Università degli Studi di Milano
Facoltà di Scienze e Tecnologie
Corso di laurea magistrale in Informatica

Towards Semantic Powered Insight Engines

Author: **Davide Gavio**
930569

Supervisor: **Prof. Dario Malchiodi**
Co-Supervisor: **Prof. Andrea Condorelli**
Co-Supervisor: **Dr. Adriano Berni**

Academic year: 2020/2021

An heartfelt thanks:
to Professors Malchiodi and Condorelli,
to Adriano, Marco and all the other colleagues from artea.com,
to my parents, my grandparents and my whole family,
to my friends.

Introduction

Project overview

The project presented in this document has been realized at artea.com [2], which is a consulting tech company that delivers business-to-business AI solutions.
The idea behind the project was to investigate the possibility of exploiting Transformer generated embeddings in order to use them in a semantic context. During the years have been proposed several approaches to represent and compare textual documents, but with the introduction of transformer technology with BERT as a forerunner, the ability of better representing documents with vectorized embeddings has made a big leap. Those embeddings try to mimic the human ability to understand the context and the relations of different parts of a written text. In NLP field embeddings are used for several tasks: document similarity, sentiment analysis, document categorization, text summarization, but one of the main challenges is ranking documents given a query: this high level problem is reducible to the similarity between single documents task. This is the same idea that lays behind search engines: once a user inputs a query it is compared to online resources in order to return a list of pertinent results.
Following this idea led to the realization of a Proof of Concept (PoC) that consists of a "Semantic Search Engine" able to semantically rank documents given a query at search time. This approach is strongly based on Transformer technology, which gives a semantic dimension to the project instead of a statistical dimension based on TF-IDF or similar techniques.
The deliverable consists of a distributable framework able to agnostically manage the entire workflow, from data cleaning to document ranking, providing a series of metrics to evaluate performances.
In this project, it has been demonstrated that is possible to exploit pre-trained transformer-produced contextualized representation of a given document corpus and use them to perform semantic search in a document set for a given query. This approach proved to be distributable using a computing framework like Spark [80], being able to tackle even very big datasets. Lastly, the realized PoC is a

solution able to manage the entire pipeline, from data extraction to result computation, passing by different intermediate processing phases.

Manuscript organization

This document is organized in different chapters that cover all project aspects, from a theoretical point of view, to an experimental point of view. The chapters are divided in the following way:

1. **State of the art**: this chapter covers the state of the art for the project subject with bibliographic references;
2. **Technologies**: this chapter covers the theoretical aspect of technologies;
3. **Architecture**: this chapter describes the system architecture which has been implemented during the project period;
4. **Methods**: this chapter focuses on the analysis on the methods applied in order to obtain final results;
5. **Experiments**: this chapter contains a selection of experiments that have been completed during the project, in order to demonstrate the goodness of the work;
6. **Conclusions**: this chapter sums up all the work, focusing on what has been done and on what is still needed to complete exhaustively the project.

Contents

Abbreviations

AI Artificial Intelligence

ALBERT A Lite BERT

ASF Apache Software Foundation

BERT Bidirectional Encoder Representations from Transformers

BPE Byte-Pair Encoding

BoW Bag-of-Words

CBOW Continuous Bag-of-Words

DAG Direct Acyclical Graph

DFS Distributed File System

ELMo Embeddings from Language Models

FFN Feed-Forward Network

GFS Google File System

GLOVE Global Vectors for Word Representation

GPT Generative Pre-trained Transformer

GPU Graphic Processing Unit

HDFS Hadoop Distributed File System

HTML HyperText Markup Language

I/O Input/Output

JSON JavaScript Object Notation

KNN K-Nearest Neighbors

LDA	Latent Dirichlet Allocation
MLM	Masked Language Modeling
MST	Minimum Spanning Tree
NLP	Natural Language Processing
OCR	Optical Character Recognition
PLM	Permuted Language Modeling
PoC	Proof of Concept
RDD	Resilient Distributed Dataset
ReLU	Rectified Linear Activation Function
SOA	Service-Oriented Architecture
STS	Semantic Textual Similarity
TF-IDF	Term Frequency-Inverse Document Frequency
YARN	Yet Another Resource Negotiator

List of Figures

Chapter 1

State of the art

1.1 Text representation

In order to be used, textual data needs to be represented in a more usable format. Textual representation can be divided in not contextualized and contextualized: not contextualized representation focuses only on words or subwords, without looking neither to local nor global context. On the other hand, contextualized approaches, try to draw the big picture, representing texts in the corpus emphasizing the context and what are the relationships among words. Contextualized approaches try to give a more semantic representation of texts.

1.1.1 Not contextualized

As previously said, not contextualized textual representations focus only on words or subwords, without giving importance to local or global context. In this paragraph there is a list of the main not contextualized approaches to represent a text.

Bag of Words (BoW)

A BoW is a way for extracting features from textual data. First of all BoW creates a vocabulary of terms present in document set. Using this vocabulary of words, it measures the occurrences of words in a single document. At the end, a document will be represented as a bag of its words that keeps the concept of multiplicity using term frequency. Every other information apart from term frequencies are discarded. A short example is visible below:

Sentence 1: *The cat sat on the hat*
Sentence 2: *The dog ate the cat and the hat*
Vocabulary: {the, cat, sat, on, hat, dog, ate, and}
BoW Sentence 1: {2, 1, 1, 1, 1, 0, 0, 0}
BoW Sentence 2: {3, 1, 0, 0, 1, 1, 1, 1}

In text similarity context, if two documents are similar their bags of words will be similar looking at term frequencies. The above example shows two different sentences: looking at just the vocabulary and the BoWs, it is noticeable that the two original sentences have something about the *cat* in common, but the rest of the two sentences are very dissimilar. Some examples of BoW implementations are shortly described here below.

1. **Bag of Tricks for Efficient Text Classification**: in [35], BoW is used to represent sentences in order to define a baseline for text classification.
2. **Microblog Sentiment Analysis Based on Cross-media Bag-of-words Model**: in [76], Bow is used to provide a unite representation of both text and image from microblog messages. This representation is then used to classify the sentiment of the message using a logistic regression.
3. **SOFTCARDINALITY-CORE: Improving Text Overlap with Distributional Measures for Semantic Textual Similarity**: in [34], BoW is used to represent pair of texts with an associated weight for each word. This representation is then used to test three novel similarity measures presented in the paper.

TF-IDF

TF-IDF represents an enhancement of the BoW approach: it is a numerical measure that works on term-document pairs. TF-IDF consist of:

1. Term frequency: refers to the number of times that a term t occurs in document d

$$\begin{gathered} d = document \\ t = term \\ f_t = \text{occurrences of term } t \text{ in document } d \\ n = \text{total number of terms that appear in the document } d \\ \text{TF}(t) = f_t/n \end{gathered}$$

2. Inverse term frequency: is a measure of whether a term is common or rare in a given document corpus. It is obtained by dividing the total number of documents by the number of documents containing the term in the corpus [44]

$$\begin{gathered} N_d = \text{total number of documents} \\ N_t = \text{number of documents containing term } t \\ \text{IDF}(t) = \ln(N_d/N_t) \end{gathered}$$

   The final result is given by the product of TF and IDF terms. An high TF-IDF score for a given couple (t, d) means that the document is relevant in the document d but it is rare in the whole document corpus.

Some examples of TF-IDF implementations are shortly described here below.

1. **KNN with TF-IDF Based Framework for Text Categorization**: in [73], TF-IDF is used to determine a weight matrix of documents that will be further used with KNN algorithm to classify documents into groups that describe the content of the documents.

2. **A novel TF-IDF weighting scheme for effective ranking**: in [46], a normalized version of TF-IDF is used to capture two different aspects of term saliency. One component of the term frequency is effective for short queries, while the other performs better on long queries. The final weight is then measured by taking a weighted combination of these components, which is determined on the basis of the length of the corresponding query. Furthermore, TF-IDF has been used to constitute a baseline used to evaluate the TF-IDF variations proposed in the paper.

3. **Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents**: in [52], TF-IDF is used to find most relevant keywords from a collection of twenty random websites belonging to four different domains. Such keywords can describe those documents and can be used as tags to categorize the content.

Word embeddings

Word embeddings are a multidimensional representation for text. Word embeddings techniques represent words as real-valued dense vectors in a given vector space. The idea is to give a similar representation for words that are used in a similar way, in opposition with approaches like BoW where words with similar usage have different representation (unless explicitly managed). Some examples of widely used text embeddings techniques are shortly described here below.

1. **Word2Vec**: given a large corpus of documents, Word2Vec is able to give a word representation based on its usage and occurrences. Word2Vec is a shallow two-layer neural network. Word2Vec has two different architectures: skip-gram and CBOW (Continuouos BoW). CBOW is similar to a feed-forward neural network, it tries to predict the correct word given a list of context words. To do that, after the network training, when a word is given in input Word2Vec produces a vector embedding using the hidden weights from the hidden layer. On the other hand, with the skip-gram approach, Word2vec predicts context words, which are words that surrounds the input word. Like CBOW, to produce an embedding vector for an input word,

the network outputs the weights coming from the hidden layer after being trained [41]. Some examples of Word2Vec applications are shortly described here below.

(a) **Sentence Similarity Calculating Method Based on Word2Vec and Clustering**: in [69], Word2Vec has been used to extract the semantic information of a given text and for retraining after a clustering phase. The approach consists in:

    i. corpus preprocessing;

    ii. Word2Vec training word vectors;

    iii. clustering of document bodies by K-means algorithm;

    iv. training the clustering results twice to get the word vector results after training;

    v. calculate the similarity based on the result of retraining.

(b) **Semantic relatedness and similarity of biomedical terms: examining the effects of recency, size, and section of biomedical publications on the performance of word2vec**: in [81], Word2Vec has been used to derive a semantic representation of biomedical publications. This representation is then used to compute the similarity with biomedical terms from the same dataset. Reference standards are used to perform an evaluation of the performances of representations produced by Word2Vec.

2. **GloVe**: this approach relies on the idea that ratios of co-occurrence probabilities between words may encode a word meaning inside a multidimensional vector. The training objective is to learn word vectors such that their dot product equals the logarithm of the words' probability co-occurrence. The loss function associates the ratios of co-occurrence probabilities with vector differences in the word vector space [49]. An example of GloVe applications is **Semantics derived automatically from language corpora contain human-like biases**: in [8], GloVe has been used to produce a representation of words from the dataset. Those vector representations are then used to demonstrate that human *biases* are propagated to Artificial Intelligence technologies.

1.1.2 Contextualized

Contextualized embeddings have become the state of the art of vectorized representation of text. Incorporating context into word embeddings — as exemplified by

BERT, ELMo, and GPT-2 — has proven to be a watershed idea in NLP. Replacing static vectors (e.g., Word2Vec) with contextualized word representations has led to significant improvements on virtually every NLP task [4]. Browsing *Papers with code* website [6], which collects information about the scores achieved by different approaches on several NLP tasks, is possible to notice that every leaderboard of every different task is populated mostly by contextualized approaches like transformers. Different tasks like *Sentiment Analysis* [62], *Semantic Textual Similarity* [59], *Text Classification* [71], *Community Question Answering* [10], *Question Answering* [53] and others, have all been performed by transformer technology with improved performances.

Transformers

Until massive adoption of transformer technology, the dominant sequence transduction models were based on complex recurrent or convolutional neural networks that include an encoder and a decoder. Transformers are a network architecture based only on attention mechanism, not explicitly exploiting recurrence and convolution, differently from previously adopted system like recurrent neural networks, long short-term memory and gated recurrent neural networks. Transformers are used to draw global dependencies between input and output. They are used to better handle long-range dependencies within a text. These technologies allow to represent texts in a more semantical way, better capturing context and relations among different parts of the text [74]. Transformers are the true state of the art in NLP field, since 2018 (year of BERT [17][45] release) transformers are used in different fields that require a semantic representation of texts. Some examples of transformer applications are shortly described here below.

1. **tBERT: Topic Models and BERT Joining Forces for Semantic Similarity Detection**: in [48], a transformer model called BERT (will be explained further in the document, in *Experiments* chapter) is used to encode sentence pairs that are used with a topic model such as LDA to enhance BERT with topics addition. The result is an even better framework that shows performance improvement across all dataset present in the study.

2. **SCIBERT: A Pretrained Language Model for Scientific Text**: in [3], the base version of BERT has been trained on a large corpus of scientific text to address the lack of high-quality, large-scale labeled scientific data. It is then evaluated on a series of different tasks, such as: Named Entity Recognition, PICO Extraction, Text Classification, Relation Classification, Dependency Parsing, showing improved results compared to the base version of BERT.

1.1.3 Textual similarity/distance

During the years, different similarity/distance functions have been applied to compute the similarity/distance between documents. Depending on the textual representation, there are two most used similarity measures: *Cosine similarity* and *Jaccard similarity.* In addition, *Euclidean distance* can be used to measure the distance between two documents. From the introduction of vectorized representations (contextualized or not contextualized) like transformers, the standard *de facto* for measuring similarity has become the Cosine similarity. The current state of the art of text similarity computation relies on two similarity functions and one distance function.

1. **Cosine similarity**: the Cosine between two vectors is used to measure their (cosine) similarity. For Cosine similarity, the input text pair needs to be represented as a couple of vectors. An example of text representation is the *term-frequency vector*, in which is contained the frequency of each word in the document. Let x and y be two vectors for comparison:

   $$\text{sim}(x, y) = \frac{x \cdot y}{\|x\| \cdot \|y\|}$$

   where $\|x\|$ is the Euclidean norm of vector $x = (x_1, x_2, \ldots, x_p)$ defined as $\sqrt{x_1^2 + x_2^2 + \cdots + x_p^2}$. A Cosine value of 0 means that the two vectors are at 90 degrees to each other (orthogonal) and have no match. The closer the Cosine value to 1, the smaller the angle and the greater the match between vectors [28] [23]. Some examples of Cosine similarity applications are shortly described here below.

   (a) **Contextualized Embeddings based Transformer Encoder for Sentence Similarity Modeling in Answer Selection Task**: in [38], Cosine similarity is used as measure for detecting similarity between question answering type. For example, while choosing the correct answer for a given question, it is necessary to rank the candidate answers to find the most suitable ones. Cosine similarity is used to rank candidate answers, using both contextualized and not contextualized embedding vectors.

   (b) **SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation**: in [9], Cosine similarity is the scoring method used for the 2017 Semantic Textual Similarity (STS) shared task, which is a venue for assessing the current state of the art.

2. **Euclidean distance**: it consists of the straight-line distance between two

vectors, the square root of the sum of squared differences between corresponding elements of the two vectors. The two vectors, like Cosine similarity, need to be numerical vectors so a given text must be converted to be used as input [23]. Given x and y vectors, the Euclidean distance is computed as follows:

$$\mathrm{d}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

The Euclidean distance is less used than Cosine and Jaccard similarities, still in the paper **Document Clustering Based On Text Mining K-Means Algorithm Using Euclidean Distance Similarity** [40] it is visible how Euclidean distance has been used for tasks like cluster distance computation in K-Means clustering algorithm.

3. **Jaccard similarity**: the Jaccard similarity [39][23] of sets S and T is the ratio of the intersection size of S and T to the size of their union.

$$S,\ T \text{ sets}$$
$$J(S, T) = \frac{|S \cap T|}{|S \cup T|}$$

Unlike Cosine similarity or Euclidean distance, Jaccard similarity works on sets, so for a document an intuitive representation is the set of unique words for a given document. Jaccard similarity measures the proportion of common words to number of unique words in both documents and ranges between 0 and 1, with the first meaning that the two sets are completely disjoint, while the second states that the two object are exactly the same. An example of Jaccard similarity usage is the paper **Measuring Performance of N-Gram and Jaccard-Similarity Metrics in Document Plagiarism Application** [18], in which Jaccard similarity is used to score the similarity of a pair of documents in order to detect plagiarism.

Chapter 2

Technologies

2.1 Cluster computing

Modern applications require to deal with immense amount of data as quick as possible. In many of these applications, data is extremely regular, so there is opportunity to exploit parallelism. Examples of possible applications are:

1. Web page ranking;
2. Social networks analysis;
3. Parallelizable computations.

This parallelization is not achieved using a super computer that can handle the computation, but is achieved using “computing clusters”. Computing clusters are large collections of commodity hardware, including conventional processors, connected by Ethernet network. The software stack begins with a new form of file system, called a “distributed file system”, which features much larger units than the disk blocks in a conventional operating system. Distributed file systems also provide replication of data or redundancy to protect against the frequent media failures that occur when data is distributed over thousands of low-cost compute nodes [39].

2.1.1 Big data

Big data [21] is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation [5].

1. **Volume**: the magnitude of data. Big data sizes could be terabytes or even petabytes.
2. **Variety**: the structural heterogeneity in a dataset. Data could be structured, semi-structured or unstructured.

3. **Velocity**: data generation rate. Data analysis speed is a strongly connected concept: the higher the velocity, the higher the needed analysis speed.

Other mentioned dimensions that are not part of the "core" definition:

1. **Veracity**: the unreliability of some data sources. For example, social media sentiments are an uncertain data source, because they imply human judgement.
2. **Variability**: the variation in data flow rates. Data could be produced by sources at not constant rates.
3. **Complexity**: the heterogeneity of data sources. Being generated through a myriad of sources, big data must be cleaned and connected in order to be used.
4. **Value**: the importance of big data. Being characterized by a "low value density", a big quantity must be analyzed in order to obtain an high value and be remunerative.

2.1.2 Distributed file system

To exploit cluster computing, data must be organized differently from the conventional file systems of single computers. This new organization is called Distributed file system (DFS) and is used as follows:

1. files can be very large, up to terabytes in size;
2. files are rarely updated. They are read as input for calculation and possibly additional data is appended to files.

Files are divided into chunks. Chunks are replicated on different computer nodes. A collection of several computer nodes connected by the same network switch constitutes a rack. If chunks are replicated on different racks, like in Figure 2.2, a rack failure won't cause any data loss. Examples of DFS are:

1. Google File System (*GFS*) [22], the first of the class;
2. Hadoop Distributed File System (*HDFS*) [64], an open-source DFS used with Hadoop;
3. Colossus [11], an improved version of GFS.

2.1.3 Hadoop

Hadoop is an open source framework overseen by Apache Software Foundation which is written in Java for storing and processing of huge datasets with the cluster of commodity hardware.

History

Hadoop was created by Doug Cutting and Mike Cafarella in 2005. It was originally developed to support distribution for the Nutch search engine project. Apache Nutch project was the process of building a search engine system that can index 1 billion pages. After a lot of research on Nutch, Cutting and Cafarella concluded that such a system will cost around half a million dollars in hardware, and along with a monthly running cost of $30000 approximately.
In 2003 Google solved half of their problem: releasing a paper about GFS [22], supplying a solution for the problem of storing very large files which were being generated because of web crawling and indexing processes.
In 2004 Google again released a paper about a distributed computation engine called MapReduce [15], which was the solution of processing those large datasets. Cutting and Cafarella started implementing Google's techniques (GFS and MapReduce) as open-source in the Apache Nutch project. In 2006, while working at Yahoo!, he separated the distributed computing parts from Nutch and formed a new project Hadoop with a large team of engineers working on this project.
In 2008, Yahoo! released Hadoop as an open source project to ASF(Apache Software Foundation). And in July of 2008, Apache Software Foundation successfully tested a 4000 node cluster with Hadoop.
In 2009, Hadoop was successfully tested to sort a PB (PetaByte) of data in less than 17 hours for handling billions of searches and indexing millions of web pages.
In December of 2011, Apache Software Foundation released Apache Hadoop version 1.0.
Currently, there is Apache Hadoop version 3.3.x which released in June 2021 [26].

Architecture

Apache Hadoop is a framework for running applications on large clusters built of commodity hardware. The Hadoop framework transparently provides applications for both reliability and data motion. Hadoop implements a computational paradigm named MapReduce, where the application is divided into many small fragments of work, each of which may be executed or re-executed on any node in the cluster. MapReduce works on YARN (Yet Another Resource Negotiator), which performs 2 operations that are Job scheduling and Resource Management. The Purpose of Job scheduler is to divide a big task into small jobs so that each job can be assigned to various slaves in a Hadoop cluster and Processing can be Maximized. Job Scheduler also keeps track of which job is important, which job has more priority, dependencies between the jobs and all the other information like job timing, ... And the use of Resource Manager is to manage all the resources that are made available for running a Hadoop cluster. In addition, it provides a distributed file system (HDFS) that stores data on the compute nodes, providing very high aggregate bandwidth across the cluster. Both MapReduce and the Hadoop Distributed File System are designed so that node failures are automatically handled by the framework [24] [25].

2.1.4 HDFS

HDFS relies on a *master-slave* architecture composed by a master server, the *NameNode*, and from several slave server, the *DataNodes*, generally one per node in the cluster. The NameNode manages the file system namespace and regulates access to files by clients. Each DataNode manages storage attached to the nodes that they run on. In Figure 2.1, is visible a simple schema of HDFS architecture.

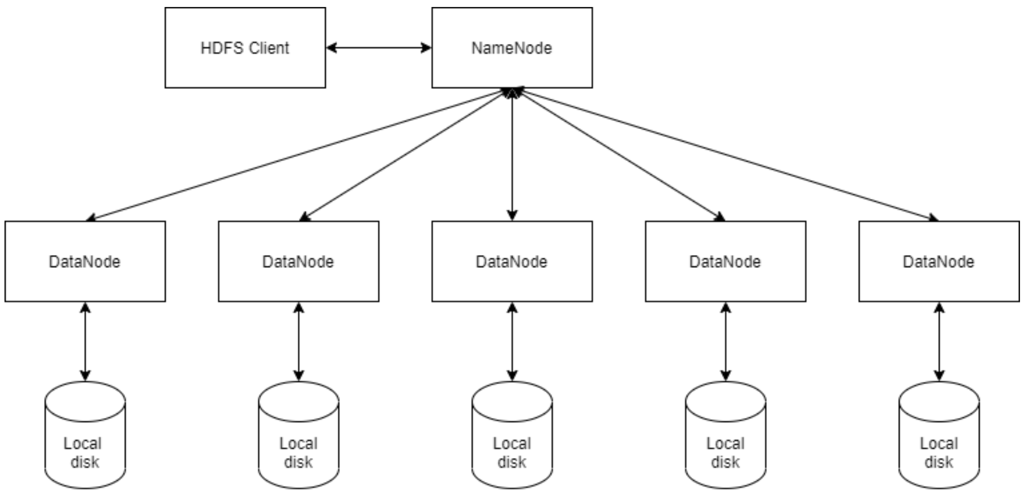

Figure 2.1: HDFS architecture

HDFS Blocks

HDFS breaks down a file into smaller units. Each of these units is stored on different machines in the cluster. This, however, is transparent to the user working on HDFS. To them, it seems like storing all the data onto a single machine. These smaller units are the blocks in HDFS.

HDFS NameNode

NameNode is the master node that runs on a separate node in the cluster. Manages the filesystem namespace which is the filesystem tree or hierarchy of the files and directories. Stores information like owners of files, file permissions, ... for all the files. It is also aware of the locations of all the blocks of a file and their size. All this information is maintained persistently over the local disk in the form of two files: Fsimage and Edit Log. Fsimage stores the information about the files and directories in the filesystem. For files, it stores the replication level, modification and access times, access permissions, blocks the file is made up of, and their sizes. For directories, it stores the modification time and permissions. Edit log on the other hand keeps track of all the write operations that the client performs. This is regularly updated to the in-memory metadata to serve the read requests. Whenever a client wants to write information to HDFS or read information from HDFS, it connects with the NameNode. The NameNode returns the location of the blocks to the client and the operation is carried out.

HDFS DataNode

DataNode are the worker nodes. They are inexpensive commodity hardware that can be easily added to the cluster. Datanodes are responsible for storing, retrieving, replicating, deletion, ... of blocks when asked by the NameNode. They periodically send heartbeats to the NameNode so that it is aware of their health. With that, a DataNode also sends a list of blocks that are stored on it so that the NameNode can maintain the mapping of blocks to DataNodes in its memory.

Rack awareness

HDFS stores replicated datablocks in at least two different racks in order to improve data availability, reliability and network bandwidth utilization. If the so called replication factor is three, a block is replicated on the local rack and two more replicas are placed in a same remote rack. This allows data recovery in case of connection loss among nodes. The inability to reach a rack (e.g. network problems) doesn't cause a big data loss because data are replicated over different racks, so lost blocks are read from the remaining rack and replicated again in order to guarantee availability, reliability and network bandwidth utilization. In Figure 2.2 is visible the rack organization of a DFS.

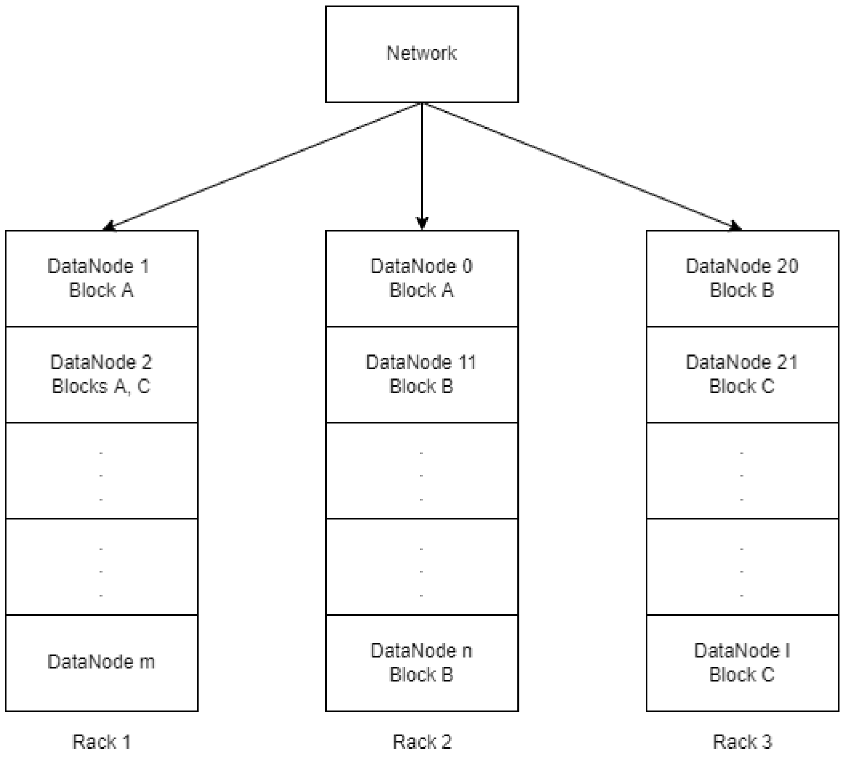

Figure 2.2: HDFS rack architecture

2.1.5 MapReduce

MapReduce [15] is a distributed computing framework that is used to manage many large-scale, parallel computations in a way that is tolerant of hardware faults. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the HDFS are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster, increased computational performances and efficiency. A MapReduce computation consists of:

1. Map task: takes in input one or more data blocks and returns a sequence of *key-value* pairs;
2. Reduce task: combines all the values associated with a key, one key at a time.

The execution overview is visible in Figure 2.3

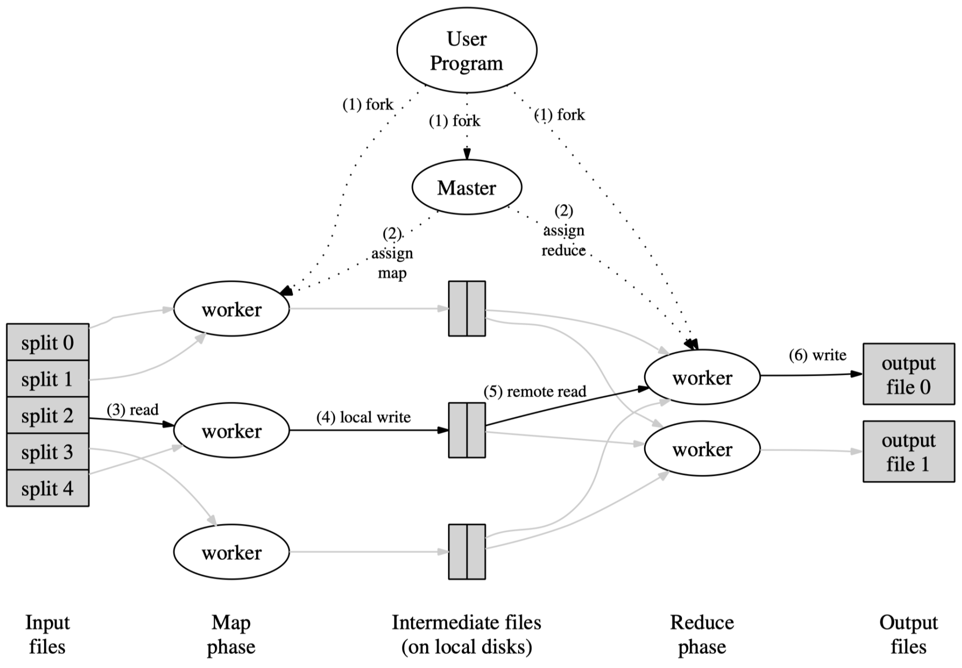

Figure 2.3: MapReduce execution overview

Map task

The input files for a Map task are elements of any type: tuple, document, ..., organized in key-value pair $(k,\ v)$ A block is a collection of elements, and no element is stored across two chunks. A Map task produces zero or more key-value pairs, with keys not necessarily uniques.

Grouping by key

Once the Map task is over, all key-value pairs are grouped by key, with a single list of values associated to that key.

Reduce task

A Reduce task takes as input a pair of key, list of values associated to that key. The task produces a sequence of zero or more key-value pairs as output. The function that is applied on the list of values for a given key is called *reducer*. For each Reduce task, there could be more reducers.

MapReduce execution

1. The user program forks a Master controller process and some Worker processes at different compute nodes. A Worker handles Map tasks or Reduce tasks, but not both.
2. According to the user program, the Master creates a given number of Map and Reduce tasks and assigns them to the Workers.
3. The Master keeps track of the status of each Map and Reduce task (idle, executing, completed) and when a Worker finishes a task, the Master schedules a new task for that Worker.

Node failure

1. If the Master fails, the entire MapReduce job must be restarted;
2. If a Map worker fails, all its tasks (even completed), are rescheduled to other workers to complete them eventually;
3. If a Reduce worker fails, its Reduce tasks will be put to idle status and rescheduled on another Reduce worker.

Execution management

The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The work of JobTracker is to manage all the resources and all the jobs across the cluster and also to schedule each map on the TaskTracker running on the same DataNode since there can be hundreds of DataNodes available in the cluster. The TaskTracker can be considered as the actual slaves that are working on the instruction given by the JobTracker. This TaskTracker is deployed on each of the nodes available in the cluster that executes the Map and Reduce task as instructed by JobTracker.

2.1.6 Spark

Apache Spark is a unified analytics engine for large-scale data processing. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Structured Streaming for incremental computation and stream processing [70].

Workflow systems Workflow systems extend MapReduce from the simple two-step workflow (the Map function feeds the Reduce function) to any collection of functions, with an acyclic graph representing workflow among the functions. That is, there is an acyclic flow graph whose arcs $a \to b$ represent the fact that function a's output is an input to function b [39].

Data structures

RDD Spark is based on Resilient Distributed Datasets (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. An RDD is distributed because it can be split into chunks and held at different compute nodes. They're resilient because it's expected that they can be recovered in case of partial or total loss. An RDD can contain elements of any kind, without restrictions [54].

Dataset A combination of DataFrame and RDD. It provides the typed interface that is available in RDDs while providing the convenience of the DataFrame. The Dataset API is available in the Java and Scala languages [14].

Dataframe A DataFrame is a Dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood [13].

Operations

Operations in Spark are:

1. Transformations: operations applied to an RDD that produce another RDD;
2. Actions: perform operations on an RDD and pass back the result to the application that called the Spark program.

Spark is and advanced workflow system, because of:

1. a more efficient failure handling;
2. a more efficient way of grouping tasks among compute nodes and scheduling execution of functions;
3. integration of programming languages features and function libraries;
4. RDD caching: caching RDDs allows to reuse them in order to reduce computation avoiding recalculating already cached operations, improving performances.

Map, FlatMap and Filter

Differently from MapReduce approach, in Spark Map functions can be applied not only on key-value pairs, but can be applied to any kind of object, producing one object as result. FlatMap is the same operation of Map in MapReduce, but without the requirement that all types be key-value pairs. Filter is a transformation that produces an RDD with only the objects that return true to a given predicate.

Reduce

Reduce operation is an action. Because of this, it returns a set of values — possibly smaller then the set taken in input — and not another RDD. On an RDD, Reduce is applied on every pair of consecutive elements until only one lement remains, that becomes the result of the Reduce operation.

Spark implementation

Like Hadoop or other MapReduce implementations, Spark manages large RDDs in the same way: it divides it in splits (MapReduces's chunks) and spreads them to different compute nodes in order to allow parallel computation.

Lazy evaluation The first difference from other MapReduce implementations, is that Spark is lazy evaluated: transformations are not applied on an RDD until it is required to do so, typically in correspondence of an action.

Resilience of RDDs Performing operations on RDDs creates a *lineage* of those RDDs, so that in case of failure Spark is able to recreate the RDD (or just a split) looking at its lineage.

Other Spark components

Apart from Spark *core* components, the engine has several other parts important depending on the application scope.

Spark SQL Spark SQL lets you query structured data inside Spark programs, using SQL. SQL provide a common way to access a variety of data sources, including Hive, Parquet, JSON, ...

MLlib Apache Spark provides a general machine learning library — MLlib — that is designed for simplicity, scalability, and easy integration with other tools. The key benefit of MLlib is that it allows data scientists to focus on their data problems and models instead of solving the complexities surrounding distributed data (such as infrastructure, configurations, and so on). The data engineers can focus on distributed systems engineering using Spark's easy-to-use APIs, while the data scientists can leverage the scale and speed of Spark core. Spark MLlib is a general-purpose library, providing algorithms for most use cases.

ML Pipelines When running machine learning algorithms, it involves a sequence of tasks including pre-processing, feature extraction, model fitting, and validation stages. The ML Pipelines is a High-Level API for MLlib that allow the user to execute a sequence of operations, like the ones named before, as a stage of a sequence. This sequence is called *pipeline* and is executed as a whole.

GraphX GraphX is a component in Spark for graphs and graph-parallel computation. It extends RDDs and exposes a set of operators on graphs (aggregations, joins, structural, property, . . .).

Spark Streaming Spark Streaming is an extension of the core Spark API that allows data engineers and data scientists to process real-time data from various sources including Kafka, Flume, ... This processed data can be pushed out to filesystems, databases, and live dashboards. Extending RDDs, Spark Streaming can be used with any other Spark components like MLlib, Spark SQL, . . .

2.2 Transformers

Until massive adoption of transformer technology, the dominant sequence transduction models were based on complex recurrent or convolutional neural networks that include an encoder and a decoder. Transformers are a network architecture based only on a attention mechanism, which can be described as mapping a query and a set of key-value pairs to an output (query, keys, values and output are all vectors) [74], not explicitly exploiting recurrence and convolution, differently from previously adopted system like recurrent neural networks, long short-term memory and gated recurrent neural networks. Transformers are used to draw global dependencies between input and output. They are used to better handle long-range dependencies within a text. These technologies allow to represent texts in a more semantical way, better capturing context and relations among different parts of the text [74]. Transformers are the true state of the art in NLP field, since 2018 (year of BERT [17][45] release) transformers are used in different fields that require a semantic representation of texts. Some examples of Transformer implementations are shortly described here below.

1. **An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale**: in [20], researchers have applied Transformers directly on sub-portions of an image in order to obtain a representation that can be further used for image classification tasks. Good results are shown when training happens on small datasets.

2. **Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition**: in [19], researchers have presented the Speech-Transformer, a no-recurrence sequence-to-sequence model entirely rely on attention mechanisms to learn the positional dependencies, which can be trained faster and with more efficiency. Results based on the Wall Street Journal speech recognition dataset show that the proposed technology achieves an error rate of 10.9%, while the whole training process only takes 1.2 days on 1 GPU, significantly faster than the published results of recurrent sequence-to-sequence models.

2.2.1 Architecture

Transformers, like most competitive neural sequence transduction, have an encoder-decoder structure with the encoder mapping an input sequence of symbols $(x_1, \ldots, x_n)$ to a continuous representation $z = (z_1, \ldots, z_n)$. Given z, the decoder generates an output sequence $(y_1, \ldots, y_m)$ of symbols one element at a time. In the left and right halves of Figure 2.4 is shown the fully-connected layers of both encoders and decoders.

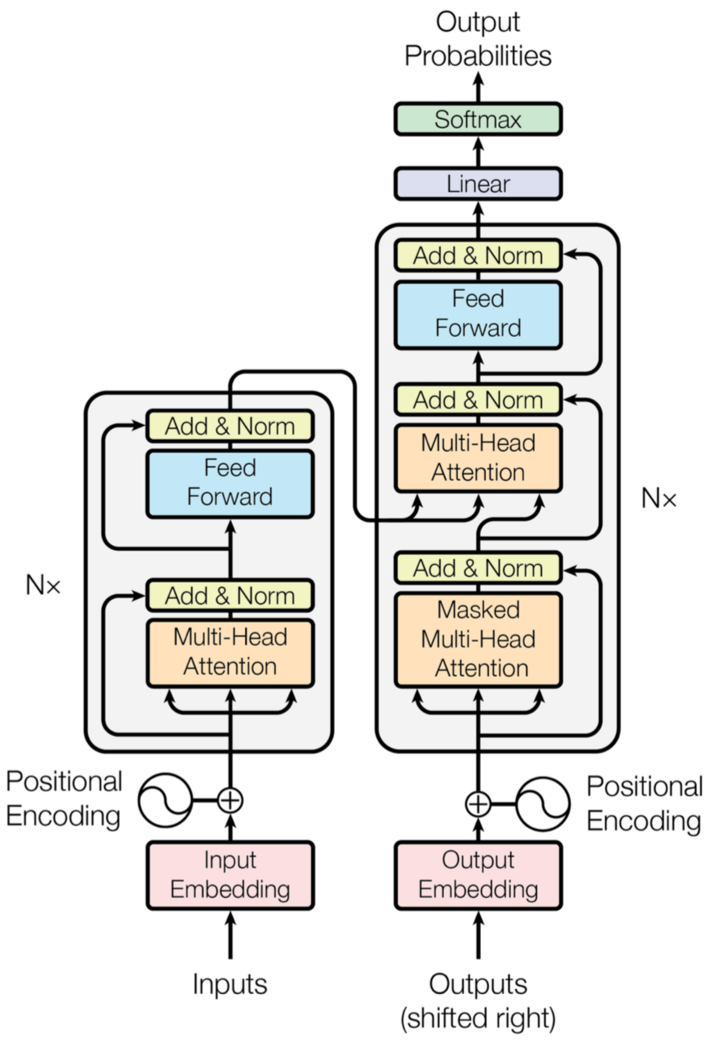

Figure 2.4: Transformer architecture

2.2.2 Encoder

The encoder from Vaswani's research [74] is composed of a stack of $N = 6$ identical layers, each one has two sub-layers:

1. a multi-head self-attention mechanism;
2. a position-wise fully connected feed-forward network.

A residual connection followed by layer normalization is employed around each sub-layer. All sub-layers in the model produce an output of dimension $d_{model} = 512$.

2.2.3 Decoder

The decoder from Vaswani's research [74] is composed of a stack of $N = 6$ identical layers. The decoder inserts a third sub-layer to the two encoder sub-layer. This sub-layer performs multi-head attention over the output of the encoder stack. The idea behind multi-head attention is to allow the attention function to extract information from different representation subspaces, which would, otherwise, not be possible with a single attention head. Similarly to the encoder, around each of the sub-layers are employed residual connections followed by layer normalization. This kind of masking combined the output embeddings, that are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i. All sub-layers in the model produce an output of dimension $d_{model} = 512$.

2.2.4 Attention

An attention function consists in mapping a query and a set of key-value pairs to an output, where the query, keys, values and output are all term vectors. The main purpose of attention is to estimate the relative importance of the keys term compared to the query term related to the same person or concept. To that end, the attention mechanism takes query Q that represents a vector word, the keys K which are all other words in the sentence, and value V represents the vector of the word. The output is computed as a weighted sum of the values, with weights computed by a compatibility function of the query with the corresponding key.

Scaled dot-product attention

The input consists of queries and keys of dimension d_k, and values of dimension d_v. To obtain the weights on the values, the dot product of the query with all keys is performed, dividing each by $\sqrt{d_k}$ and applying a softmax function. The queries, values and keys sets are packed together into Q (queries), K (keys) and V (values) matrices. The attention function is computed using Q, K^T (K transposed) and V matrices, producing an output matrix in this way:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Scaled dot-product attention is both faster and more space efficient of two commonly used attention functions: additive attention and dot-product (multiplicative) attention. These performances reachable because of the highly optimized implementation of matrix multiplication code.

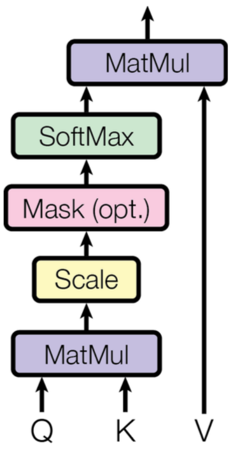

Figure 2.5: Scaled dot-product attention

Multi-head attention

As visible in Figure 2.6, queries, values and keys are linearly projected h times, with different learned linear projections to d_k, d_q and d_v dimensions respectively. The attention function is performed in parallel on each projection, yielding d_v-dimensional output values. These are concatenated and once again projected, resulting in the final values.
Multi-head attention allows the model to jointly participate to information from different representation subspaces at different positions.

$$\text{MultiHead}(Q, V, K) = \text{Concat}(head_1, \ldots, head_h)W^O$$
$$\text{where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices

$$W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_k} \text{ and } W_i^O \in \mathbb{R}^{d_{model} \times d_k}$$

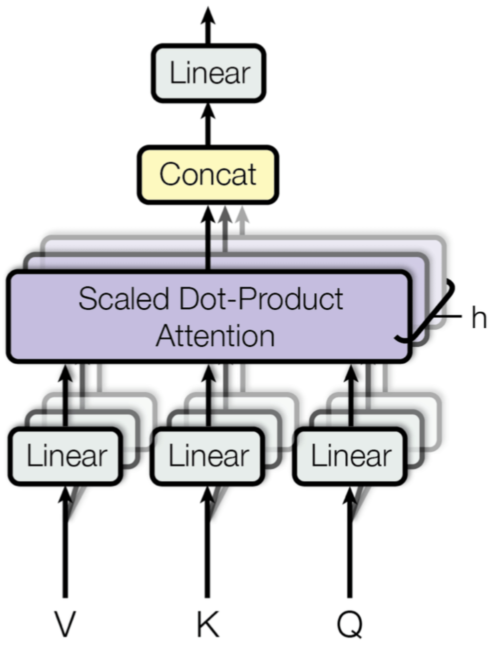

Figure 2.6: Multi-head attention

2.2.5 Position-wise Feed-Forward Networks

Every layer in both encoders and decoders contain a fully connected feed-forward neural network, which is applied to each position separately and identically. It consists of two linear transformations with a ReLU activation in between.

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

The linear transformations remain the same across different positions, using different parameters on each layer. The dimensionality of input and output is $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{\text{ff}} = 2048$.

2.2.6 Embeddings and softmax

Learned embeddings are used to convert the input and output tokens to vectors of dimension d_{model}. The learned linear transformation and softmax function are used to convert the decoder output to predicted next-token probabilities.

2.2.7 Positional encodings

Since the model contains neither recurrence nor convolution, relative or absolute tokens positional encodings are injected in order to make use of the order of the sequence. Positional encodings are added to the input embeddings of the encoder and decoder stacks. They have the same dimension d_{model} as the embeddings, so the two can be summed.

2.2.8 Self-attention motivations

In order to understand why self-attention layers are used, it's necessary to compare them with recurrent and convolutional layers. There are three aspects that are important to consider: the total computation complexity per layer, the amount of computation that can be parallelized (intended as the minimum number of sequential operations required) and the path length between long-range dependencies in the network.
In terms of computation that can be parallelized, a self-attention layer connects all positions with a constant number of sequentially executed operations, whereas a recurrent layer requires $O(n)$ sequential operations. In terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence length n is smaller than the representation dimensionality d, which is most often the case with sentence representations used by state-of-the-art models in machine translations, such as word-piece and byte-pair representations. Table 2.1 shows the complexity comparison between self attention, recurrence and convolutions from the point of view of maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types.

Layer type	Comp. per layer	Seq. ops	Max path length
Self attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$

Table 2.1: Self attention complexity

Chapter 3

Architecture

3.1 Software architecture

The *software architecture* of a system depicts the system's organization or structure, and provides an explanation of how it behaves. A system represents the collection of components that accomplish a specific function or set of functions. In other words, the software architecture provides a sturdy foundation on which software can be built. A series of architecture decisions and trade-offs impact quality, performance, maintainability, and overall success of the system. [50] [67]. There are three different classes of architectural elements:

1. **Processing elements**: elements that supply the transformation on data elements;
2. **Data elements**: elements that contain the information that is used and transformed;
3. **Connecting elements**: elements that connect different architectural elements together.

The architectural form consists of weighted properties and relationships, where the weight indicates either the importance of the property/relationship, or the necessity to select among alternatives, some of which may be preferred over others. The so called *Architect* is able to distinguish between important and decorative formal aspects using properties and relationships weights. These aspects, carefully chosen and designed by the architect, are the key to achieving and reasoning about the system's design goals [77].

3.2 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an architectural style that supports service-orientation. Service-orientation is a way of thinking in terms of services and service-based development and the outcomes of services [63]. The SOA architectural style has the following distinctive features:

1. it is based on the design of the services – which mirror real-world business activities – comprising the enterprise (or inter-enterprise) business processes;
2. service representation implements services using service orchestration;
3. it places unique requirements on the infrastructure;
4. implementations are environment-specific;
5. it requires strong governance of service representation and implementation.

A service:

1. is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports);
2. is self-contained;
3. may be composed of other services;
4. is a "black box" to consumers of the service.

3.3 Project architecture

The implemented architecture follows the idea of SOA. It's based on a series of services that read the input and write the result on the data lake. The final result is a pipeline of services, executed one after the other.

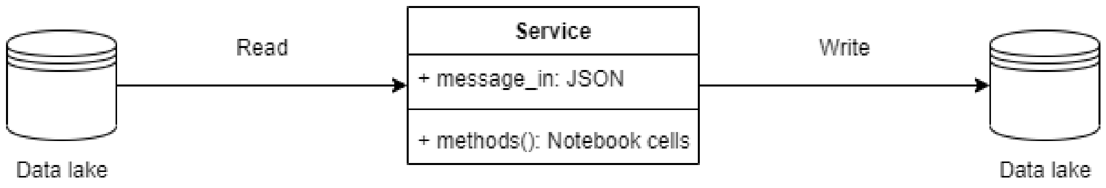

Figure 3.1: Architecture example

Every service executed is a transaction, if there is an error in the execution there's no update in written data. Reading and writing to disk is the most efficient approach, because it avoids to break the memory managing big datasets.

3.3.1 Categories of services

In the implemented framework there are two main categories of services.

1. **Preprocessing services**: preprocessing services are services that prepare the dataset to be used. Those services cover the corrective operations that are applied on the original dataset: textual data are cleaned from special characters, documents are adapted in order to be used further on, documents are filtered given a condition, and so on.
   In this category there are two services: *cleaning* and *filtering* services.

2. **Processing services**: processing services are services that constitutes the core of the framework: in those services, data is processed in order to produce usable embedding representation in order to perform a scoring algorithm to fulfill a semantic similarity search task.
   In this category there are three services: *embedding extraction*, *numerosity reduction* and *scoring* services.

3.3.2 DAG

Once defined the services that need to be executed, these make up a so called Direct Acyclical Graph of execution: it consists of a pipeline where the result of a service execution is the input for the subsequent service. Ideally, the correct form of the DAG is visible in Figure 3.2:

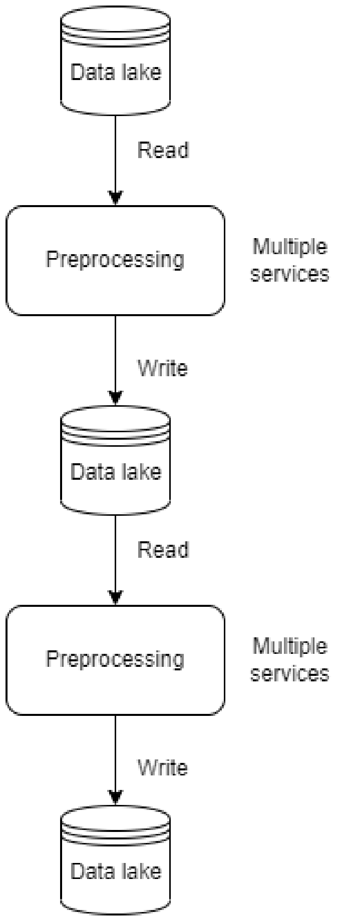

Figure 3.2: DAG example

The two phases (*preprocessing* and *processing*), as visible in Figure 3.2, are subsequent and linked. If anything breaks the pipeline, the dataset will be updated to the last written result, allowing to restart from that checkpoint and not recomputing from the beginning, saving up precious machine-time.

3.3.3 Orchestrator

An orchestrator is a module of a framework which has the task of managing the execution of other modules in order to guarantee the right pipeline steps execution order. The orchestrator reads a series of messages that parametrize different services, than executes the associated services passing read messages as service input. Every message is a correctly formatted JSON object which contains all values that are necessary. An example of a message used as parameter for a service is visible in Listing 3.1.

```
        {
         "service_name": "test_service",
         "parameter_0": "value",
         "parameter_1": "value",
         ...,
         ...,
         "parameter_n": "value,
         "hyperparameters": ["hyperparameter_0",
                             "hyperparameter_1",
                             ...,
                             "hyperparameter_m"]
        }
```

Listing 3.1: "Parameter message example"

Each of the above parameters and hyperparameters are passed to the associated service in order to instantiate different variables that will drive the service execution. The pseudocode for the orchestrator is shown in Algorithm 1.

Algorithm 1: Orchestrator pseudocode

```
Data: List of JSON messages
Result: Execution of services
initialization;
while not at end of message list do
    read message;
    call right service;
    pass current message as input;
```

Chapter 4

Methods

In this chapter is explained the finality of each service. Due to legal reasons in this manuscript there won't be any code or dataset example. Each service component will be explained through a pseudocode representation in order to make the whole system more understandable.
All services are Jupyter notebooks that exploit the parallelization ability of PySpark, working on several rows at the same time using MapReduce approach which is the *de facto* standard in big data field.

4.1 Preprocessing

As said in Chapter 3, this is the phase where the dataset is elaborated in order to be used further on.

4.1.1 Cleaning

Given a textual dataset, cleaning service performs all the operations needed to have cleaned and well formed documents.

HTML cleaning

The first (parametrized) step is to remove all HTML special characters. This operation is preparatory for next phases that need to deal with human-readable texts. The pseudocode of HTML removal phase is visible in Algorithm 2. This service is based on Python HTML class [31], which allows to parse textual files filled with HTML special characters and tags.

```
Algorithm 2: HTML removal
Data: Textual dataset
Result: Cleaned texts
initialization
foreach dataset row do
    read document body
    unescape HTML special tags
    substitute resulting HTML tags with an empty string
```

Sentence splitting

This is the first step to overcome transformer technology limitations: in order to not increase to much memory usage, transformers libraries set a maximum of tokens that the model is able to manage. Therefore, long texts are difficult to manage as a whole, so they need to be divided in sentences (splits). In Algorithm 3 is visible the pseudocode for splitting phase.

```
Algorithm 3: Text split
Data: Cleaned dataset
Result: Splitted texts
initialization
foreach dataset row do
    read document body
    split text according to a regular expression
```

Split cleaning

In order to obtain the best vectorial representation using transformers, splits need to be cleaned from all punctuation and special characters that are not truly useful. Those characters could be present in text due to human errors, OCR errors or even errors in format-to-format conversion. In Algorithm 4 is visible the pseudocode for split cleaning operation.

Algorithm 4: Split cleaning

```
Data: Splitted dataset
Result: Cleaned splits
initialization
foreach dataset row do
    read document splits
    foreach split do
        remove all useless punctuation
        remove all useless characters
        split in words
        if word is all uppercase then
            convert to lowercase
        else
            keep the word
        rebuild the sentence
```

Split filtering

The final step of cleaning service is split filtering. In this phase, splits that do not satisfy certain conditions (splits too short, splits too long, splits with too many repeated words, . . .) are filtered out in order to keep only correct splits. In Algorithm 5 is visible the pseudocode for split filtering operation.

Algorithm 5: Split filter

```
Data: Cleaned dataset
Result: Filtered texts
initialization
foreach dataset row do
    read document splits foreach split do
        if split is longer than threshold AND split length/number of word ratio
          is bigger than threshold then
            keep split
        else
            filter out split
```

4.1.2 Filtering

In some applications, it is needed to reduce the number of records inside a dataset. In the implemented framework this is performed using the filtering service. Reading a list of keys which has to be kept, this service is able to filter out those keys that aren't part of the list mentioned earlier. Algorithm 6 shows how two datasets are joined in order to filter the bigger one. This operation is simply an inner join between two datasets resulting in a smaller dataset due to the filter one.

Algorithm 6: Dataset filter

```
Data: Cleaned dataset
Data: Filter dataset
Result: Filtered dataset
initialization
join cleaned and filter dataset
return filtered dataset with rows that are part of both datasets
```

4.2 Processing

This category includes services that generate the information needed to perform all similarity tasks of the framework. Ideally, these services work on a cleaned dataset, so they are the next step of preprocessing services.

4.2.1 Embedding extraction

This service is the core of the framework: decisions made in this service constitute the biggest discriminant of the framework. Using a given model, it converts sentences into vectorial representations (embedding). This is possible using a library called SentenceTransformers [60], based on their original work *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks* [56]. SentenceTransformers allows to choose a transformer model from either the library itself or from another repository of transformer models: *Huggingface* [32]. Here below is visible the list of Transformer families with a short description that have been used during the project:

1. **BERT**;
2. **ALBERT**;
3. **DistilBERT**;
4. **MPNET**;
5. **XLM**.

BERT BERT has been the first transformer model released after Vaswani's [74]. It has been released and open sourced by Google, reaching the state-of-the-art for NLP. Reading this Google post [45], BERT can represent a word using its surrounding scope, contextualizing the representation. BERT has been the first deeply bidirectional, unsupervised language representation, pretrained using only a plain text corpus [16].

ALBERT Like BERT, ALBERT has been released by Google. Looking at this blog post by Google [1], ALBERT implements two main design changes respecting to BERT, in order to optimize the performances. The first change is to factorize the embedding parametrization, splitting the embedding matrix between input-level embeddings with a relatively-low dimension, while the hidden-layer embeddings use higher dimensionalities. The second aspect on which ALBERT focuses is layer stacking: differently from BERT, in ALBERT there is a parameter sharing across the layers, resulting in a slightly diminished accuracy while reducing the size of the model [37].

MPNET MPNET has been released by Microsoft in 2020. As visible from their blog post [43], researchers that developed MPNET, realized that two of the main pretraining tasks of transformers, masked language modeling adopted in BERT and permuted language modeling adopted in XLNet [79] — another transformer model presented by Google and Carnegie Mellon University researchers from which MPNET inherits —, have different limitations. They came up with and approach that inherits the advantages of Masked language modeling (MLM) and Permuted language modeling (PLM) and avoids their limitations. MLM is proposed in BERT, which randomly masks some tokens with a masked symbol [M] and predicts the masked tokens given remaining tokens. PLM is proposed in XLNet, which randomly permutes a sequence and predicts the tokens in the right part (predicted part) in an autoregressive way. PLM can model the dependency among the predicted tokens with autoregressive prediction, but it cannot see the position information of the full sentence, which will cause mismatches between the pretraining and fine-tuning since the position information of the full sentence can be seen in the downstream tasks. MPNet uses masked and permuted language modeling to model the dependency among predicted tokens and see the position information of the full sentence [68].

DistilBERT DistilBERT has been released by HuggingFace engineers in 2019. They focused on the problem of growing model size and how to put in production such models. From the post on Huggingface blog [66], they used the *distillation* technique to compress a large model into a smaller model.
Distillation: Knowledge distillation (sometimes also referred to as teacher-student learning) is a compression technique in which a small model is trained to reproduce

the behavior of a larger model (or an ensemble of models). It was introduced by Bucila et al. [7] and generalized by Hinton et al. [30] a few years later [57].

XLM XLM transformer has been released by Facebook, in 2019 [36]. As visible Facebook blog post [12], researchers at Facebook developed this transformer model to extend the pretraining for English language to other languages. XLM uses a known pre-processing technique and a dual-language training mechanism with BERT in order to learn relations between words in different languages. First, instead of using word or characters as the input of the model, it uses Byte-Pair Encoding (BPE) that splits the input into the most common sub-words across all languages, thereby increasing the shared vocabulary between languages. Furthermore, it extends BERT architecture in the following two ways.

1. Each training sample consists of the same text in two languages, whereas in BERT each sample is built from a single language. As in BERT, the goal of the model is to predict the masked tokens, however, with the new architecture, the model can use the context from one language to predict tokens in the other, as different words are masked words in each language (they are chosen randomly).
2. The model also receives the language ID and the order of the tokens in each language, i.e. the Positional Encoding, separately. The new metadata helps the model learn the relationship between related tokens in different languages.

SentenceTransformers abstracts all the tokenization and averaging logics: given a sentence, it is tokenized, for each token it is extracted a n-dimensional vector (e.g. $[1, 768]$) and finally those vectors are averaged in order to obtain a single n-dimensional vector. To overcome the problem of transformer model longest manageable sequence, the service performs a tokenization, using the model tokenizer, in order to check if the input sentence is manageable by the chosen model. If it's longer, the sentence is splitted in order to perform the extraction without any problem. The pseudocode is visible in Algorithm 7.

Algorithm 7: Embedding extraction

```
Data: Cleaned dataset
Result: Filtered dataset
initialization
foreach document split do
    if split length > threshold then
        further split
        splits embedding extraction
    else
        split embedding extraction
return embedded dataset
```

4.2.2 Numerosity reduction

Working with long documents ends up in having very long sets of vectors: their dimension remains the same because every extraction is performed with the same model, but their numerosity may vary: a document with 10 sentences has 10 different vectors, and so on. Depending on the model used, these vectors may increase the memory size of the document in a consistent way. To tackle this problem, we proposed and implemented a technique of vector numerosity reduction: in this way it's possible to reduce the memory consumption sacrificing some informative content. The implemented algorithm inside the framework is a service that exploits cosine similarity in order to reduce the vector number: after computing the internal similarity of the cartesian product of all sentences inside a document, the minimum spanning tree is extracted as follows. Every sentence is considered a node in a fully connected graph, while the edges are weighted with the cosine similarity value between the two sentences. The minimum spanning tree defines the path to follow in order to find highly similar sentences: if two sentences are similar over a given threshold, the two vectors are merged reducing the numerosity.
Graph: a graph is a diagram of vertices or nodes connected by edges [65].
Fully connected graph: a fully connected graph is a graph in which every node is connected to every other node [65].
Tree: a tree is a structure consisting of one node called the root and zero or more subtrees [72].

Matrix representation: Given a graph G with vertices labelled 1, 2, ..., n, its *adjacency matrix* A, visible in Figure 4.1, is a $n \times n$ matrix whose ij-th entry is the number of edges joining vertex i and vertex j. If, in addition, the edges are labelled 01, 12, 23,... , nm, its *incidence matrix* M, visible in Figure 4.2, is the $n \times m$ matrix whose ij-th entry is 1 if vertex i is incident to edge j, and 0 otherwise. If the edges have a weight, its *weight matrix* W, visible in Figure 4.3, is the $n \times n$ matrix whose ij-th entry is the weight of the edge connecting vertex i and vertex j.

$$A = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 0 & 1 \\ 3 & 1 & 0 & 1 & 0 \end{array}$$

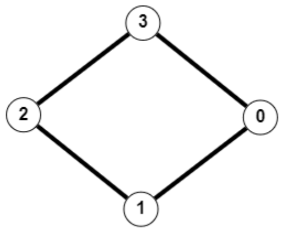

Figure 4.1: Graph corresponding to adjacency matrix A

$$M = \begin{array}{c|cccccc} & 01 & 12 & 23 & 30 & 02 & 13 \\ \hline 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 2 & 0 & 1 & 1 & 0 & 1 & 0 \\ 3 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}$$

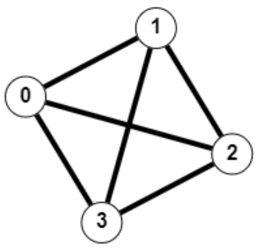

Figure 4.2: Graph corresponding to incidence matrix M

$$W = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 0.2 & 0.60 & 0.40 \\ 1 & 0.20 & 0 & 0.25 & 0.35 \\ 2 & 0.60 & 0.25 & 0 & 0.15 \\ 3 & 0.40 & 0.35 & 0.15 & 0 \end{array}$$

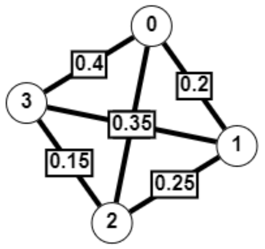

Figure 4.3: Graph corresponding to weight matrix W

Minimum spanning tree: An edge-weighted graph is a graph where we associate weights or costs with each edge. The spanning tree of a graph (G) is a subset of G that covers all of its vertices using the minimum number of edges. A graph can have more than one spanning tree. A minimum spanning tree (MST) [58] of an edge-weighted graph is a spanning tree whose weight (the sum of the weights of its edges) is no larger than the weight of any other spanning tree. It is visible in Figure 4.4 in gray lines. The pseudocode for numerosity reduction algorithm is visible in Algorithm 8.

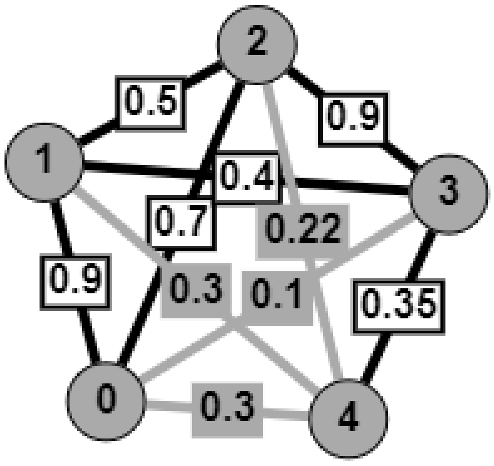

Figure 4.4: MST example

Algorithm 8: Numerosity reduction

```
Data: Embedded dataset
Result: Reduced dataset
initialization
weight matrix calculation
MST computation
foreach edge in MST do
    if edge weight > similarity threshold then
        merge similar sentence embeddings
        recompute weight matrix
        recompute MST
return filtered dataset with rows that are part of both datasets
```

In order to be able to perform this reduction, in this framework it is used *SciPy* [75] package with its features: to compute the weight matrix is used *cosine_ distances* method, while for MST computation is used *minimum_ spanning_ tree* method.

4.2.3 Similarity scoring

In order to complete a similarity search task, a way of scoring this similarity is needed. The generated embeddings as terms of a given similarity function.

Long documents problem

Working with long documents is not trivial, due to intrinsic limitations of transformer models. As said, each model has an input sequence maximum length that cannot be exceeded because it will lead to errors in vectorized representation (sequence truncation, exception raising, . . .). A possible solution to this problem could be averaging all the vectors of different sentences that constitute the original text. The higher the number of sentences, the higher the information dilution: the final vector will be a vector that doesn't truly represent anything inside the original text, leading to high similarity scores even if the original documents are very different. In the next subsection will be presented the chosen similarity scoring approach.

Similarity scoring algorithm

As previously mentioned, working with long documents is not that easy. Starting from the cleaning phase where text bodies are divided in sentences, it has been decided to use a *voting* approach. The voting approach allows to compare documents using the sentences that compose them, allowing in this way to let each sentence "choose" its most similar sentence from other document. The similarity is calculated using cosine similarity, which has been observed to work well in semantic similarity fields. The choice of the most similar sentence is performed keeping the highest scoring sentence in the research document and its score. To give a final result, the algorithm summarizes a resulting score computing one last operation (mean, median, ...) on the list of highest scoring sentences scores chosen by query sentences. The service produces a *result dataset* with rows built like Listing 4.1.

```
        {
         "query_document_key": "query_key",
         "search_document_key": "search_key",
         "score": 0.8
        }
```

Listing 4.1: "Parameter message example"

The pseudocode is visible in Algorithm 9.

Algorithm 9: Scoring algorithm

```
Data: Query document
Data: Search document
Result: Scores dataset
initialization
foreach sentence in query document do
    foreach sentence in search document do
        compute cosine similarity
    choose most similar sentence
compute final score
return score
```

4.3 Evaluation

After computing all necessary scores, in order to understand if the hyperparameters configuration works well it's essential to define an evaluation method. The evaluation method is mandatory in order to have a reliable set of hyperparameters.

4.3.1 Baseline dataset

In order to perform an evaluation, a baseline dataset must be defined. The baseline dataset used throughout the project is constituted by 10 query documents and each one has associated 10 search documents: of these 10 documents, 5 are adherent to the query and 5 are not adherent.

4.3.2 Evaluation method

To understand if the framework is working well, it has been decided to focus on two main aspects:

1. **Classification precision**: the first aspect to consider is if the framework is able to give back the same results that a human would give back to a query search;
2. **Difference between right and wrong**: the second aspect to consider is how differently the system scores right and wrong search documents.

Evaluation metrics

Precision@n In order to understand how precise is the framework, it has been used an evaluation metric called *precision*: recalling the structure of the baseline dataset, where there are 5 correct documents and 5 incorrect documents for each query, the computed scores are ordered in descending order: the number of correct placed documents (first half for adherent documents, second half for non adherent documents) denotes the precision for that given query. Precision denotes the proportion of predicted positive cases that are correctly Real Positives [51].
As the name says, Precision@n is an evaluation metric computed on the n corpus elements: for a given query the resulting measure is *precision@10*, where 10 are the documents associated to the given query. For a given set of queries with two groups of mutually exclusive documents associated (*related*, *not related*), the final result will be the sum of these precisions: for example, for 10 queries there will be a *precision@100* score.

Difference mean/median The other aspect which deserves particular attention is the difference between right and wrong documents. In order to have a reliable framework, it's necessary not only that it assigns right documents to right queries, it would be better if it could be able to strongly differentiate right documents from wrong ones. In order to measure this metric, the system performs the mean of median of similarity scores assigned to right documents in baseline and computes the difference with the mean or median form wrong ones: the higher the difference, the better the result. An high score for these metrics would mean that the system can well recognize right documents, resulting in a more reliable result.

Chapter 5

Experiments and results

5.1 Experimental setup

Due to computation time constraints, it has not been possible to perform experiments on the whole document corpus made available by *artea.com*. Thus, the dataset has been reduced in order to make possible the execution of different experiments in a reasonable time. As mentioned before in Chapter 4, a resulting dataset of 100 records has been created. For each of 10 queries, 5 related and 5 not related documents were searched manually. Each record is a key-value pair object composed like Listing 5.1.

```
        {
         "query_document_key": "query_key",
         "query_field_0": "value",
         ...: ...,
         "query_field_n": "value",
         "search_document_key": "search_key"
         "search_field_0": "value",
         ...: ...,
         "search_field_n": "value",
         "relation": "true"
        }
```

Listing 5.1: "Experimental dataset object"

All the informative content is maintained. At this point, the memory footprint of the dataset is at his peak, because it contains all the previous fields both of the original objects, query and search documents.
As mentioned in Chapter 4, the dataset is composed by one hundred query-search documents coupled: these couples have been decided in a qualitative way, choosing the right documents for each query. For each query there are five documents that are strongly correlated to it and five documents that are not correlated at all. This organization is useful both form the point of view of evaluation (as previously mentioned in Chapter 4), but it is also useful from a computational point of view,

given that the dataset is rather small. An idea of the resulting dataset is visible in Table 5.1, for sake of brevity some information have been omitted.

Query key	Document key	Related
query_key_0	document_key	true
...	...	...
query_key_0	document_key	false
...	...	...
query_key_9	document_key	true
...	...	...
query_key_9	document_key	false

Table 5.1: Resulting dataset

As visible, for each query identified by *query_key_n*, there are multiple documents associated with an attribute called *Related* indicating ndicating whether a document is closely related to the query.

5.1.1 Transformer models

As said in Chapter 4, the model choice is the biggest discriminant for the final result. In Chapter 2 was presented the original transformer architecture based on Vaswani's paper [74]. Such technology has been further enhanced in order to obtain way more usable and representative embeddings as described in Chapter 4.

5.1.2 Libraries and modules

NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more [29].

The *ndarray* is the core of the NumPy package, which encapsulates different n-dimensional arrays of homogeneous data types. There are several important differences between NumPy arrays and the standard Python lists.

1. **Array size**: NumPy array that have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original one.
2. **Element type**: elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory.
3. **Mathematical operations**: NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are performed more efficiently and with less code than using Python's built-in lists.
4. **Advanced libraries**: the growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today's scientific/mathematical Python-based software, just knowing how to use Python's built-in sequence types is insufficient – one also needs to know how to use NumPy arrays.

SciPy

SciPy is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data. With SciPy, an interactive Python session becomes a data-processing and system-prototyping environment rivaling systems, such as MATLAB, IDL, Octave, R-Lab, and SciLab. The additional benefit of basing SciPy on Python is that this also makes a powerful programming language available for use in developing sophisticated programs and specialized applications. Scientific applications using SciPy benefit from the development of additional modules in numerous niches of the software landscape by developers across the world. Everything from parallel programming to Web and data-base subroutines and classes have been made available to the Python programmer. All of this power is available in addition to the mathematical libraries in SciPy [75].

Transformers

Huggingface *Transformers* library provides thousands of pretrained models to perform tasks on texts such as classification, information extraction, question answering, summarization, translation, text generation and more in over 100 languages [78].

scikit-learn

Scikit-learn is an open source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities [47].

NetworkX

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. With NetworkX one can load and store networks in standard and nonstandard data formats, generate many types of random and classic networks, analyze network structure, build network models, design new network algorithms, draw networks, and much more [27].

html

html is a module that defines a class HTMLParser which serves as the basis for parsing text files formatted in HTML (HyperText Mark-up Language) and XHTML. It has been used to clean up all the HTML tags and special characters present inside the body of the textual documents [31].

re

re is a module that module provides regular expression matching operations. Regular expressions are character patterns used to match different portions of a string. It is possible to manage these sub-strings, replacing, deleting, modifying the characters within the,. This module is heavily used for cleaning and splitting operations [55].

SentenceTransformers

SentenceTransformers is a Python framework for state-of-the-art sentence, text and image embeddings. This library abstracts some of the biggest problems of transformers, like sentence length, embedding averaging, even similarity is provided inside the package. In this project, it has been only used the embedding extraction part that takes car of all the concerning aspects [61].

Plotly

Plotly's [33] Python graphing library makes interactive, publication-quality graphs. Built on top of the Plotly JavaScript library (plotly.js), plotly enables Python users to create beautiful interactive Web-based visualizations that can be displayed in Jupyter notebooks, saved to standalone HTML files, or served as part of pure

Python-built Web applications using Dash. Plotly became useful for the realization of some more interactive graphs and charts, like when graphs have been plotted for debug purpose.

5.1.3 Cluster settings

In order to perform all the experiments, we have used the following cluster configuration:

1. **Environment**: Databricks;
2. **Cluster dimension**: 13 nodes cluster;
3. **Programming language**: Python=3.7.10;
4. **Spark version**: PySpark=3.1.2;
5. **Libraries**:
   (a) numpy=1.18.1;
   (b) scikit-learn=0.22.1;
   (c) transformers=4.10.2;
   (d) sentence-transformers=2.0.0;
   (e) html=3.4;
   (f) re=2.2.1;
   (g) scipy=1.7.1;
   (h) networkx=2.6.3;
   (i) plotly=5.3.1;

This setting allowed to execute an entire experiment, from dataset cleaning to evaluation, in a time between one and three hours, depending on the transformer model.

5.2 Results

Using a different combination of hyperparameters for each execution led to a series of different experiments. Each experiment consisted in a combination of: transformer models, numerosity reduction similarity thresholds and voting techniques.

Algorithms working with previously mentioned hyperparameters have been explained in Chapter 4.
In Table 5.2 is visible the association between model label, model name and result entry in Tables 5.5 and 5.3. For readability reasons, each model name has been converted to a shorter and more readable label.

Label	Model	Table entry
STSBMpnetBaseV2	stsb-mpnet-base-v2	Mpnet
ParaphraseAlbertBaseV2	paraphrase-albert-base-v2	PAlbert
ParaphraseXlmRMultilingualV1	paraphrase-xlm-r-multilingual-v1	Xlm
AlbertBaseV2	albert-base-v2	Albert
DistilbertBaseMultilingualCased	distilbert-base-multilingual-cased	Distilbert

Table 5.2: Model name - Model label - Table entry correspondence

5.2.1 Result discussion

In this subsection there is a discussion about some important aspects of the results that have emerged by our experiments.
The first important aspect that emerges from performed experiments is that models trained using two specific datasets achieve better results overall than other models. As visible from Table 5.5, using models trained using a set of sentence pairs with a similarity score, like *stsb-mpnet-base-v2* (Mpnet), and models trained using paraphrase datasets, like *paraphrase-albert-base-v2* (PAlbert), show better result overall than other generic models and are visible in Table 5.3. Those models show a very good result for *Precision@100*, which measures the percentage of right positioned documents in relation to the input query, meaning that the framework is able to assign right documents to right queries, but it's interesting to notice also that they can separate quite well right documents form wrong ones, as visible looking at *Difference mean*, which measures the difference between the mean of correlated documents and not related documents, and *Difference median* metrics, which measures the difference between the median similarity score of correlated documents and not related documents. Median proves to be a better voting technique then mean due to its greater robustness to outliers.

Model	P@100	Diff mean	Diff median	Red ratio %	Voting tech
Mpnet	0,88	0,13	0,12	9,88	median
PAlbert	0,86	0,1	0,11	6,52	median
Mpnet	0,86	0,12	0,12	11,49	median
PAlbert	0,84	0,1	0,1	2,63	median
PAlbert	0,84	0,09	0,1	10,87	mean
Mpnet	0,84	0,12	0,12	15,72	mean

Table 5.3: Best results table

Secondly, the implemented numerosity reduction shows a result conservation useful if cluster usage is a concern: reducing space occupation and maintaining similar result is considerable as an achievement. The first line of Table 5.5, which is the best result achieved, shows how a 10% reduction of the number of sentences in the corpus doesn't influence the final result while using a reduction threshold of 0.4. Recalling Chapter 4, *Red %* visible in Tables 5.3 5.4 and 5.5 is the threshold used in the numerosity reduction phase. Table 5.4 highlights this phenomena and shows that these models, in the scope of the project, don't face a result degradation when the number of sentences is reduced. When dealing with big data this could become useful to reduce disk and memory occupation, resulting in shorter read and write times. Moreover, working on a smaller dataset tend to reduce computational time, which may be very costly depending on the cluster setting. Obviously, this approach works well on a dataset built like the one used during the project, it needs further tests to see how it behaves on differently built datasets.

Model	P@100	Diff mean	Diff median	Red ratio %	Voting tech	Red %
Mpnet	0,88	0,13	0,12	9,88	median	0,4
Mpnet	0,88	0,13	0,13	0	median	0
PAlbert	0,86	0,1	0,11	6,52	median	0,4
PAlbert	0,86	0,1	0,11	0	median	0

Table 5.4: Result conservation

Thirdly, Table 5.5 show that for the project scope is generally better to use a monolingual model (in our case English) instead of using a multilingual one. This happens because retraining a model like BERT, ALBERT, . . . is very demanding and usually is performed by big companies (Google, Facebook, . . .) or universities (University of Darmstadt, University of Massachusetts, . . .) leading to the majority of models being specialized on English. To emphasize this, going on Huggingface website [42] is visible that for the task of semantic similarity there are

383 models while just 1 Italian, 9 French, 3 German, ...
Fourthly, Table 5.5 show that for the project scope is generally better to use median as voting technique (Chapter 4) in order to compute the final result, as it output the best results. This happens due to the median higher robustness to outliers when compared to the mean: the reason for this is that the mean can be dragged up or down by extreme values, but since the median is just the middle value in a distribution, it is not influenced by the outliers.
Lastly, not furtherly specialized models like *albert-base-v2* (AlbertBaseV2) and *distilbert-base-multilingual-cased* (DistilbertBaseMultilingualCased) show worse results than others. In the project scope, is demonstrated that using those models doesn't reach good enough performances, emphasizing even more the concept that the model choice is crucial for a system like the one presented in this document and that the choice operation needs a big attention in order to make the best choice.

Table 5.5 gathers the whole set of experimental result.

Model	P@100	Diff mean	Diff median	Multilang	Red ratio %	Voting tech	Red %
Mpnet	0,88	0,13	0,12	false	9,88	median	0,4
Mpnet	0,88	0,13	0,13	false	0	median	0,1
Mpnet	0,88	0,13	0,13	false	0	median	0,2
Mpnet	0,88	0,13	0,13	false	0	median	0,3
Mpnet	0,88	0,13	0,13	false	0	median	0
PAlbert	0,86	0,1	0,11	false	0	median	0,2
PAlbert	0,86	0,1	0,11	false	6,52	median	0,4
PAlbert	0,86	0,1	0,11	false	0	median	0
Xlm	0,86	0,12	0,1	true	0	median	0
PAlbert	0,86	0,1	0,11	false	0,74	median	0,3
PAlbert	0,86	0,1	0,11	false	0	median	0,1
Mpnet	0,86	0,12	0,12	false	11,49	median	0,5
PAlbert	0,84	0,1	0,1	false	2,63	median	0,5
Xlm	0,84	0,11	0,09	true	3,64	median	0,5
Mpnet	0,84	0,12	0,12	false	0	mean	0,2
Mpnet	0,84	0,12	0,12	false	0	mean	0,1
PAlbert	0,84	0,09	0,1	false	10,87	mean	0,5
Mpnet	0,84	0,12	0,12	false	8,62	mean	0,4
Mpnet	0,84	0,12	0,12	false	6,4	mean	0,3
Mpnet	0,84	0,12	0,12	false	15,72	mean	0,5
Mpnet	0,84	0,12	0,12	false	0	mean	0
PAlbert	0,82	0,1	0,11	false	0	mean	0,2
PAlbert	0,82	0,1	0,11	false	0	mean	0,1
PAlbert	0,82	0,1	0,11	false	5,56	mean	0,3
PAlbert	0,82	0,1	0,11	false	0	mean	0
PAlbert	0,82	0,1	0,11	false	2,67	mean	0,4
Xlm	0,8	0,1	0,1	true	1,69	mean	0,5
Xlm	0,8	0,11	0,11	true	0	mean	0
Albert	0,74	0,02	0,02	false	0	median	0
Distilbert	0,74	0,04	0,05	true	0	median	0

Table 5.5: Experimental results

Model	P@100	Diff mean	Diff median	Multilang	Red ratio %	Voting tech	Red %
Albert	0,72	0,02	0,02	false	0	mean	0
Distilbert	0,72	0,04	0,04	true	0	mean	0
Distilbert	0,68	0,02	0,02	true	79,73	median	0,1
Distilbert	0,64	0,01	0,03	true	93,75	mean	0,4
Distilbert	0,64	0,01	0,03	true	96,52	median	0,4
Distilbert	0,62	0,01	0,02	true	67,21	mean	0,1
Distilbert	0,58	0	0,03	true	97,47	median	0,5
Distilbert	0,58	0	0,03	true	97,22	mean	0,5
Albert	0,54	0	0,02	false	96,67	mean	0,1
Albert	0,54	0	0,02	false	96,15	median	0,1
Distilbert	0,54	0,01	0	true	84,48	mean	0,2
Distilbert	0,5	0	0	true	87,5	median	0,3
Distilbert	0,5	0,01	0	true	71,88	median	0,2
Distilbert	0,5	0	0	true	94,74	mean	0,3
Albert	0,48	0	0	false	96,08	mean	0,5
Albert	0,48	0	0	false	94,44	mean	0,3
Albert	0,48	0	0	false	95	median	0,5
Albert	0,48	0	0	false	94,44	median	0,3
Albert	0,48	0	0	false	95,56	mean	0,2
Albert	0,48	0	0	false	96,08	mean	0,4
Albert	0,48	0	0	false	97,37	median	0,2
Albert	0,48	0	0	false	96,08	median	0,4

Table 5.6: Continued from previous table

Chapter 6

Conclusions

6.1 Project conclusions

Wrapping up, semantic search task relies heavily on the concept of semantic similarity. When thinking outside computer science, the concept of semantic similarity is about finding a common meaning between two sentences, documents, . . . , while possibly having different words and expression within. This research aimed to explore the opportunities to exploit a semantic technology like transformers in order to enhance semantic search in a scope, like the project one, where compared textual data are different from each other. The Proof of Concept (PoC) presented in this document consists of a "Semantic Search Engine" able to semantically rank documents given a query at search time based entirely on Transformer technology, able to capture the context and the relations inside given documents.
It has been demonstrated throughout the project development that an approach like the one described in this document could be helpful in context where there is a big quantity of documents and a human has to manually search within this corpus. Using this framework is possible to retrieve the most pertinent result(s) for a given query.
An important aspect that emerges from the document is that the model choice is absolutely crucial: the more specialized the model is for the task, the better the results obtainable. As visible in Chapter 5, using generic models such as *distilbert-base-multilingual-cased* or *albert-base-v2* led to inconsistent results, while using a model like *stsb-mpnet-base-v2* which is designed to work well in similarity tasks return better results.
In conclusion, during the course of this thesis, satisfactory results were achieved and has been developed a software solution able to take care of all aspects pertaining to NLP, from data extraction to result evaluation. This piece of software provides, even potentially to not expert, a solution to the problem of semantic similarity in big data and is composed of:

1. a preprocessing engine able to clean and reorganize the content of a document to be more digestible for the processing engine;

2. a processing engine able to take well formed documents and rank them in order to provide the best result for a given input query;
3. an evaluation engine able to estimate quantitatively how good are the choices that the user made in terms of hyperparameters.

The PoC leverages the most cutting edge technologies like Transformers, Spark and HDFS in order to deliver a solution able to be up-to-date with the evolution of the subject: the framework is able to read different kind of documents even if not well formed and it is able to scale thanks to the distributed computing technology adopted not needing any code modification.

6.2 Future work

As said in Chapter 4, due to time constraints, it hasn't been possible to test the system on a bigger portion of the dataset so a possible further improvement could be given by the application of the framework on a different and bigger portion of the dataset.
As said in Chapter 1, one of the main challenges has been dealing with long documents: an improvement to this work could be testing a different kind of document vectorization, perhaps adding a previous information retrieval layer able to extract relevant text portions and then vectorize only these parts, reducing computational and memory load. Speaking about the algorithmic part of the project, every further possible enhancement could constitute a future work. Supervised approaches and different similarity computation methods could bring benefit to the final result.
Another future work, should be testing the framework on a dataset coming from a completely different domain. As previously stated, the system is able to work with every kind of textual document from a technical point of view, but speaking about performances it is not as easy as it seems: the best model for the current dataset could be the worst for another dataset, the cleaning approach applied could be harmful when applied on another dataset and so on.
Wrapping the framework inside an application architecture to allow the runtime computation of needed similarities will be another important future work.
Lastly, an interesting contribution could be specializing a chosen model for the current dataset in order to theoretically obtain even better result than the starting model.

Bibliography

[1] *ALBERT: A Lite BERT for Self-Supervised Learning of Language Representations.* https://ai.googleblog.com/2019/12/albert-lite-bert-for-self-supervised.html.

[2] *artea.com.* https://artea.com/.

[3] Iz Beltagy, Kyle Lo, and Arman Cohan. "SciBERT: A Pretrained Language Model for Scientific Text". In: *EMNLP*. 2019.

[4] *BERT, ELMo, & GPT-2: How Contextual are Contextualized Word Representations?* http://ai.stanford.edu/blog/contextual/.

[5] *Big Data.* https://www.gartner.com/en/information-technology/glossary/big-data.

[6] *Browse State-of-the-Art.* https://paperswithcode.com/sota.

[7] Cristian Buciluundefined, Rich Caruana, and Alexandru Niculescu-Mizil. "Model Compression". In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* KDD '06. Philadelphia, PA, USA: Association for Computing Machinery, 2006, pp. 535–541. ISBN: 1595933395. DOI: 10.1145/1150402.1150464. URL: https://doi.org/10.1145/1150402.1150464.

[8] Aylin Caliskan, J. Bryson, and A. Narayanan. "Semantics derived automatically from language corpora contain human-like biases". In: *Science* 356 (2017), pp. 183–186.

[9] Daniel Matthew Cer et al. "SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation". In: *SemEval@ACL.* 2017.

[10] *Community Question Answering.* https://paperswithcode.com/task/community-question-answering.

[11] James C. Corbett et al. "Spanner: Google's Globally Distributed Database". In: *ACM Trans. Comput. Syst.* 31.3 (Aug. 2013). ISSN: 0734-2071. DOI: 10.1145/2491245. URL: https://doi.org/10.1145/2491245.

[12] *Cross-lingual pretraining sets new state of the art for natural language understanding.* https://ai.facebook.com/blog/cross-lingual-pretraining/.

[13] *Dataframe.* `https://spark.apache.org/docs/latest/sql-programming-guide.html`.

[14] *Dataset.* `https://databricks.com/it/spark/getting-started-with-apache-spark/quick-start`.

[15] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *OSDI.* 2004.

[16] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: `1810.04805`. URL: `http://arxiv.org/abs/1810.04805`.

[17] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[18] Nova Eka Diana and Ikrima Hanana Ulfa. "Measuring Performance of N-Gram and Jaccard-Similarity Metrics in Document Plagiarism Application". In: *Journal of Physics: Conference Series* 1196 (Mar. 2019), p. 012069. DOI: `10.1088/1742-6596/1196/1/012069`. URL: `https://doi.org/10.1088/1742-6596/1196/1/012069`.

[19] Linhao Dong, Shuang Xu, and Bo Xu. "Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2018), pp. 5884–5888.

[20] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *ArXiv* abs/2010.11929 (2021).

[21] A. Gandomi and Murtaza Haider. "Beyond the hype: Big data concepts, methods, and analytics". In: *Int. J. Inf. Manag.* 35 (2015), pp. 137–144.

[22] GhemawatSanjay, GobioffHoward, and LeungShun-Tak. "The Google file system". In: *Operating Systems Review* (2003).

[23] W. H. Gomaa and A. Fahmy. "A Survey of Text Similarity Approaches". In: *International Journal of Computer Applications* 68 (2013), pp. 13–18.

[24] *Hadoop.* `https://cwiki.apache.org/confluence/display/hadoop`.

[25] *Hadoop – Architecture.* `https://www.geeksforgeeks.org/hadoop-architecture/`.

[26] *Hadoop | History or Evolution.* `https://www.geeksforgeeks.org/hadoop-history-or-evolution/`.

[27] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference.* Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15.

[28] Jiawei Han, Micheline Kamber, and Jian Pei. "2 - Getting to Know Your Data". In: *Data Mining (Third Edition).* Ed. by Jiawei Han, Micheline Kamber, and Jian Pei. Third Edition. The Morgan Kaufmann Series in Data Management Systems. Boston: Morgan Kaufmann, 2012, pp. 39–82. ISBN:

978-0-12-381479-1. DOI: https://doi.org/10.1016/B978-0-12-381479-1.00002-2. URL: https://www.sciencedirect.com/science/article/pii/B9780123814791000022.

[29] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[30] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML].

[31] *html.parser — Simple HTML and XHTML parser*. https://docs.python.org/3/library/html.parser.html.

[32] *Hugging Face*. https://huggingface.co/.

[33] Plotly Technologies Inc. *Collaborative data science*. 2015. URL: https://plot.ly.

[34] Sergio Jiménez, C. Becerra, and Alexander Gelbukh. “SOFTCARDINALITY-CORE: Improving Text Overlap with Distributional Measures for Semantic Textual Similarity”. In: **SEMEVAL*. 2013.

[35] Armand Joulin et al. “Bag of Tricks for Efficient Text Classification”. In: *EACL*. 2017.

[36] Guillaume Lample and Alexis Conneau. “Cross-lingual Language Model Pretraining”. In: *CoRR* abs/1901.07291 (2019). arXiv: 1901.07291. URL: http://arxiv.org/abs/1901.07291.

[37] Zhenzhong Lan et al. “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations”. In: *CoRR* abs/1909.11942 (2019). arXiv: 1909.11942. URL: http://arxiv.org/abs/1909.11942.

[38] Md Tahmid Rahman Laskar, Xiangji Huang, and Enamul Hoque. “Contextualized Embeddings based Transformer Encoder for Sentence Similarity Modeling in Answer Selection Task”. In: *LREC*. 2020.

[39] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. 3rd ed. Cambridge University Press, 2020. DOI: 10.1017/9781108684163.

[40] E. Lydia, P.Govindaswamy, and D. Ramya. “Document Clustering Based On Text Mining K-Means Algorithm Using Euclidean Distance Similarity”. In: 2018.

[41] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[42] *Models*. https://huggingface.co/models.

[43] *MPNet combines strengths of masked and permuted language modeling for language understanding*. https://www.microsoft.com/en-us/research/blog/mpnet-combines-strengths-of-masked-and-permuted-language-modeling-for-language-understanding/.

[44] David Nettleton. “Chapter 11 - Text Analysis”. In: *Commercial Data Mining.* Ed. by David Nettleton. Boston: Morgan Kaufmann, 2014, pp. 171–179. ISBN: 978-0-12-416602-8. DOI: https://doi.org/10.1016/B978-0-12-416602-8.00011-X. URL: https://www.sciencedirect.com/science/article/pii/B978012416602800011X.

[45] *Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing.* https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html.

[46] J. H. Paik. “A novel TF-IDF weighting scheme for effective ranking”. In: *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval* (2013).

[47] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[48] Nicole Peinelt, D. Nguyen, and Maria Liakata. “tBERT: Topic Models and BERT Joining Forces for Semantic Similarity Detection”. In: *ACL.* 2020.

[49] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP).* 2014, pp. 1532–1543.

[50] D. Perry and A. Wolf. “Foundations for the study of software architecture”. In: *ACM Sigsoft Software Engineering Notes* 17 (1992), pp. 40–52.

[51] David M. W. Powers. “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation”. In: *CoRR* abs/2010.16061 (2020). arXiv: 2010.16061. URL: https://arxiv.org/abs/2010.16061.

[52] Shahzad Qaiser and R. Ali. “Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents”. In: *International Journal of Computer Applications* 181 (2018), pp. 25–29.

[53] *Question Answering.* https://paperswithcode.com/task/question-answering.

[54] *RDD.* https://spark.apache.org/docs/latest/rdd-programming-guide.html#resilient-distributed-datasets-rdds.

[55] *re.* https://docs.python.org/3/library/re.html.

[56] Nils Reimers and Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.* 2019. arXiv: 1908.10084 [cs.CL].

[57] Victor Sanh et al. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *CoRR* abs/1910.01108 (2019). arXiv: 1910.01108. URL: http://arxiv.org/abs/1910.01108.

[58] Robert Sedgewick and Kevin Wayne. *Algorithms, 4th Edition.* Addison-Wesley, 2011, pp. I–XII, 1–955. ISBN: 978-0-321-57351-3.

[59] *Semantic Textual Similarity.* https://paperswithcode.com/task/semantic-textual-similarity.

[60] *SentenceTransformers.* https://www.sbert.net/.

[61] *SentenceTransformers.* URL: https://www.sbert.net/index.html.

[62] *Sentiment Analysis.* https://paperswithcode.com/task/sentiment-analysis.

[63] *Service-Oriented Architecture.* https://www.opengroup.org/soa/source-book/soa/p1.htm#soa_definition.

[64] K. Shvachko et al. "The Hadoop Distributed File System". In: *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (2010), pp. 1–10.

[65] M. Siu. "Introduction to graph theory (4th edition), by Robin J. Wilson. Pp. 171. £14.99. 1996. ISBN : 0-582-24993-7 (Longman)." In: *The Mathematical Gazette* 82 (1998), pp. 343–344.

[66] *Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT.* https://medium.com/huggingface/distilbert-8cf3380435b5.

[67] *Software Architecture & Software Security Design.* https://www.synopsys.com/glossary/what-is-software-architecture.html.

[68] Kaitao Song et al. "MPNet: Masked and Permuted Pre-training for Language Understanding". In: *CoRR* abs/2004.09297 (2020). arXiv: 2004.09297. URL: https://arxiv.org/abs/2004.09297.

[69] Wanpeng Song. "Sentence Similarity Calculating Method Based on Word2Vec and Clustering". In: *DEStech Transactions on Engineering and Technology Research* (2020).

[70] *Spark.* https://spark.apache.org/docs/latest/.

[71] *Text Classification.* https://paperswithcode.com/task/text-classification.

[72] *Tree Data Structure.* https://www.cs.cmu.edu/~clo/www/CMU/DataStructures/Lessons/lesson4_1.htm.

[73] B. Trstenjak, Sasa Mikac, and D. Donko. "KNN with TF-IDF based Framework for Text Categorization". In: *Procedia Engineering* 69 (2014), pp. 1356–1364.

[74] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems.* 2017, pp. 5998–6008.

[75] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[76] Min Wang et al. "Microblog Sentiment Analysis Based on Cross-media Bag-of-words Model". In: *ICIMCS '14.* 2014.

[77] *What Is Your Definition of Software Architecture.* https://resources.sei.cmu.edu/asset_files/factsheet/2010_010_001_513810.pdf.

[78] Thomas Wolf et al. *Transformers: State-of-the-Art Natural Language Processing.* Oct. 2020. URL: `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

[79] Zhilin Yang et al. “XLNet: Generalized Autoregressive Pretraining for Language Understanding”. In: *NeurIPS*. 2019.

[80] Matei Zaharia et al. “Spark: Cluster computing with working sets.” In: *HotCloud* 10.10-10 (2010), p. 95.

[81] Yongjun Zhu, Erjia Yan, and Fei Wang. “Semantic relatedness and similarity of biomedical terms: examining the effects of recency, size, and section of biomedical publications on the performance of word2vec”. In: *BMC Medical Informatics and Decision Making* 17 (2017).