysis to the Kindle Store corpus is the fact that its size on disk does not require a cluster (the entire Amazon corpus weights about 700Gb) but can be processed by a single machine in a reasonable time.

[1] https://nijianmo.github.io/amazon/index.html
[2] Original code is available on GitHub at the link https://github.com/moiraghif/Amazon-Rating-Prediction
[3] https://hadoop.apache.org/
[4] https://spark.apache.org/
[5] pkasela/amazon_review_hadoop_spark

1. Preprocessing and Data Cleaning

As the dataset is composed by real reviews from Amazon's Kindle Store, the cleaning part is fundamental to reach good performance on all tasks. The file is in .json format and contains approximately $5 \cdot 10^6$ observations (5,722,988 reviews of 493,859 products) coming from Amazon U.S.A. Kindle Store in a period starting from May 1996 to October 2018. For each observation some recordings are reported, such as ID of the product, of the reviewer, timestamp (in seconds from epoch), text, rating (number of stars from 1 up to 5) and votes of the review (given by other users). Fields of interest are selected using a series of regular expression (regex) to parse the .json string, instead of the official Python's json library, obtaining a 20% gain in speed.

Reviews are written for the most in English, but a small fraction (approximately 5%) is composed by texts written in other languages (for the most Spanish or French) or contains just emoticons. For the purpose of this Project, those reviews are just ignored to avoid the additional complexity caused by the multilingual case. CLD-2[1] library is used to filter reviews by language, removing those that are not in English or do not contain text: it uses a Recurrent Neural Network (RNN) to implement a character-level analysis of the text and predict its language. This library is selected among others due to its high speed in managing the quantity of data used in the Project: a comparison made by authors shows that it is more than 200 times faster than Google's langdetect and achieves comparable performances[1, p. 6]:

Name	TextCat	CLD-2	lang-detect	langid
Languages in WiLI-2018	**140**	90	55	95
Languages not in WiLI-2018	**115**	10	0	2
Speed for 10^6 elements	$1.5 \cdot 10^6$ s	**38 s**	8025 s	1858 s
†Mean Precision	48 %	96 %	**97 %**	93 %
†Mean Recall	42 %	95 %	**96 %**	90 %
†Mean F1	38 %	95 %	**96 %**	90 %
WiLI-2018 accuracy	35 %	**36 %**	22 %	**36 %**
Unknown prediction accuracy	—	**37 %**	25 %	**37 %**

After this first phase of cleaning, data is stored in a structured form on the Hadoop file-system (HDFS) and is parsed by a series of regular expression to make it more clean:

- HTML tags and URLs are removed;
- punctuation is normalized (e.g. four or more consecutive points are reduced to three);
- contractions are re-written in extended form;
- particular constructions are re-written in simplified form (e.g. "have to" → "must", "is going to" → "will");
- domain-specific terms are simplified (e.g. "ebook" → "book", all Amazon's offers becomes "Amazon", all Kindle e-book readers becomes "kindle");
- numbers are re-written according to their semantics (when easily understandable by the context) or removed (e.g. all prices becomes "money");
- modal verbs are lemmatized;
- words with emphatic repetition of a letter are normalized;
- remaining contractions or words composed by a single letter are removed;
- punctuation is removed.

After this pipeline, the text is processed by a SpaCy[6] model that tokenize sentences and words within each sentence (using a Deep Learning approach); each word is then either stemmatized by Porter's Stemmer (second version) offered by NLTK[7] or removed if it is a stopwords. Stopwords are taken from NLTK's English list, from which negations and some prepositions are removed, and with the addition of some domain-specific words (such as "book" or "author").

The entire process, composed by two only-mapping jobs, takes less than one hour on a single machine, due to high efficiency provided by HDFS' parallel distribution of the work. After this process, data is rewritten on the HDFS in a more clean format and is ready to be analyzed.

[6] https://spacy.io/
[7] http://www.nltk.org/

2. Task 1: Classification

The first task is to predict if a user liked the book or not according only on the text of the review (a binary-classification task). Target variable is a binary transformation of the number of stars of the review: if the user gave four of five stars to the book, it is considered "liked"; on the other hand, if the user gave three or less stars, he or she does "not like" the book.

2.1 Exploratory Analysis

Original ratings given by users are not distributed equally: following table shows that more than 80% of reviews give four or five stars to the ebook.

stars	frequency
1 star	257,185
2 stars	212,912
3 stars	477,640
4 stars	1,202,145
5 stars	3,323,996

But, having 5,473,878 observations, this percentage is not considered problematic. In fact, Apache Spark, in the case of binary classification, does not compute accuracy but AUC (Area Under - ROC - Curve), that is much more informative for unbalanced classes: after a simple optimization pipeline, the resulting model should not be influenced by the unbalance of classes.

2.2 Classification Pipeline

Loading the cleaned text from HDFS, all analysis is made directly in Spark with a simple pipeline that uses a 3-folds cross validation to estimate performance (using the mean) in a grid search:

- target variable is binarized;
- text is tokenized with a simple regex-tokenizer;
- n-grams are extracted;
- Tf-Idf matrix is calculated (Idf values are trained only on the train set), removing columns with less than 5 occurrences (considered noise according to Zipf's Law);
- Tf-Idf values are used to train a logistic model (using only the train set);
- *AUC* is computed on the test set.

Parameters of search for the grid-search optimization are the size of n-grams (starting from 1 up to 3) and the learning rate of the logistic regression (that is computed iteratively for optimization).

2.3 Results and Interpretation

Results of the optimization process are the following (on rows n-grams size, on columns learning rate value):

	0.1	0.05	0.01
1	0.90306	0.90317	0.90141
2	0.90640	0.90395	0.89798
3	0.77204	0.76828	0.76119

As the table shows, the best configuration found in the process is using bi-grams with a high learning rate. Poor performances yielded by the use of three-grams can be justified by an insufficient quantity of data: resulting matrix is very high-dimensional, so the risk of overfitting the train set is very high. On the other hand, while using bi-grams produces a matrix bigger than one obtained by mono-grams, it catches some constructions with semantics (such as name and surname of the author): for this reason their use performs better. Three-grams on the other hand catch even more complex constructions, but lack of data make them unusable.

2.4 Possible Improvements

Due to higher performance of bi-grams instead of mono-grams, one strategy to improve results consists of using a simple Ontology to support the tokenization of texts, replacing names of authors, characters or titles with the corresponding URI; in alternative, an Entity Recognition algorithm can be used. This is a very aggressive way to normalize names and titles, so that constructions such as "name + surname" are represented by just one token.

2.5 Possible Uses of the Model

Using texts written in natural language in a informal context, the Model should generalize well on texts taken from other social media (such as Facebook or Twitter). This model therefore can be tested on social media platforms to make market research or to integrate data from different sources in a Recommending System.

3. Task 2: Clustering

TODO

4. Conculsions

TODO

References

[1] Martin Thoma. The WiLI benchmark dataset for written language identification. 2018. arXiv: 1801.07779 [cs.CV].