Technical University of Crete

Diploma Thesis

A Recommendation System for Personalized Reviews in Apache Spark

---

Nektaria Mariolou

THESIS COMMITTEE :

Supervisor: Associate Professor Antonios Deligiannakis
Professor Minos Garofalakis
Associate Professor Michail G. Lagoudakis

A thesis submitted in fulfillment of the
requirements for the degree of Diploma in
Electrical and Computer Engineering

October, 2018

Abstract

Personalized recommendation systems play an increasingly growing role in customer's decision making process. The various e-commerce sites differ in their objectives, functions and characteristics but their common primary goal is to efficiently identify user decisions. The most frequent approach is the selection of the reviews with the highest percentage of helpfulness' votes by users who have read the reviews. Thus, they end up with a selection derived from a limited scope of criteria. In this work, we focus on retrieving a subset of reviews, using personalized criteria. In order to determine which set of reviews may correspond to individual users' preferences, we focus on the importance of product aspects for each user. Our system is built on Apache Spark enabling the processing of reviews, we evaluate it with a dataset from Amazon and the results are indexed in the distributed search engine, Elasticsearch.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my supervisor Prof. Antonios Deligiannakis for the continuous support for my thesis, for his patience, motivation and immense knowledge. Besides Mr Antoni Deligiannaki, I would like to thank the rest of my thesis committee for their insightful comments and their time to evaluate this work. And especially my family and friends for their support and patience.

Contents

1 Introduction

The outbreak of information has lead to the exponential explosion in the amount of data. Big Data is used to describe large amounts of data (structured, unstructured and semi-structured) which are being continually generated and analyzed in many fields such as Internet search, business social networks, social media, streaming services. By observing and comparing more data points, relationships will begin to emerge that were previously hidden, and these relationships enable us to learn and inform our decisions.
The all-encompassing term of big data refers to data that are characterized by gigantic volume, great variety and ubiquitous nature of its sources. Conventional databases cannot handle these large datasets neither individual machines support efficient processing capabilities. Thus, scalability and performance are key issues concerning large-scale data collection as much as its store and mining. Machine learning includes extremely useful data management tools for retrieving valuable information and also there is a proliferation of large-scale commodity clusters with distributed system architectures that are able to store and use the large datasets effectively.
This thesis presents a distributed and scalable system for streaming textual data analysis using Natural Language Processing (NLP) strategies which aim at aim at making personalized recommendations through supervised ML methods.

1.1 Motivation

With the tremendous growth of users, products made available on the web has become more difficult for users to handle the information overload problem. The emergence of e-commerce websites like Amazon, CNet and ebay has further boosted the volume of online information resources since they allow users to post their experience with products, and share information contents such as images, texts. Thus it has been possible to easily search for purchasing related advice given by other consumers. On one hand, the abundance of online information may virtually guarantee that users are able to find what they are looking for. On the other hand, the same abundance

makes it more difficult to handle and find the useful one.

User-generated reviews may contain a wide range of both objective and subjective product-related information, including features of the product, evaluations of its positive and negative attributes and various personal experiences. However, when a user will have to, in pages like Amazon for example, go through a couple of dozens of pages, he likely will not have the patience to read all of them. Therefore, the majority of users will read only the top reviews and their decision will be influenced from a small set of reviews. The problem is that the set that is going to determine the user's purchase is not adapted in his needs and interests. The reviews displayed in Amazon for example, are the ones that were highly rated by customers as the most helpful ones which may not include some important aspects of the reviewed item.

1.2 Thesis Layout

The rest of the thesis is composed by the following chapters, organized as follows: Chapter 2 is a study of related work that is of importance to the work described in this thesis and it is presented and explained. Also it describes the theoritical background of each component of our designed system. Specifically we explain the fundamental concepts about recommender systems and its main approaches. Also we present the concepts of the big data technologies that were used. In Chapter 3 we introduce the approach we followed for the recommendation of the top reviews set. In Chapter 4 we introduce our system, the architecture and communication between the different components are expounded, design and technological choices are justified. Chapter 5, presents the system's evaluation. The data is examined and experiments and statistics are illustrated and explained. Chapter 6 forms the conclusion of the work described. Finally Chapter 7, describes suggestions for possible future work.

2 Background

In this chapter, a brief description of the the related work as well as the theoritical work that we have mostly relied on is presented. A short background of the most well-known techniques used in recommendation systems until to date, as well as specific tools and components used for the implementation are analyzed in order to make the work presented comprehensible to the reader.

2.1 Related Work

Recommender systems have attracted attention in both academia and industry area. They became an independent research area in the mid-1990's when researchers started addressing recommendations relying on the ratings structure. However, the roots of recommender systems derive from an extensive work even earlier in the cognitive science [1], information retrieval [2] and also user choice modeling in marketing [3].

Amazon launched item-based collaborative filtering in 1998, and since 2003 [4] when the algorithm was presented in IEEE Internet Computing, it has been widely used by other popular web services too, like YouTube and Netflix. The main idea of Amazon's approach [5] is to define the item that will be unusually bought by a user who bought one item. However, the system does not rely just on the building of related items' table. Amazon by using e-mails, browse pages, product detail pages, user's past interests but also time's proximity between two purchases of items by a user has succeeded at making more and more successful recommendations.

2.2 Recommendation Techniques

In order to implement its core function the system must predict an item but mainly its utility. Some recommendation systems do not fully estimate the utility before making a recommendations but they may apply some heuristics to hypothesize that an item is of use to a user or other they estimate the utility based on other variables called "contextual". The basic models deal with two kinds of data which are the

user-item interactions, such as ratings or users purchase history or either the attribute information about the users and items such as textual profiles or relevant keywords. In the following, the basic models of recommendation systems are presented and analyzed.

Content-Based Recommender Systems

This type of system recommends items based on a comparison between the content of them and a user profile. The similarity derives from the data which are available from the user, either explicitly (rating) or implicitly (clicking on a link), any past activity of user and is calculated based on the features associated with the compared items. Based on that data, a user profile is generated, which is then used to make suggestions to the user. A problem arises in case where the quality has to be distinguished and for example two items may have the same characteristics but their quality is not.

Collaborative Filtering Systems

One of the most widely implemented and used is the one of collaborative. The basic concentration of this type of system is on finding similar users preferring similar items or a user expressing similar preferences for similar items. Collaborative filtering methods are often classified as memory based (user-based) or model-based (item-based) models. Their main difference with the content-based ones is shown in **Figure** 1

Memory-Based Collaborative Filtering Systems: This type of approach can be divided into two main sections: user-item filtering and item-item filtering. A user-item filtering takes a specific user and finds users that are similar to the target user. Users' similarity is calculated based on similarity of ratings and thus recommendations are items that similar users liked. Item-item filtering systems match item purchased or rated by the target user to similar items and combines those similar items and thus to makes the recommendation to the user. This approach is popular due to its simplicity, efficiency and ability to produce accurate and personalized recommendations. However an issue comes with this technique, the one called a "cold start" problem. When a new item is added to the system or a new user, there is no rating info so this approach is unable to generate a user profile.

Model-Based Collaborative Filtering Systems: Model-based collaborative filtering algorithms provide item recommendation by first developing a model of user ratings. These methods use a probalistic approach and envision the collaborative filtering

process as computing the expected value of a user prediction based on user's rating history on the rest remaining item list. In case the model is parametrized, the parameters of this model are learned within the context of an optimization framework. Known methods used for this type of systems are decision trees, rule-based models, Bayesian methods, Singular Value Decomposition and latent factor models. Model-based systems are less sensitive than the memory-based ones to some common problems like sparsity, cold-start problems, reliability (spam attacks for misleading recommendations).

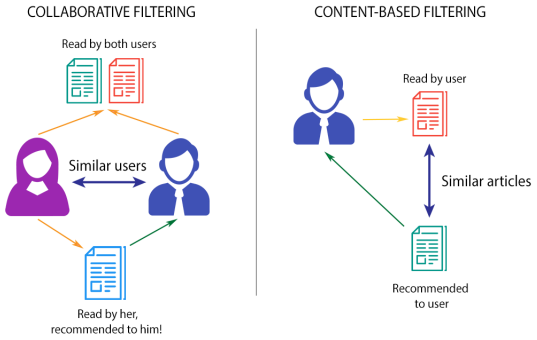


Figure 1: Collaborative filtering and Content-based filtering approach

Demographic Recommender Systems

In demographic recommender systems, the demographic information about the user is leveraged to learn classifiers in order to generate their recommendations, sometimes with the help of pre-generated demographic rules. The assumption is that different recommendations should be produced for different demographic areas. Many Web sites adopt simple and effective personalization solutions based on this type of method. For example, users are dispatched to particular Web sites depending on their language or country. Although in this type of systems the best results are not achieved on a stand-alone basis, they tend to improve the performance of other recommender systems as a component of hybrid systems.

Knowledge-Based Recommender Systems

The main goal of this method is to recommend items by using prior knowledge on how the item responds to user needs and preferences. Knowledge-based recommenders are divided in two subcategories: case-based and constraint-based recommenders.

Case-Based Recommendation: In case of case-based systems, their accuracy is judged based on a similarity function which estimates how much the recommended item responds to the user needs.

Constraint-Based Recommendation: Despite the fact that an association of product's features and user needs is generated too, in this approach of recommendation this association works as a constraint. This type of systems encourages the purchasing of the less frequently preferred items.

Community-Based Recommender Systems

This method is fully described by the epigram "Tell me who your friends are, and I will tell you who you are" [6], [7]. The recommendations are produced based on user's friends preferences. Open social-networks have invaded in people's lives and one could call them as a digital "reflection" of society. Thus, due to the fact that users tend to be influenced by friends' suggestions, this approach has been followed in recommendations in social-networks. In community-based method data referred to the social relation of the users and the preferences of the user's friends are required, in order to produce recommendation based on ratings that were provided by the user's friends.

Hybrid Recommender Systems

Hybrids systems are a very promising type of approach, as they are able to exploit the strengths of all the above types of systems and combine them to gain better performance and to mutually eliminate their weaknesses. For instance, collaborative filtering methods face the cold-start problem and so in cases for example where there is no knowledge of new-item ratings, it can use a content based method since the recommendation is based on items features. Some of the hybrid techniques are weighted, switching, mixed, feature combination, cascade, feature augmentation and meta-level.

2.3 Machine Learning

2.3.1 Text Mining

There is a wide range of technologies and focus areas in Human Language Technology (HTL) which consists of areas such as Natural Language Processing (NLP), Speech Recognition, Machine Translation, Text Generation and Text Mining. Natural Language Processing (NLP) and Text mining are Artificial Intelligence (AI) technologies that allows users to make fast transformation from the key content in text documents into quantitative, actionable insights.

NLP is a component of text mining that performs special kind of linguistic analysis that essentially helps a machine "read" text. A variety of techniques are developed to decipher the ambiguities in human language, including automatic summarization, part-of-speech tagging, disambiguation, natural language understanding and recognition. It includes a wide range of methods used such as stemming (removing suffixes) or a related technique, lemmatization (replacing an inflected word with its base form), synonym normalization, multiword phrase grouping.

Text mining is the process for analyzing large collections of written resources to generate new information, and to transform the unstructured text into structured. Text Mining includes areas such as automatic text classification according to a fixed collection of categories, text clustering, automatic summarization, extraction of topics from texts, documents, aspect segmentation and the analysis of topic trends in text streams.

2.3.2 User-Based Similarity Computation

In collaborative filtering where the recommendation is based on the users similarity the system a vast number of computations must be made in order to have the final rating predictions. Several different similarity functions have been proposed and evaluated in the literature such as Pearson Correlation, Cosine Similarity, Adjusted Cosine Similarity, Jaccard Similarity, Mean Squared Differences, Spearman Correlation etc.

Correlation: Correlation represents the extent to which two variables/vectors tend to change together. The coefficient describes both the strength and the direction of the relationship.

Pearson Correlation: This method [8] is one of the most commonly used and it finds the linear correlation between two vectors. A relationship is linear when a change in one vector is associated with a proportional change in the other. The range of the results' valued are between -1 and 1, where -1 represents a negative correlation

whereas 1 represents high positive correlation.

Cosine Similarity: A method [9] also used in collaborative filtering in recommender systems. Cosine similarity finds how two vectors are related to each other using measuring cosine angle between two vectors. The limit with cosine similarity is that it considers null preferences as negative preferences.

Spearman Correlation: Spearman Correlation evaluates the monotonic relationship between two continuous or ordinal vectors. In a monotonic relationship the vectors tend to change together, but not necessarily at a constant rate. Its limit is in case of partial orderings. Weak orderings occur when there are at least two items in ranking such that neither item is preferred over the other. When the system ranks same rated items at different levels, then Spearman correlation will be penalized for every pair of items rated same by the user.

Adjusted Cosine Similarity: Similarity computation in user-based collaborative filtering and item-based collaborative filtering differ in the computation. To be more specific, in the case of user-based the similarity is computed along the rows of the vector whereas in the case of item-based is computed along the columns. Basic cosine measure has one drawback while computing the similarity in item-based, as the rating scale between users are not taken into account. Thus, adjusted cosine similarity deals with that, by subtracting the corresponding user average from each co-rated pair.

Jaccard Similarity: The Jaccard similarity is estimated by comparing the members of the vectors and see which are shared and which are distinct. The similarity gets a range from 0

2.4 Big Data

Today's data deluge is largely due to the rise of computers and technology's ability transforming physical information into digital. Practically everything on the Internet is recorded. When a search occurs on Google or Bing, queries and subsequent clicks are recorded. Also when a purchase occurs on Amazon or eBay every click is captured and logged. Users read online newspaper, watch videos, track their financial transactions and this behavior is recorded too. The recording of individual behavior does not stop with the Internet: text messaging, cell phones and geo-locations, scanner data, employment records and electronic health records are all part of the data footprint that every user leaves behind.

Big data is used as a term referring to large amounts of those digital data. The term was introduced in 1997 [10], when the difficulty of taxing the capacities of

main memory, local or remote disks appeared, which lead to the need acquiring more resources. Globally, 98 percent of all today's stored data is in digital form compared to 2000 when only a quarter of the world's information was, while the rest was preserved on paper and other analogue media like film [11].

An aspect that has been noted as being a core resource when dealing with big data, pertains to the 3Vs: Volume, Velocity, Variety, a designation originally developped by Gartner [12]. Within the course of the following years, a variety of new dimensions were added such as Veracity, Variability, Value, Viability.

Volume concerns the exponential increase in the amount of data generated and stored. It's estimated that 2.5 quintillion bytes of data is created each day, and as a result, there will be 40 zettabytes of data created by 2020 which highlights an increase of 300 times from 2005. Velocity refers to the speed at which data is processed. The size of big data is overwhelming and today its volume is described by petabytes (PB), exabytes (EB), zettabytes (ZB) or even yottabytes (YB). What may be seemed big data today may not be the threshold in the future because storage capacities will increase and technologies will be evolved. However brontotype (BB) is expected to be the future data measurement unit.

Velocity refers to the speed at which data is generated, stored and streamed for analysis. The increase of integrated sensors in all types of devices contribute to the continuous influx of data. In terms of velocity and Big Data, it is easy to deal with the increased speed in which data is flowing into most organizations today, especially from data sources such as social media. However it is not just the incoming data that is important, but mainly the speed of real or near real processing data.

Variety corresponds to the structural heterogeneity in a dataset like structured, semi-structured and unstructured. Data may be derived from both internal and external data sources and adopt various formats such as transaction and log data from various applications, structured data as database table, semi-structured such as XML or even unstructured data such as text, images, video streams, audio and more. These different representations of data, various origins or complex problems such as high dimensionality that may be included, can cause difficulty in processing. Thus, a strong computational and comprehensive technology, which enable organizations to leverage data in their business process it is not just a challenge but a need too.

The industry leaders and academics agreed on the 3Vs as a standard but other dimensions, like the ones referred in the beginning of the section, characterize big data. Also important is the fact that these dimensions are not independent of each other. As one dimension changes, the likelihood increases that another dimension

will also be affected, too.

Big data combines all the features of data discussed before. Quite often data is produced at high velocity and need to be processed and analyzed in real-time but also sometimes data expires at the same high velocity. Data can be ambiguous, which makes its interpretation quite challenging. Traditional technologies of analyzing like Data Warehouses, relational databases can handle data defined by large volumes, various data types and from versatile data sources. The key to success is distributed parallelization based on a "share nothing" architecture and non-blocking networks ensuring a smooth communication among servers.

2.4.1 Apache Spark

Apache Spark is a fast, high-performance, distributed and general - purpose cluster computing system for large scale data processing. It has been built on top of Scala and can run in a standalone cluster mode that simply requires the Apache Spark framework and a JVM on each machine in your cluster. It was developed in 2009, at the university of California, Berkeley's AMPLab for processing data on large scale. In 2013 it was donated to Apache Software Foundation, which has maintained it since.

Spark has become the framework of choice when it comes in the processing of big data, overtaking the old MapReduce paradigm that brought Hadoop to prominence. The purpose of his initial design was to cover the gaps and limits encountered in specific applications that required repeated data access. MapReduce has proven to be a suitable platform to implement complex batch applications as diverse as sifting system log, running ETL, computing web indexes and powering personal recommendation systems. However, its reliance to persistent storage to provide fault tolerance and its once-pass computation model make MapReduce a poor fit for low-latency application and iterative computations.

Hadoop was initialized as a Yahoo project in 2006, becoming a top-level Apache open-source project later on. It consists of several components: the Hadoop Distributed File System named as HDFS, which are responsible for storing files in Hadoop-native format and parallelizing them across a cluster; YARN, a schedule that is responsible for the coordination application runtimes; and last but not least MapReduce, the algorithm that processes the data in parallel. Hadoop is built on top of Java but is also accessible through many programming languages like Python through a Thrift client.

Whereas Hadoop reads and writes files to HDFS, Spark processes data in RAM using a concept known as an RDD which stands for Resilient Distributed Dataset.

Spark can run either in stand-alone mode, with a Hadoop cluster serving as the data source, or in conjunction with Mesos. Spark keeps track of the data that each of the operators produces, and enables applications to reliably store this data in memory. This is the key to Spark's success, as it allows applications to avoid costly disk accesses.

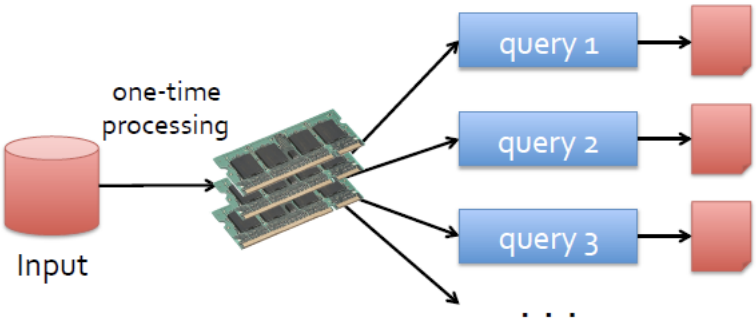


Figure 2: Low-latency computations (queries)

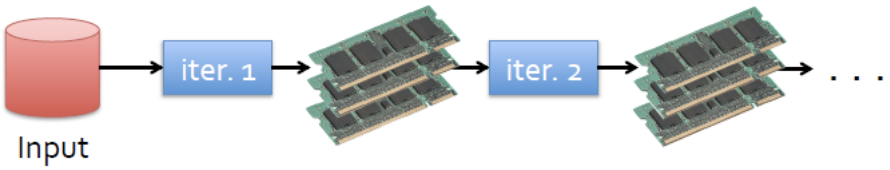


Figure 3: Iterative computations

Apache Spark is more suitable for batch based processing and real time processing, It offers the flexibility, a single unified framework and a programming approach to handle both batch based as well as real time. Whereas Storm focuses on stream or complex event processing. So it's a very powerful tool address different kind of processing needs.

The Spark Core engine uses RDD, as its basic data type which contains the collection of records which are partitioned. The RDD is designed in such a way so as to hide much of the computational complexity from users. It aggregates data and partitions it across a server cluster, where it can then be computed and either moved to a different data store or run through an analytic model. Each partition is one

logical division of data which is immutable and created through some transformation on existing partitions. However it comes with limitations regarding structured data as RDDs cannot take advantage of Spark's advanced optimizers including catalyst optimizer but also they don't infer the schema and must be declared from the user.

Like an RDD, a DataFrame is an immutable distributed collection of data. Unlike an RDD, data is organized into named columns, like a table in a relational database, or more suitable comparison is with Pandas dataframes in Python. It can process both structured and unstructured data efficiently. DataFrames allow the Spark to manage schema. Designed to make large data sets processing even easier, DataFrame allows to impose a structure onto a distributed collection of data, allowing higher-level abstraction.

Datasets takes on two distinct APIs characteristics: a strongly-typed API and an untyped API, as shown in the table below. DataFrame can be considered as an alias for a collection of generic objects Dataset[Row], where a Row is a generic untyped JVM object. Dataset, by contrast, is a collection of strongly-typed JVM objects.

Spark has been found to run 100 times faster in-memory compared to Hadoop for large scale data processing by exploiting in memory computing and other optimizations and 10 times faster on disk. It primarily achieves this by caching data required for computation in the memory of the nodes in the cluster. An important benefit that comes with this is that once in-memory, the data can be shared between the streaming computations and historical (or interactive) queries. It's also been used to sort 100 TB of data 3 times faster than Hadoop MapReduce on one-tenth of the machines.

Spark comes packaged with higher-level libraries including support for SQL queries, streaming data, machine learning and graph processing. These components increase the productivity and can be combined in order to create workflows. Spark Ecosystem has evolved since 2010, with the integration of various libraries and frameworks which allow faster and more advanced analytics than Hadoop and thus gaining popularity.

Spark comes with Spark SQL which is focused on the processing of query structured, using a dataframe approach like in R or Python (Pandas dataframes). It provides a standard interface for reading data and writing to other datastores like csv, son files or even a SQL like repository, HDFS, Apache Hive, JDBC, Apache Parquet. Other popular stores are NoSQL Databases like Cassandra, MongoDB, Apache HBase. Spark SQL also provides a SQL2003-compliant interface for querying data and this is the reason why it a powerful too for both analysts as well as developers.

Spark Streaming was an early addition to Apache Spark and is used to process data streams in real-time or near real-time. It uses micro batching which gives low

latency and integrates with a variety of popular data sources like HDFS. Previously, batch and stream processing in Apache Hadoop were separate things. Running on top of Spark, Spark Streaming enables powerful interactive and analytical applications across both streaming and historical data, while inheriting Spark's ease of use and fault tolerance characteristics. It really integrates with a variety of popular data sources like HDFS, Flume, Kafka and Twitter.

Machine learning has quickly emerged as a critical piece in mining Big Data for actionable insights. Built on top of Spark, MLlib includes a framework for creating machine learning pipelines that delivers both high-quality algorithms (e.g., multiple iterations to increase accuracy) and blazing speed (up to 100x faster than MapReduce). MLib comes with distributed implementations of clustering and classification algorithms such as k-means clustering and random forests that can be swapped in and out of custom pipelines with ease. Models can be trained in Apache Spark using Java, Scala, and Python as part of Spark applications.

GraphX is a graph computation engine built on top of Spark that enables users to interactively build, transform and reason about graph structured data at scale. It comes complete with a library of distributed algorithms for processing graph structures including an implementation of Google's PageRank. These algorithms use Spark Core's RDD approach to modeling data; with the GraphFrames package graph operation on dataframes can be performed taking advantage of the Catalyst optimizer for graph queries, too.

The ecosystem of Apache Spark described is displayed in **Figure** 4

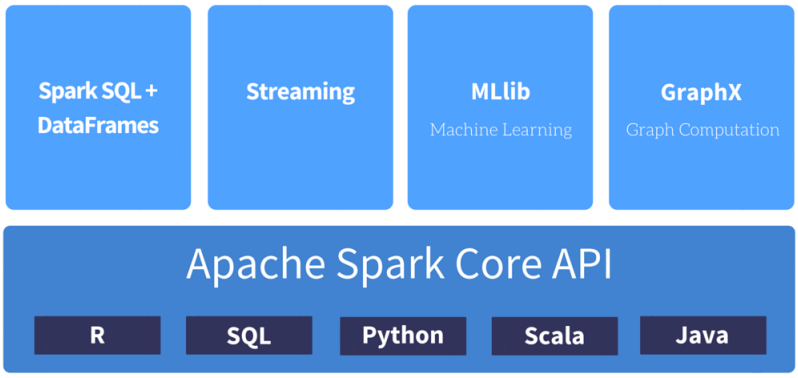


Figure 4: Apache Spark ecosystem

Apache Spark MLib

MLib which stands for Machine Learning Library is Apache Spark's machine learning and provides with Spark's scalability and ease-of-use when the users has to work with machine learning problems. The library includes an extensive collection of features regarding machine learning algorithms such as Regression (Linear, Generalized Linear, Decision Trees, Random Forests etc), Classification (Logistic, Decision Tree, Gradient-boosted tree, Linear Support Vector Machine etc.), Clustering (K-means, Latent Dirichlet allocation etc.), Collaborative Filtering. It also provides Featurization such as Feature extraction, transformation, dimensionality reduction and also selection but also Pipelines (tools for constructing, evaluating and tuning ML Pipelines), Persistence (saving and load algorithms, models, and Pipelines), Utilities (Linear Algebra, Statistics,Data handling).

It also includes Basic Statistics which includes the most basic machine learning techniques such as summary statistics (mean, variance, count, max, min and numNonZeros), correlations, Statified Sampling (these include sampleBykey, sampleByKeyExact), Hypothesis Testing (Peason's chi-squared test) and Random Data Generation (RandomRDDs, Normal and Poisson for random data generation).

One of the major advantage of Spark is the ability to scale computation massively which is necessary for machine learning algorithms. However the limitation is that all machine learning algorithms cannot be effectively parallelized as each algorithm has its own challenges for completing this, whether it is task parallelism or data parallelism.

John Snow Labs

Spark ecosystem includes also Spark Natural Language Processing library. The John Snow NLP Library is under Apache 2.0 license, written in Scala with no dependencies on other NLP or ML libraries. It basically extends the Spark ML Pipeline API. The framework provides the concepts of annotators, like NLTK, and comes out of the box with: Tokenizer, Normalizer, Stemmer, Lemmatizer, Entity Extractor, Date Extractor, Part of Speech Tagger, Sentiment analysis, Named Entity Recognition, Sentence boundary detection, Spell checker.

Additionally, given the tight integration with Spark ML, it offers more options when building NLP pipelines which includes word embeddings, topic modeling, stop word removal, a variety of features engineering functions like tf-idf, n-grams, similarity metrics...) and using NLP annotations as features in machine learning workflows.

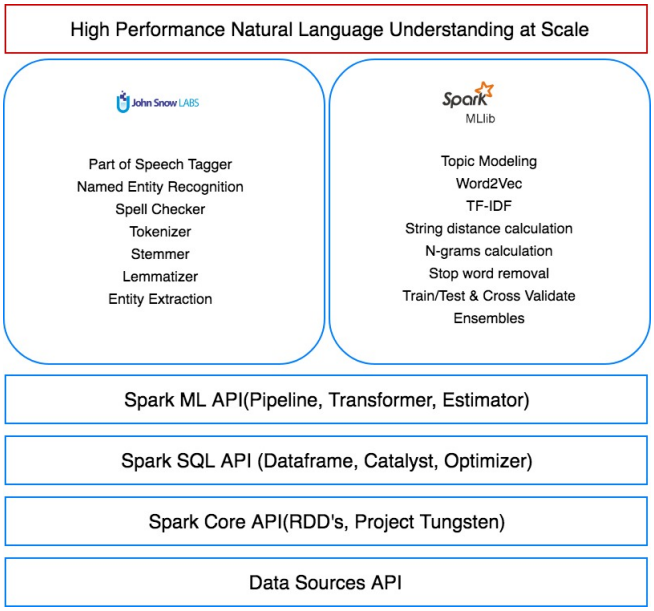


Figure 5: Spark ML and John Snow Labs

2.4.2 Elasticsearch

Elasticsearch is real-time open-source, broadly-distributed, readily-scalable, full-text search and analytics engine that is built on top of Apache Lucene. A platform designed by the Elastic company, alongside Logstash and Kibana, which are all designed to operate as an integrated package. It is suitable for storing, searching and analyzing large volumes of data in near real-time.

Elasticsearch uses Lucene as the core search engine but provides many features which are not part of Lucene. It is open source developed in Java, like Lucene, and licensed under the Apache license version. It is a cross platform and can operates in many systems. Elasticsearch was designed from the beginning to be scalable - distributed from the ground up. Also, it is designed to take data from any source, analyze it and make it searchable. Accessible through an extensive and elaborate API, Elasticsearch can power extremely fast researches, that support data discovery applications. It is near real-time, from the point in time that one adds, modifies or deletes a document the changes are propagated through the entire cluster within one or two seconds.

The communication with the search server is done through a HTTP REST API. The documents that are stored in Elasticserch are schema-less JavaScript Object Notation (JSON) documents much like NoSQL databases. This means that there is

no need to define fields and datatypes before adding data like in case of relational databases. Each document is inserted into an index, an abstraction which can be associated with the common databases of RDBMS, therefore an index consists of a collection of documents (e.g. product, account, movie) that denote marked similarity based on a characteristic.

The key concepts of Elasticsearch are the following. Cluster is a collection of nodes (servers) and consists of one or more nodes depending on the scale. A cluster provides indexing and search capability across all nodes. Node refers to a single running instance of Elasticsearch It stores searchable data and in case there are multiple nodes it stores part of it and not all data. Single physical and virtual server accommodates multiple nodes depending upon the capabilities of the physical resources like RAM, storage and processing power. Indexes are used when indexing, searching, updating and deleting documents within the index. There is no limit in the number of indexes that can be defined within a cluster depending on the scale of the project. Type/Mapping represents a set of documents sharing common fields present in the same index. An index can have one or more types defined, each with their own mapping. Document is a basic unit of information which can be indexed and defined in JSON format. Each document is associated with a unique identifier, called the UID. An index can be divided into multiple pieces called shards. A shard is a fully functional and independent index and can be stored in any node in a cluster. Shard allows to scale horizontally by content volume (index space) and to distribute and parallelize operations across shards, that is why it provides high performance. Finally, replica is a copy of shard and provides high availability in case of failure but also improves the performance of searching by carrying out a parallel search operation in these replicas.

3 Approach

With the advent and popularity of social network, more and more users choose to share their experiences, such as ratings, reviews and blogs. Users are overloaded and cannot have access to the information desired straight away. That happens in case of Amazon, the most popular e-commerce website. Users have to scroll down thousands of reviews in order to find the reviews which correspond to their preferences and they contain the information they are looking for.

Plenty of approaches for the web personalization have been proposed in the literature. We implement this [13] approach which tries to solve this problem. This methods recommends reviews not only of high quality and coverage of a variety of aspects but also focus more on the product aspects that are important to the user. Its goal is to improve the quality of the top reviews set displayed to the user by predicting which products aspects reflect the needs and preferences of the user. The problem that arises as shown in in **Figure** 6 and **Figure** 7 is that user cannot find quickly the reviews that address his/her needs due to the huge amount of reviews. Also the top reviews set includes the reviews which have the highest rating helpfulness.

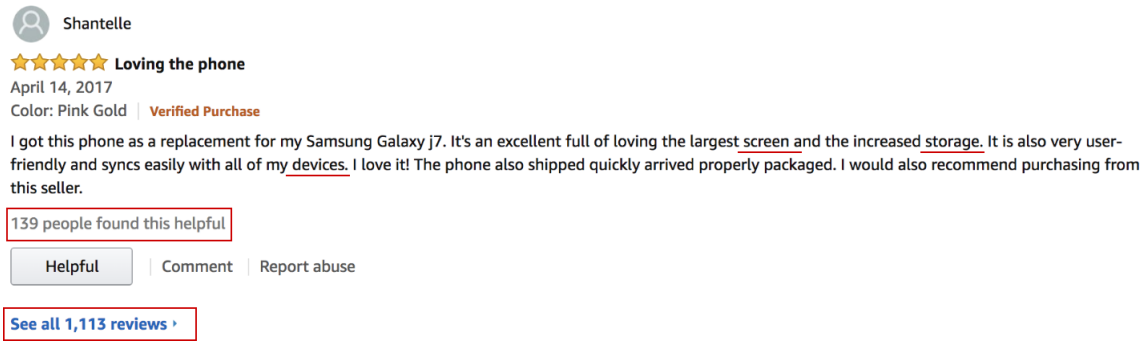


Figure 6: Graphical illustration of the problem definition and method's motivation

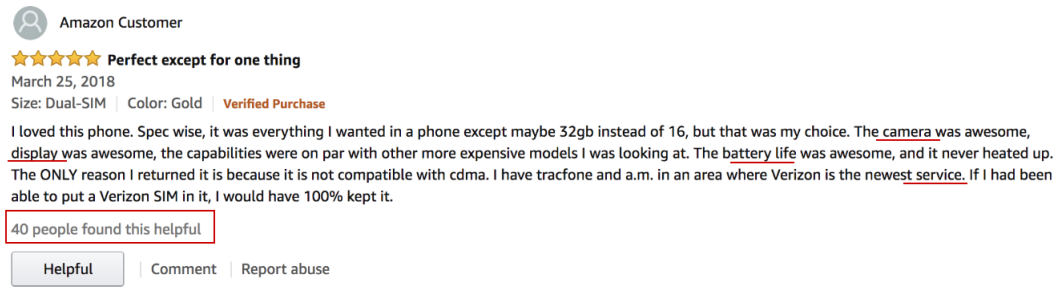


Figure 7: Graphical illustration of the problem definition and method's motivation

3.1 LARA algorithm

Latent Aspect Rating Analysis (LARA) is a novel text mining problem used to analyze opinions expressed in online reviews at the level of topical aspects. The goal is to map the sentences in a reviews subsets corresponding to each aspect. It is assumed that a few keywords describe each aspect and a boot-strapping algorithm is used to obtain more related words for each aspect. The algorithm takes a set of review texts with overall ratings and a specification of aspects as inputs, and discovers each individual's latent ratings on the given aspects and the relative emphasis a reviewer has placed on different aspects.

Algorithm: Aspect Segmentation Algorithm

Input: A collection of reviews $\{d_1, d_2, \ldots, d_{|D|}\}$, set of aspect keywords $\{T_1, T_2, \ldots, T_k\}$, vocabulary V, selection threshold p and iteration step limit I.
Output: Reviews split into sentences with aspect assignments.
Step 0: Split all reviews into sentences, $X = \{x_1, x_2, \ldots, x_M\}$;
Step 1: Match the aspect keywords in each sentence of X and record the matching hits for each aspect i in $Count(i)$;
Step 2: Assign the sentence an aspect label by $a_i = argmax_i\ Count(i)$. If there is a tie, assign the sentence with multiple aspects.
Step 3: Calculate χ^2 measure of each word (in V);
Step 4: Rank the words under each aspect with respect to their χ^2 value and join the top p words for each aspect into their corresponding aspect keyword list T_i;
Step 5: If the aspect keyword list is unchanged or iteration exceeds I, go to ***Step 6***, else go to ***Step 1***;
Step 6: Output the annotated sentences with aspect assignments.

Figure 8: LARA algorithm

For identifying product aspects, the methodology proposed in [14] is used. The

aspects that they declared by the user is shown in **Figure** 9.

Product Aspects	Aspect - related terms (seed words)
Value	value, price worth, purchase, budget
Screen	scree, touch, resolution, fingerprint, lighting
Camera	camera, photos, pictures, videos, analysis
Sound	sound, YouTube, voice, speakers, audio
Internet	wifi, network, antenna, 4g, 3g
Batter Life	battery, charger, battery life, long lasting,
Performance	speed, performance, processor
Accessories	earphones, headphones, case, adapter, gadget
Applications	games, apps, emails, maps, applications

Figure 9: Product aspects and aspect-related terms

After applying latent rating aspect analysis algorithm, we obtained the related product aspects **Figure** 10.

phone case screen battery price time product charger screen protector iPhone
device quality protector button protection color charge sound use thing cover
power review fit battery life ear cable back headset day plastic volume design
port one music problem way bit call work headphone size sound quality look
Bluetooth speaker unit camera volume love light hand port size look

Figure 10: Extracted product aspects

3.2 Personalized Top Reviews Set

3.2.1 Notations

In order to formulate the problem some notations in necessary to be set. We assume a set U of users, a set R of reviews and a set P of products. All products come from the same category of products in case of Amazon data, and share a set of product aspects A. Users write reviews to products. A review r_i^j is written by a user u_i for a product p_j.

3.2.2 Defining Ground Truth

The goal is that when a user u_i searches for a product p_j, a set of reviews for describing product p_j needs to be displayed. The *aspect-importance distribution* is a m-dimensional vector $F = f_1^{i,j}, f_2^{i,j}, ..., f_m^{i,j}$ which represents the importance level of

the review for the user. Based on the aspect set $A = a_1, a_2, ..., a_m$ each value $f_l^{i,j}$ corresponds to the importance level a_l of product p_j to user u_i.

The assumption that is made by the writers is that the number of words that a user uses to describe an aspect of the problem indicates that this aspect is in high importance to him/her. Thus the percentage of the words in the review r_i^j used for describing aspect a_l, corresponds to the importance level of aspect a_l for user u_i on product p_j. Supposing that $S_l^{i,j}$ indicates the sentences that are included in r_i^j and mention aspect a_l each F_i^j can be defined as follows:

$$\widehat{f}_l^{i,j} = \sum_{s \in S_l^{i,j}} \frac{N^w(s)}{N^a(s)} \tag{1}$$

and,

$$f_l^{i,j} = \frac{f_{l'}^{i,j}}{\sum\limits_{a_{l'} \in A} \widehat{f}_{l'}^{i,j}} \tag{2}$$

$\widehat{f}_l^{i,j}$ equals to the number of words used for describing aspect a_l in reviews r_i^j, $N^w(s)$ denotes the number of words contained in the sentence s and $N^a(s)$ is the number of aspects mentioned in s. In case that in the sentence more than one aspects are mentioned then the total words in the sentence is divided by the number of aspects contained. Finally $\widehat{f}_l^{i,j}$ is normalized so that the final $f_l^{i,j}$ sum to unity.

3.2.3 Prediction of Personalized aspect-importance distribution

The method proposed is based on a user-based collaborative filtering framework **Figure** 11. The limit of this approach as many of other approaches proposed in research community is the cold-start problem which occurs when there is a new user that has been registered to the system, in our case in Amazon website. In that case there is no prior information - history for the user and thus a personalized recommendation cannot be generated. Providing recommendations to users with no history record is a difficult problem for collaborative filtering techniques as their learning and predictive ability is restricted and in some cannot exist. Multiple researches have been developed approaching this problem with hybrid models.

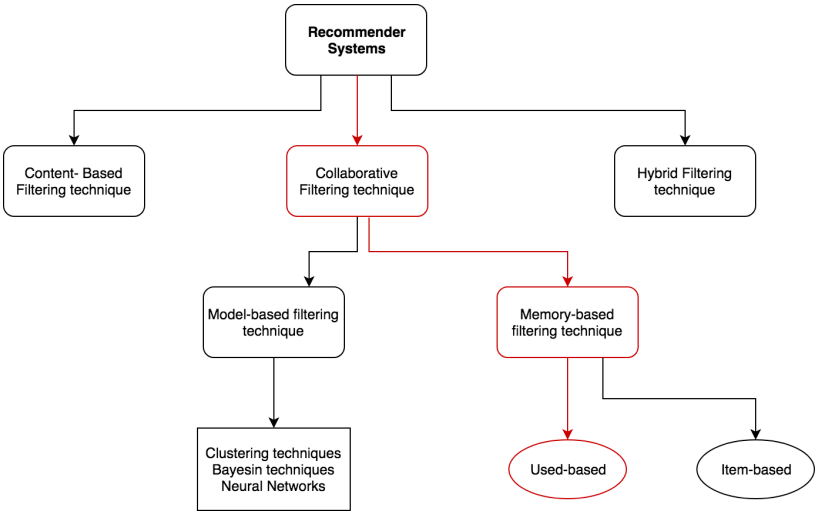


Figure 11: Used-based collaborative filtering framework used for the recommendation

When a new user u_i registers to the system and searches for a product p_j , there is no history record of the user reviews in other products. Therefore the real-aspect importance distribution cannot be predicted in order to retrieve a set of reviews based on a current F_i^j .

This approach focuses on the case where user's history is provided. By applying user-based collaborative filtering [15] , [16] it can be retrieved preferences from u_i's similar users regarding to target item to infer the preferences from $w^{i,v}$ to target item. Therefore when a user searches for a product, we aggregate all aspect-importance distribution F_i^v for each user u_v who has commented on the target product p_j; during the aggregation each F_j^v is weighted depending on the similarity between users u_v and u_i.

In order to evaluate similarity between users' preferences, we make the assumption that two users are similar if they always agree with each other regarding which aspects of products in P weigh more to them. The following equations describe the assumption referred.

$$w^{i,v} = \cos(l^i, l^v) \tag{3}$$

$$l^i = \frac{1}{|P^i|} \sum_{p_j \in P^i} F_i^v \tag{4}$$

$$l^v = \frac{1}{|P^v|} \sum_{p_j \in P^v} F_j^v \tag{5}$$

In the above equations l^i describes the average of aspect importance distributions of u_i's derived from user's historical reviews. In the same way l^v states for the average of historical aspect distributions of each user who has commented on product p_j. Thus $w^{i,v}$ is the user similarity between users u_i and u_v. Then we predict the

$$F_j^i = \frac{\sum\limits_{U_v \in U_j} w^{i,v} F_j^v}{\sum\limits_{U_v \in U_j} w^{i,v}}, \tag{6}$$

where U_j includes the users who have reviewed p_j.

The time complexity of the prediction of the aspect importance distribution is O(mn), assuming that the requests from users for products that they are interested in are m and the average number of user, for each product, that have commented is n.

3.2.4 Selection of Personalized aspect-importance distribution

The next step after the prediction of the aspect importance distribution is to retrieve the top reviews set. The goal is that aggregation of the set should not only have high quality but also a big number of aspects which indicated high coverage of aspects. However it's necessary that selection corresponds to the personalized criteria. Therefore it is suggested that the weight of each product aspect will be adjusted to the scoring function $F_p(R)$

$$F_p(R) = \sum_{a_l \in A} (f_l^{i,j} + \delta) f(R, a_l) \tag{7}$$

$$f(R, a_l) = \max_{r \in R_{a_l}} q(r) \tag{8}$$

where $f_l^{i,j}$ is the predicted importance distribution of aspect a_l of product p_j to a user u_i. The parameter δ gets a very small value. The addition of δ helps the reviews whose the prediction of aspect importance distribution is zero, to have a chance to be aggregated. Thus, the selection results avoid to suffer from data sparsity.

4 System Design

4.1 Data Source

For the evaluation of the algorithm, the experiments were conducted in real public datasets provided that have also been used in previous research in review mining. Due to the limitation of the definition of the aspects by the user, they were used only the reviews of "Cell phones and Accessories" category. The data used for the experiments come from Mr. Julian McAuley's lab [17] where a variety of datasets for recommender systems research are provided.

This dataset available contains product reviews and metadata from Amazon, including 142.8 million reviews spanning from May 1996 - July 2014. This dataset includes reviews (rating, text, helpfulness votes), product metadata (descriptions, category information, price, brand and image features). For the experiments the dataset which contains reviews of the category "Cell phones and Accessories" is used. As one can understands the aspects for cell phones and the aspects of accessories are different so a threshold in price is defined in order not include the reviews for cell phones' accessories. So metadata were used in order not to retrieve the reviews id after applying the price threshold. The total size of the reviews in category "Cell Phones and Accessories" is 2 GB and the number of the number is 3.447.249 reviews. The metadata is 346.794 products of size 410 MB. After applying the price threshold in metadata in order to exclude the reviews referring to Accessories, the dataset ended with 225.720 reviews.

The reviews includes the following fields: *reviewerID*: the ID of the reviewer, *asin*: product's ID, *reviewerName*: name of the reviewer, *helpful*: review's helpfulness rating, *reviewText*: review's text, *overall*: product's rating, *summary*: review's summary, *unixReviewTime*: time of the review in unix format, *reviewTime*: time of the review(raw).

The metadata includes the following fields: *asin*: product's ID, *title*: name of the product, *price*: poduct's price in US dollars (at time of crawl), *imUrl*: url of the product image, *salesRank*: sales rank information, *brand*: brand name, *categories*:

list of categories the product belongs to.

4.1.1 Data Preprocessing

In order to process the reviews and find the aspect terms, it is necessary to apply text cleaning. All the transformations that need to be applied, exist with NLTK which stands for Natural Language Toolkit, a free library for text-mining in Python which is included in SparkMLib too.

After loading the data into a pyspark dataframe,the column *helpful* that contains information about the helpfulness, is transformed into float number as it is provided in array format. This field is useful for the evaluation of the recommendation afterwards. Also the field *unixReviewTime* is transformed from unix format into date time in order to be able to apply recommendation afterwards. The final pyspark dataframe that is processed is shown in **Figure** 12

```
spark_df.show(6)
```

```
+--------------------+----------+-------+--------------------+-------------------+
|          reviewerID|      asin|helpful|          reviewText|     unixReviewTime|
+--------------------+----------+-------+--------------------+-------------------+
|A00861053U77CGZSG...|B0093HKM7M|    1.0|Sony Xperia is a ...|2012-11-24 00:00:00|
|A00861053U77CGZSG...|B004P8JL76|    0.0|First of all the ...|2012-11-24 00:00:00|
|A00861053U77CGZSG...|B008TSG8YM|   0.75|This is a very go...|2012-11-24 00:00:00|
|A01812943VUCELJJ8...|B0067M9JBQ|    0.0|excellent product...|2012-12-26 00:00:00|
|A01812943VUCELJJ8...|B0064JU5CO|    0.0|I received it in ...|2012-12-26 00:00:00|
|A02044053UEHK7HNS...|B00D89CW5Q|    0.5|Meet all my expec...|2014-04-28 00:00:00|
+--------------------+----------+-------+--------------------+-------------------+
only showing top 6 rows
```

Figure 12: Data to be processed in pyspark dataframe

The cleaning of the reviews text is applied. Each review is splitted into sentences and each sentence is splitted into words (tokenizing). All tokens are filtered out from standalone punctuation after they have been converted into lowercase. In addition we remove stopwords which do not contribute to the meaning of the text and NLTK provides a list of commonly agreed upon stop words for a variety of languages, English in our case.

4.1.2 Defining Ground Truth

After text mining, and loading the product aspect terms, LARA algorithm is applied for the aspect segmentation. After extracting the aspects from the algorithm,for each review r of each user u for each product p, the ground truth of the aspect importance distribution is estimated for each row.

```
train_data.head(6)
```

	asin	reviewerID	0	1	2	3	4	5	6	7	8	9
10458	B0093HKM7M	A00861053U77CGZSG1ZL1	0.0	0.00000	0.0	0.28125	0.425	0.000000	0.0	0.0	0.0	0.293750
10459	B004P8JL76	A00861053U77CGZSG1ZL1	0.0	0.43787	0.0	0.00000	0.000	0.278107	0.0	0.0	0.0	0.284024
13027	B0067M9JBQ	A01812943VUCELJJ8FG7C	0.0	0.00000	0.0	0.00000	0.000	0.000000	0.0	0.0	0.0	0.000000
8207	B00D89CW5Q	A02044053UEHK7HNSK3XS	0.0	0.00000	0.0	0.00000	1.000	0.000000	0.0	0.0	0.0	0.000000
8208	B002QSV6DY	A02044053UEHK7HNSK3XS	0.0	0.00000	0.0	0.00000	0.000	0.000000	0.0	0.0	0.0	0.000000
5349	B008VUZPUQ	A02553654ORZAIT2HWI4	0.0	0.00000	0.0	0.00000	0.000	0.000000	0.0	0.0	0.0	0.000000

Figure 13: The dataframe of the ground truth transformed in Pandas dataframe for better display

4.1.3 Prediction of aspect-importance distribution

In order to make our prediction we remove the review from the dataset and use the methodology described. We compare our prediction with the ground truth which is already computed, in order to test the accuracy of the approach. Each step described in the previous section in order to get the prediction of the aspect importance distribution, are seen in the figures below.

1 : Find past apect importance distribution of the target user

```
li = train.filter((F.col('reviewerID') == reviewerID_test) )
```

```
columns_list = sorted(list(set(li.columns) - {'asin', 'reviewerID'}))
drop_list = ['asin', 'avg)(asin)']
```

```
#train.groupBy('reviewerID').agg(F.avg('0')).show()
l_i = li.groupBy('reviewerID').avg()
l_i = l_i.select([column for column in l_i.columns if column not in drop_list])
```

2 : Find all users who have commented on the target product

```
lv = train.filter((F.col('reviewerID') != reviewerID_test) & (F.col('asin') == asin_test)).union\
    (test.filter((F.col('reviewerID') != reviewerID_test) & (F.col('asin') == asin_test)))
```

Find past aspect importance distributions of users who have commented in the target product

```
#train.groupBy('reviewerID').agg(F.avg('0')).show()
l_v = lv.groupBy('reviewerID').avg()
l_v = l_v.select([column for column in l_v.columns if column not in drop_list])
```

3 : Cosine similarity of target user with each user

Where l_iT and l_vT denote the transposed data frames. Once we transpose them, we can join them on the column na mes.In cos_sim you have three columns: the CustomerID from the first and second data frames and the cosine simil arity (provided that the values were normalized first). This has the nice advantage that we only have rows for ReviewerID pairs that have a nonzero similarity (in case of zero values for some ReviewerIDs)

```
kvs = F.explode(F.array([
       F.struct(F.lit(c).alias('key'), F.columm(c).alias('value')) for c in [columns_list]
      ])).alias('kvs')

l_iT = (l_i.select(['reviewerID', kvs])
        .select('reviewerI', F.column('kvs.name').alias('column_name'), F.column('kvs.value').alias('column_value'))
        )
l_vT = (l_v.select(['reviewerID', kvs])
        .select('reviewerID', F.column('kvs.name').alias('column_name'), F.column('kvs.value').alias('column_value'))
        )

l_vT = (l_vT.withColumnRenamed('reviewer1', 'reviewer2')
        .withColumnRenamed('column_value', 'column_value2')
       )
cos_sim = (l_iT.join(l_vT, l_iT.column_name = l_vT.column_name)
          .groupBy('reviewer1' , 'reviewer2')
          .agg(F.sum(F.column('column_value')*F.column('column_value2')).alias('cosine_similarity'))
         )
```

4 : Predict aspect importance distribution of the target user

```
df_joined = l_v.join(cos_sim, ['reviewerID'])
for col_name in df_joined.columns:
    if col_name != 'cos_sim' and col_name != 'reviewerID':
        df_joined = df_joined.withColumn(col_name, F.column(col_name) * F.column('cos_sim'))
```

```
drop_list = ['cos_sim']
df_joined = df_joined.select([column for column in df_joined.columns if column not in drop_list])
```

```
prediction = []
for col_name in df_joined.columns:
    if col_name != 'reviewerID':
        prediction.append(list(df_joined.select(col_name).groupBy().sum().collect()[0]))
prediction = [prediction[i][0] for i in  range (0,len(prediction))]
```

```
x = list(cos_sim.select('cos_sim').groupBy().sum().collect()[0])
prediction[:] = [i / x[0] for i in prediction]
```

4.1.4 Writing to Elasticsearch

After computing the aspect importance distribution for each user, which is going to be used as the ground truth for the evaluation of the algorithm. The pyspark dataframe is converted into json format in order to be stored in elasticsearch. Each tuple includes the following fields: reviewerID, asin, helpful, reviewText, unixReviewTime and the weight for each aspect. The data are organized in indexes and their type is defined when creating the index.

Write to ELS

```
from elasticsearch import Elasticsearch
# test your ES instance is running
es = Elasticsearch()
es.info(pretty=True)

create_index = {
    "mappings": {
        "aspect_importance": {
          "properties": {
                "reviewerID": {"type": "string"},
                "asin": { "type": "string"},
                "helpful": {"type": "double"},
                "reviewText": {"type": "string"},
                "unixReviewTime": {"type": "date"},
                "0": {"type": "double"},
                "1": {"type": "double"},
                "2": {"type": "double"},
                "3": {"type": "double"},
                "4": {"type": "double"},
                "5": {"type": "double"},
                "6": {"type": "double"},
                "7": {"type": "double"},
                "8": {"type": "double"},
                "9": {"type": "double"}
          }}}}
# create index with the settings and mappings above
es.indices.create(index="demo", body=create_index)
```

```
{u'acknowledged': True, u'shards_acknowledged': True}
```

Figure 14: Load aspect importance distribution to Elasticsearch

4.2 System Architecture

Every recommender system needs data for both items and users in order to work and provide recommendations of the user. Recommender systems have some drawback regarding latency which is a key factor in any personalization service but also cost. Calculating recommendations just-in time are very expensive and slow. Thus the best approach is to pre-calculate recommendations and when a recommendation request is received the only need will be to read the DB for the pre-calculated recommendations.

In our system, we load the reviews and calculate the aspect importance distribution. We consider a part of the dataset as train set in order to be able to evaluate the approach. As we mentioned the limit of the approach is that address non-cold start users. The results are stored in Elasticsearch. The rest of the dataset which includes the last reviews of the users (we consider that they have not written reviews for the

specific product yet) become the requests to our system.

The request to the system is formed as a user who wants a recommendation of k top reviews set for a specific product. Apache Spark computes the prediction for the user which is stored then to Elasticsearch too.

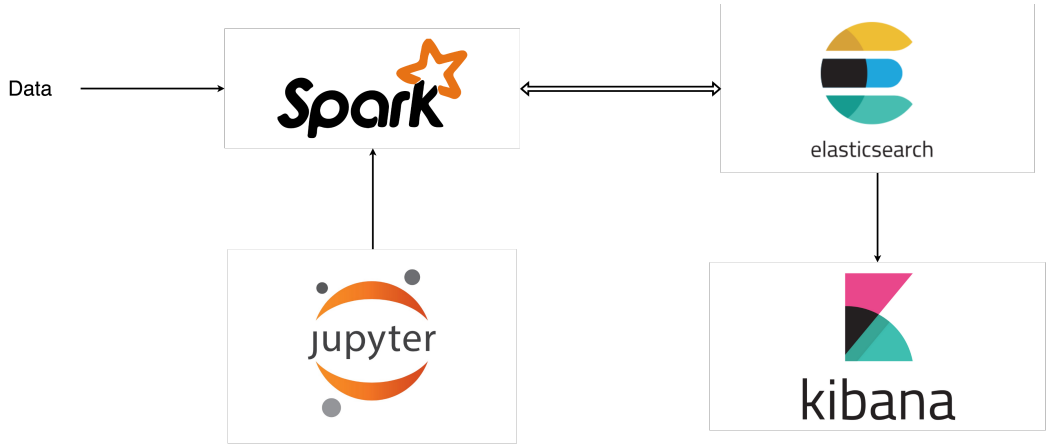


Figure 15: System architecture

5 Experiments

In this chapter we describe the results obtained after taking several experiments regarding. In the beginning the evaluation of the approach followed is described and it is compared regarding. Also the time processing and the time of top reviews set retrieval is examined. It should be mentioned that all experiments were performed on a single computer.

5.1 Evaluation of the proposed method

The set of the reviews that is used as a ground truth is the historical reviews for each pair of (u_i, p_j) and as test the most recent ones. In order to evaluate the method discussed, we use Cosine Similarity, Pearson and Spearman average correlations as a prediction accuracy in order to see how close is the prediction with the ground truth that we have already computed and store in Elasticsearch.

Table 1: Performance of predicting personalized aspect-importance distributions

Method	**On Amazon**
Cosine	0.61
Pearson	0.59
Spearman	0.57

5.2 Recommendation of top reviews set

There is a request from a user for a specific product that he/she is interested in and after defining the number of reviews that are going to be displayed to the user, the desired number of reviews is shown. We can see in **Figure** 16 and **Figure** 17 below the recommendation of the reviews for a target product requested from a specific user. In the example below the number of the reviews is set *k=2*. It is also displayed the first review which is recommended and some of the aspects mentioned in the review.

```
reviewerID_test = getrows(test,'reviewerID', rownums=[210]).collect()[0][0]
asin_test = getrows(test,'asin', rownums=[210]).collect()[0][0]
```

```
recommended_top_reviews_set = select_top_reviews(test,train,all_reviews,reviewerID_test, asin_test, k=2)
recommended_top_reviews_set.show()
```

```
+--------------+----------+------------------+--------------------+-------------------+
|    reviewerID|      asin|           helpful|          reviewText|     unixReviewTime|
+--------------+----------+------------------+--------------------+-------------------+
|A3223W5IROMYTY|B00D1VVQ4O|0.9528301886792453|Before I start my...|2013-06-27 00:00:00|
|A3VHGUZCGYA4KK|B00D1VVQ4O|               1.0|Is it OK to say I...|2014-01-04 00:00:00|
+--------------+----------+------------------+--------------------+-------------------+
```

Figure 16: Recommendation of the top reviews set with k=2

```
PD_recommended_top_reviews_set = recommended_top_reviews_set.toPandas()
print PD_recommended_top_reviews_set['reviewText'][1]
```

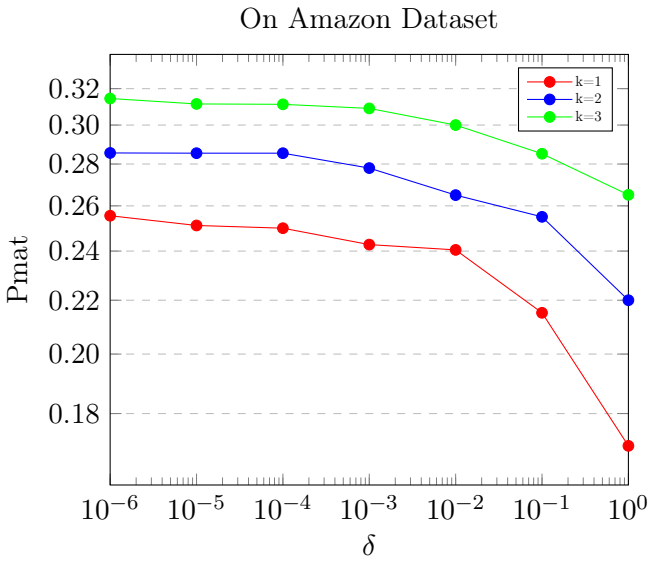
```
Is it OK to say I Love my smartphone? It does so many things for me. I can't imagine how we got along without them. O
f the many, many phones that  I have owned over the years, this is my favorite!  My previous phone was the Samsung Ga
laxy Note II, and I loved that phone, too, but my new lover: this Samsung Galaxy Mega 6.3 is divine! You can read all
about the specs for this phone from so many online reviews (as everyone should do their due diligence in researching
their next smartphone love), so I will just share what I really value and appreciate about this phone. The screen is
big, bright, and clear. It is so big that it really replaced my 7in Galaxy Tab (which I have sold to a good friend),
YET it still fits comfortably in my pocket. It is that perfect size sweet spot -- for me, it's the "Goldilocks" phon
e: not too big, not too small, but JUST right. With this single device I have a decent sized e-reader (Kindle app), t
ablet, phone, game system, and mini-computer. Compared to my previous Note II, this phone is just as fast, and I neve
r really used the stylus. There might be others comparing the Note series to the Mega, and as for myself, I haven't l
ooked back and do not miss my old Note II. I usually don't keep a phone for more than 6-8 months, but I have a feelin
g that I will keep this phone for some time, as it meets all my needs, and it looks gorgeous, too.
```

Figure 17: Review display

5.3 Selection of top reviews set

The goal of the recommendation system was to display to users reviews which contain aspects responding to their profile and preferences. Recall that this was managed through the aspect importance distribution extracted from a review and which reveals how much information used is used for each aspect. Thus, we evaluate the recommended top reviews set regarding to the information provided and to its importance to the user. For the evaluation we use the *Personalized match* (Pmat) metric of the review set $R_{i,j}$ which is the average correlation values between F_r for each review in the review set and F_j^i

$$Pmat_{i,j} = \frac{1}{|R_{i,j}|} \sum_{r \in R_{i,j}} Cor(F_j^i, F_r) \tag{9}$$



We have conducted some experiments, by changing the values of the *delta* in range from 10^{-6} to 1.0 so at to observe the performance of *Pmat*. In case the value of δ is small enough the performance is pretty stably and as it increases it is observed a sharped change. However, this behavior was expected as there is an inverse relationship between the aspect distribution and δ. Thus larger values in δ should be avoided in order to ensure a stable performance.

6 Conclusion

In this work, we attempted to implement a recommendation of top reviews set in the scope of targeting user profile and preferences. It is important for both websites to provide reliable and targeted collection of reviews and for users to exploit this, in order to have the desired information needed but also quickly. In order to achieve this, a prediction of the weight of each aspect for the target user is provided. In addition besides the high quality and high coverage is taken into consideration during the retrieval of the collection of the reviews. but also the predicted aspect-importance distribution. The experiments show that this method provide recommendations targeted to the user without impair the quality but also the high coverage.

This work could be extended, by addressing the problem of cold-start users having more information about their past activity like regarding the product they have seen or bought. Also this method could be tested in the rest categories of Amazon website but also in other websites which contains users reviews like Yelp, Tripadvisor.

Regarding the architecture that was implemented, an initial improvement would be that the estimation of the aspect distribution to be implemented in streaming process too. Also that would be applied in a cluster and conduct the experiments in order to evaluate its efficiency.

Bibliography

[1] E. Rich, "User modeling via stereotypes," *Cognitive science*, vol. 3, no. 4, pp. 329–354, 1979.

[2] G. Salton, "Automatic text processing. addison welsley," *Reading, Massachusetts*, vol. 4, 1989.

[3] G. L. Lilien, P. Kotler, and K. S. Moorthy, "Marketing models,(1992)."

[4] G. Linden, B. Smith, and J. York, "Amazon. com recommendations: Item-to-item collaborative filtering," *IEEE Internet computing*, vol. 7, no. 1, pp. 76–80, 2003.

[5] B. Smith and G. Linden, "Two decades of recommender systems at amazon. com," *IEEE Internet Computing*, vol. 21, no. 3, pp. 12–18, 2017.

[6] F. Ricci, L. Rokach, and B. Shapira, "Recommender systems: introduction and challenges," in *Recommender systems handbook*, pp. 1–34, Springer, 2015.

[7] D. Ben-Shimon, A. Tsikinovsky, L. Rokach, A. Meisles, G. Shani, and L. Naamani, "Recommender system from personal social networks," in *Advances in Intelligent Web Mastering*, pp. 47–55, Springer, 2007.

[8] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: an open architecture for collaborative filtering of netnews," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pp. 175–186, ACM, 1994.

[9] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge & Data Engineering*, no. 6, pp. 734–749, 2005.

[10] M. Cox and D. Ellsworth, "Application-controlled demand paging for out-of-core visualization," in *Proceedings of the 8th conference on Visualization'97*, pp. 235–ff, IEEE Computer Society Press, 1997.

[11] E. M. Micheni, "Diffusion of big data and analytics in developing countries," 2015.

[12] D. Laney, "3d data management: Controlling data volume, velocity and variety," *META Group Research Note*, vol. 6, no. 70, 2001.

[13] W. Tu, D. W. Cheung, and N. Mamoulis, "More focus on what you care about: Personalized top reviews set," *Neurocomputing*, vol. 254, pp. 3–12, 2017.

[14] H. Wang, Y. Lu, and C. Zhai, "Latent aspect rating analysis on review text data: a rating regression approach," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 783–792, ACM, 2010.

[15] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative filtering recommender systems," in *The adaptive web*, pp. 291–324, Springer, 2007.

[16] W. Pan, "A survey of transfer learning for collaborative recommendation with auxiliary data," *Neurocomputing*, vol. 177, pp. 447–453, 2016.

[17] H. Lakkaraju, J. J. McAuley, and J. Leskovec, "What's in a name? understanding the interplay between titles, content, and communities in social media.," *ICWSM*, vol. 1, no. 2, p. 3, 2013.