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. In a second part a collaborative filter will be implemented to provide suggestions to the users according to their preferences. In the last part, some unsupervised algorithms are tested to clusterize reviews in similar groups: without a ground truth, a semantic analysis of the process is necessary to interpret results.

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 a Docker 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 precaution: 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 18.04) 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.
Another reason to use Apache Hadoop and Spark is their ability to parallelize computations to any number of cores or machines supplied using Scala or Python notebook as usual, in a transparent way to the user. In particular for this Project Hadoop 3 and Spark 3 are used: these versions are still in a beta phase of production, and have better performance and new functionality than their precedent versions. Data is always kept on the HDFS partition of hadoop to maximize Spark's performances (in this way data is already partitioned into small chunks).

Kindle Store is chosen corpus to avoid problems that physical products reviews have in common: a lot of reviews do

[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

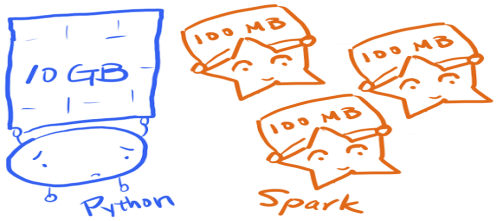

Figure 1. How spark manages memory in a more efficient way python while managing Big Data and large files in general.

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. 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, time-stamp (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`[5] library is used to filter reviews by language, removing those that are not in English or do not contain text: it uses a Nave Bayesian classifier based on 4-grams, to 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 in terms of accuracy[5, 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 one hour and half 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/`

Part I
Classification

2. Multi-label Classification

The first task is to predict the vote given by a user according only on the text of the review: while target labels are the number of stars from 1 to 5, input features will be extracted to encode documents in a vectorial space.

2.1 Exploratory Analysis

Ratings are not distributed equally: figure 2 shows that more than half of the reviews gave five stars; and, with the exception of the "2 stars" class, lower is the vote, less frequent it is.

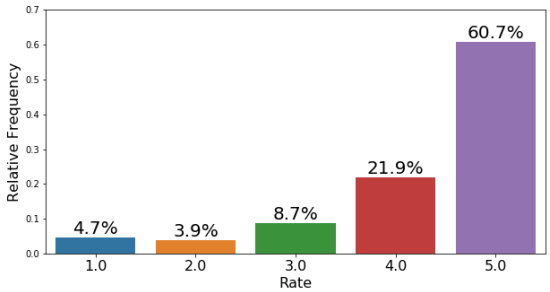

Figure 2. Distribution of review votes (measured in stars).

To achieve good results, f_1 measure is used as performance index instead of accuracy, and the model is optimized through a basic grid-search to find a configuration of hyper-parameters that generalize well on the task.

2.2 Pipeline

All analysis is made using Apache Spark, reading data directly from HDFS; then a Linear Support Vector Classifier (Liner SVC) is trained to predict classes in a one-vs-all manner: five models will be trained, each one to predict if observation belongs to its class or not. Six combinations of hyper-parameters are tried in order to get better performance; each combination is tried using 3-folds cross validation to estimate its score. Then the model with best average score is selected and analyzed in details. The pipeline is the following:

- clean 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 one-vs-all SVC model (using only the *train set*);
- f_1 is computed on the *validation set*.

The grid-search tries to maximize f_1 score changing size of *n*-grams and regularization parameter of the machine-learning model.

2.3 Results and Interpretation

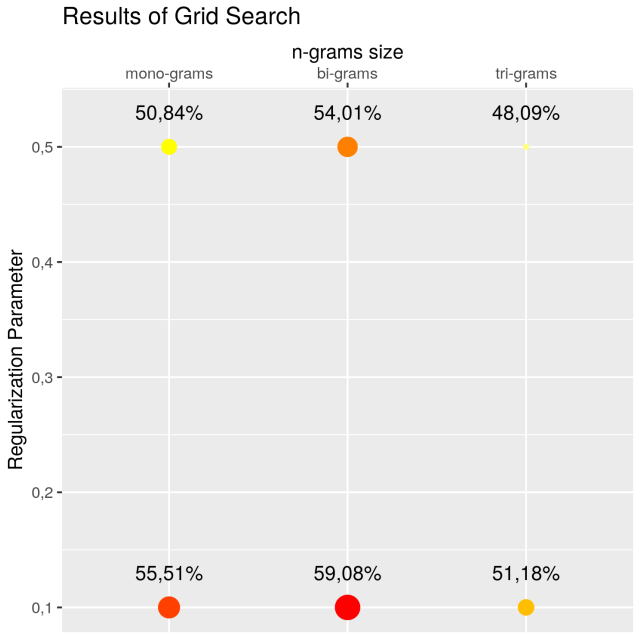

Figure 3. f_1 of the model with various combinations of hyper-parameters.

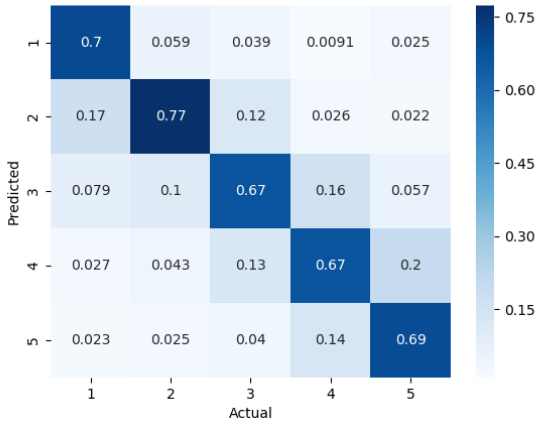

Figure 4. Confusion-matrix (normalized by the support) of the multi-label classification with the best configuration.

As shown in figure 3, using an SVC classifier, results depends both on Regularization parameter and n-gram size, with best results obtained with bi-grams: they can represent simple syntactic constructions (such as "name + surname") but without adding an high number of dimensions to the tf-idf matrix. Tri-grams on the other hand catch more complex constructions but dimensions of the tf-idf matrix is too big for the quantity of data used in this Project.

Analyzing the regularization parameter it is clear that reviews are linearly-separable using a tf-idf encoding: best results are obtained with a low value of this parameter.

Results of classification are shown in Figure 4: the confusion matrix shows that most reviews are well classified, and most errors occurs between two consecutive labels, although the model does not known that, semantically, are sorted.

3. Binary Classification

To improve classification performance, the task is simplified into a binary classification: the model does not predict how much the book has been liked by the reader but just if he or she liked the book. 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.

3.1 Exploratory Analysis

Original ratings given by users are not distributed equally: the two positive classes are the most frequent ones, with a relative frequency of more than 80%; but, having $5,473,878$ observations, this percentage is not considered problematic. In fact, a weigted f_1 was used also in this case being that is much more informative for both classes: after a simple optimization pipeline, the resulting model should not be influenced by the unbalance of classes.

3.2 Classification Pipeline

The pipeline is identical to the multi-label case, but for the binarization of target variable and the use of a simple logistic regression model instead of SVC. This is justified by the fact that better performance are achieved by SVC with low regularization parameter, so a linear model should be sufficient to achieve acceptable results saving training time. 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).

3.3 Results and Interpretation

As shown in figure 5, the best configuration found in the process is using *bi-grams* with a low *learning rate*. Poor performances yielded by the use of *tri-grams* can be again justified by the 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. *Tri-grams* on the other hand catch even more complex constructions, but lack of data make them unusable.

The f_1 on the rare class is 0.69, with precision$=0.85$ and recall$=0.58$, and an overall accuracy of 0.91

3.4 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 to obtain better tokens. Those are very aggressive ways to normalize names and titles, but catch semantics of texts in a better way.

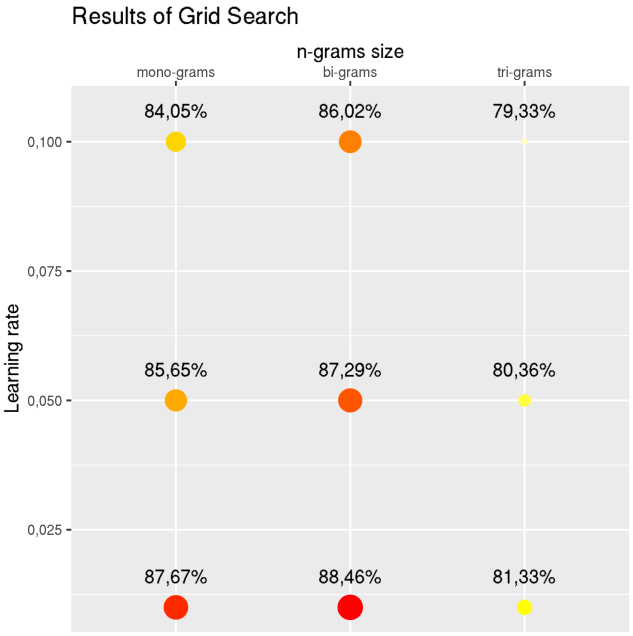

Figure 5. Results of grid search of binary classification. Values represent the weighted f_1 of the model using the corresponding configuration.

3.5 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.

4. Sentiment Polarity

To prove the correlation between review text and rate, a pretrained model[8] is used to assign a score between -1 and $+1$ to each observation. Raw text is used for this task, since even the stop words can change the polarity calculated by the TextBlob library. It can be useful to see how much the sentiment, which is considered as an ordinal value, is correlated to the rate value, which could potentially be an external regressor together with the tf-idf for future improvements.

The correlation between the two variables is $\rho = 0.37$: sentiment extracted from text can be useful for the vote prediction. It is clear that, greater is the average Sentiment Polarity, better the rating is going to be for a product. This can be also seen by taking the average sentiment for number of star:

[8] Using Python TextBlob library.

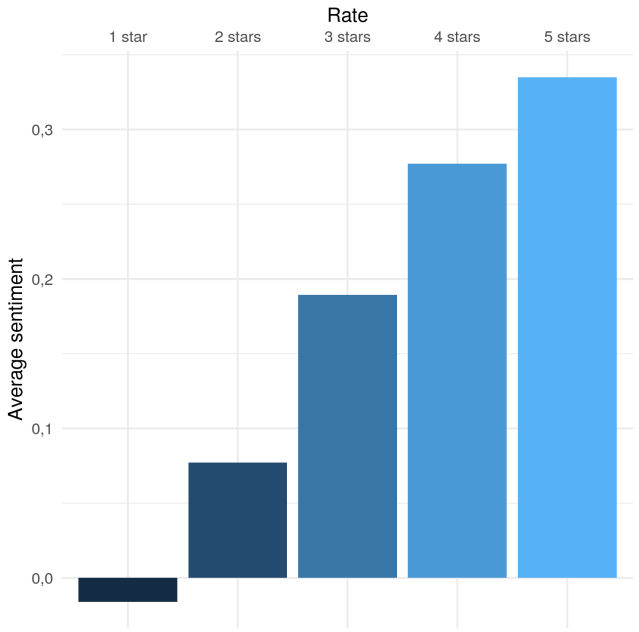

Figure 6. Average Sentiment Polarity for each review star.

Part II
Recommender System

5. Collaborative Filtering

The algorithm used to implement the collaborative filtering is ALS (*Alternating Least Squares*), included in Apache Spark. To optimize the process, Apache Spark does not store User-Product matrix R in a direct way but it decompose it into two matrix U and V (such that $R = U \times V$), calculated by an iterative process that yields an estimation. The iteration starts with random guess of the U and V matrix and alternatively use the least squares reach the approximation of the two matrices. This method is extremely efficient, since the SVD or any other way to factorize a matrix has a quadratic computational cost, while the ALS, at the cost of some precision, is able to do it in a limited number of iterations and it works well also with the implicit collaborative filters, in this case the rate was explicitly present.

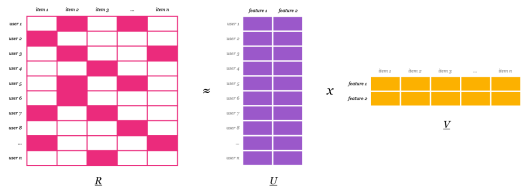

Figure 7. Representation of the ALS factorization.

Once factorized the R matrix, the V matrix is used to find items that are similar (using the dot product) and to make the recommendation for the i-th user is computed as: $score = U_i \times V^T$. So only the UserID, Product and the rating given to the product by the user is needed.

The hyper-paramters of this models are the regularization parameter λ, the number of iterations and the rank (number of features). A grid-search optimization (using a 3-folds cross validation) is used to find best configuration of hyper-paramters, using RMSE (*Root Mean Squared Error*) as performance index (trying to minimize it). Relevance score is the rating itself.

The time efficiency can be seen also by the fact that it takes around only one minute to fit the model and 1 minute to evaluate it in each fold of 3-folds cross-validation.

Results are shown in the Table 1:

λ	maxIter	rank	RMSE
0.05	10	5	0.9045
0.05	10	10	0.9294
0.05	10	20	0.9379
0.05	15	5	0.8934
0.05	15	10	0.9094
0.05	15	20	0.9058
0.1	10	5	0.8318
0.1	10	10	0.8332
0.1	10	20	0.8286
0.1	15	5	0.8262
0.1	15	10	0.8242
0.1	15	20	0.8171

Table 1. grid-search results for ALS.

Best configuration (highlighted in the Table 1) obtains an RMSE of 0.8171. The model improves as the number of iteration increases but the improvement is really small, so increasing the number of iteration might decrease the RMSE but the risk of overfitting will increase.

Part III
Clustering

6. Clustering

The data is also clustered on the processed dataset (stopwords removed, stemmatized and normalized), n-grams are extracted from it, followed by a tf-idf and finally there is the k-means with the euclidean distance.

The k-means (using the classic Lloyd algorithm) takes k random centers and iterates till coverrgence, which is a very simple and scalable algorithm but it has a few shortcomings:

- It takes many iteration (on average) to converge;
- It is very sensitive to the initial points;
- Random points can be initiated in the same cluster, and the k-means gets stucked in a local optima.

Actually the optimum initial points for the k-means is a NP-hard problem, and Llyod method trivializes it: using random sampling trying to solve a NP-hard problem in a constant time, thus degrading possibly the k-means performance. An alternative approach, used in this case is the k-means++ algorithm [1], which initializes the centroid in an intelligent way, here is how it works:

Algorithm 1: k-means++

Choose the first center, c_1, uniformly at random from the data set;
for $2 \leq i \leq k$ **do**
  $\forall x$ calculate $d(x)$, the distance between x and the nearest center;
  Choose c_i to be sampled from the data with probability $\propto d(x)^2$;
end
Proceed with the usual k-means algorithm;

The kmeans++ needs k passes on the data before actally starting the k-means algorithm and starts having trouble to scale with large data applications and big k. A variant has been developed, k-means||[2], which basically updates the distribution less frequently, oversampling each point with a larger probability, this variant is able to scale well and requires less passes on the data in initial phase, yielding similar results and is able to parallelize very well. The initial passes reduces the clusting effort later on, reducing the overall convergence time and the overall clustering results also improves, since it achieves near optimal initialization.

To find the optimal number of clusters k and the optimal number of n for the n-gram a grid-search is performed with $2 \leq k \leq 7$ and $1 \leq n \leq 3$, while trying to maximize the silhouette. The optimal cluster is with $n = 1$ and $k = 2$ with a silhouette$= 0.78$.

Once fixed the number of clusters k, the time required for executing the pipeline [Tf, Idf, Clustering, Evaluation] depends on the n chosen for the n-gram, in particular for n=1, the clustering required approximately 20 minutes, while for n=2 it required 17 minutes and finally for n=3 (which increases a lot the dimensions) required 85 minutes. Surprisingly the time does not change too much if the number of cluster changes.

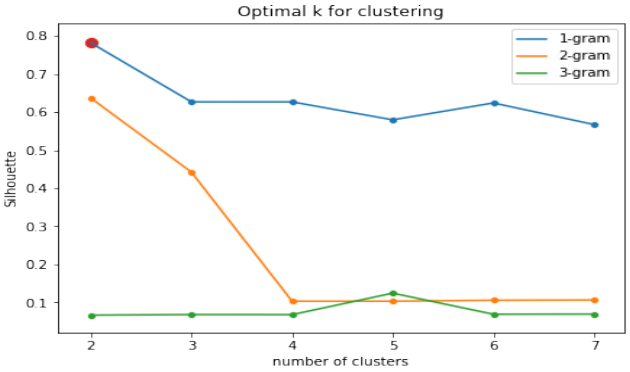

Figure 8. Optimal K-means and n-grams grid-search.

The actual number of optimal clusters are actually lower than expected, moreover the number of elements inside the cluster are surprising: the first cluster has $5,057,795$ elements and the second cluster has only $396,720$ elements. The two cluster's average rate is very similar, 4.22 and 4.31 respectively, so the distinction of the cluster is not given by the negative or positive words. The most frequent words of the two clusters overlaps, one option could be to remove these common words and try again, but this approach might not be the best option since some of these words seem to be important such as: story, great, enjoy, interesting etc.

A deeper analysis of the cluster might be needed to improve it, so another type of exploratory technique is used: *Topic Modeling*.

7. LDA - Topic Modeling

The basic hypothesis is the a document is made up of a set of arguments, which are characterized by a distribution of words. The objective of LDA [3] is to find the latent arguments (topics). Analyzing the similarity between the distribution of the words of the document and the topics helps in understanding the semantics of the text. Mathematically it tries to learn:

$$P(\theta, z, w|D, \alpha, \eta)$$

where θ is a distribution of topics, a k-dimensional random Dirichlet variable, z are the n topics for each document, w is the distribution of words for each topic, D is the corpus, α is the document-topic distribution and η is the topic-word distribution. Only words appearing in at least 30 documents are considered, the number of topics is fixed to 10 and the maximum number of iterations, given the "poor" results of k-means, was chosen to be 100. The algorithm converges in 3:30 hours and the topics found are: From the words describing the topics in Table 2, in the mostly all cases, the real

Topic				
1	2	3	4	5
love	picture	love	life	town
not	money	hot	god	murder
want	version	not	children	mate
get	color	wait	help	find
one	download	sa	love	friend
feel	not	seri(es)	us	famili(y)
relationship	ship	sex	kid	mysteri(y)
man	print	one	live	love
life	free	stori(y)	inspir(e)	emma
know	use	sexi(y)	stori(y)	bear
Topic				
6	7	8	9	10
novel	recip(e)	money	seri(es)	not
world	food	war	stori(y)	money
money	eat	busi(y)	charact(er)	like
stori(y)	diet	use	love	good
render	easi(y)	histori(y)	enjoy	no
charact(er)	inform	inform	not	time
tale	use	peopl(e)	great	get
fiction	cook	understand	next	edit
time	weight	work	read	use
one	healthi(y)	practic	end	word

Table 2. The topics with the top words describing it.

topics such as genre can be deduced, for example the second topic is about the kindle format of book itself, the third topic is about some erotic/romantic genre, the fifth topic seems to talk about some crime related books, the seventh topic about some cookbooks, the eight topic about some historical genre. While some topics most probably describes the feeling of the reviewer, the forth one talk about some inspiration in life and god, which can be either a gospel kind of genre or the feeling of the reviewer, the ninth one talks about enjoying and wanting to read some other books.

Conclusions

References

[1] Marcel R. Ackermann et al. “StreamKM++: A clustering algorithm for data streams”. In: *ACM Journal of Experimental Algorithmics* 17 (2010).

[2] Bahman Bahmani et al. *Scalable K-Means++*. 2012. arXiv: 1203.6402 [cs.DB].

[3] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. “Latent Dirichlet Allocation”. In: *Advances in Neural Information Processing Systems 14*. Ed. by T. G. Dietterich, S. Becker, and Z. Ghahramani. MIT Press, 2002, pp. 601–608. URL: http://papers.nips.cc/paper/2070-latent-dirichlet-allocation.pdf.

[4] Nakatani Shuyo. *Language Detection Library for Java*. 2010. URL: http://code.google.com/p/language-detection/.

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