I'm not a bot

Open-source data analytics cluster computing frameworkApache SparkOriginal author(s)Matei ZahariaDeveloper(s)Apache SparkInitial releaseMay26, 2014; 10 years ago(2014-05-26)Stable release3.5.4 (Scala 2.13) / December20, 2024; 5 months ago(2024-12-20)RepositorySpark RepositoryWritten inScala[1]Operating systemMicrosoft Windows, macOS, LinuxAvailable inScala, Java, SQL, Python, R, C#, F#TypeData analytics, machine learning algorithmsLicenseApache License 2.0Websitespark.apache.org Apache Spark is an open-source unified analytics engine for large-scale data processing. Spark provides an interface for programming clusters with implicit data parallelism and fault tolerance. Originally developed at the University of California, Berkeley's AMPLab starting in 2009, in 2013, the Spark codebase was donated to the Apache Software Foundation, which has maintained it since.Apache Spark has its architectural foundation in the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way.[2] The Dataframe API was released as an abstraction on top of the RDD, followed by the Dataset API. In Spark 1.x, the RDD was the primary application programming interface (API), but as of Spark 2.x use of the Dataset API is encouraged[3] even though the RDD API is not deprecated.[4][5] The RDD technology still underlies the Dataset API.[6][7]Spark and its RDDs were developed in 2012 in response to limitations in the MapReduce cluster computing paradigm, which forces a particular linear dataflow structure on distributed programs: MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk. Spark's RDDs function as a working set for distributed programs that offers a (deliberately) restricted form of distributed shared memory.[8]Inside Apache Spark the workflow is managed as a directed acyclic graph (DAG). Nodes represent RDDs while edges represent the operations on the RDDs.Spark facilitates the implementation of both iterative algorithms, which visit their data set multiple times in a loop, and interactive/exploratory data analysis, i.e., the repeated database-style querying of data. The latency of such applications may be reduced by several orders of magnitude compared to Apache Hadoop MapReduce implementation.[2][9]Among the class of iterative algorithms are the training algorithms for machine learning systems, which formed the initial impetus for developing Apache Spark.[10]Apache Spark requires a cluster manager and a distributed storage system. For cluster management, Spark supports standalone native Spark, Hadoop YARN, Apache Mesos or Kubernetes.[11] A standalone native Spark cluster can be launched manually or by the launch scripts provided by the install package. It is also possible to run the daemons on a single machine for testing. For distributed storage Spark can interface with a wide variety of distributed systems, including Alluxio, Hadoop Distributed File System (HDFS),[12] MapR File System (MapR-FS),[13] Cassandra,[14] OpenStack Swift, Amazon S3, Kudu, Lustre file system,[15] or a custom solution can be implemented. Spark also supports a pseudo-distributed local mode, usually used only for development or testing purposes, where distributed storage is not required and the local file system can be used instead; in such a scenario, Spark is run on a single machine with one executor per CPU core.Spark Core is the foundation of the overall project. It provides distributed task dispatching, scheduling, and basic I/O functionalities, exposed through an application programming interface (for Java, Python, Scala, .NET[16] and R) centered on the RDD abstraction (the Java API is available for other JVM languages, but is also usable for some other non-JVM languages that can connect to the JVM, such as Julia[17]). This interface mirrors a functional/higher-order model of programming: a "driver" program invokes parallel operations such as map, filter or reduce on an RDD by passing a function to Spark, which then schedules the function's execution in parallel on the cluster.[2] These operations, and additional ones such as joins, take RDDs as input and produce new RDDs. RDDs are immutable and their operations are lazy; fault-tolerance is achieved by keeping track of the "lineage" of each RDD (the sequence of operations that produced it) so that it can be reconstructed in the case of data loss. RDDs can contain any type of Python, .NET, Java, or Scala objects.Besides the RDD-oriented functional style of programming, Spark provides two restricted forms of shared variables: broadcast variables reference read-only data that needs to be available on all nodes, while accumulators can be used to program reductions in an imperative style.[2]A typical example of RDD-centric functional programming is the following Scala program that computes the frequencies of all words occurring in a set of text files and prints the most common ones. Each map, flatMap (a variant of map) and reduceByKey takes an anonymous function that performs a simple operation on a single data item (or a pair of items), and applies its argument to transform an RDD into a new RDD.val conf = new SparkConf().setAppName("wiki_test") // create a spark config objectval sc = new SparkContext(conf) // Create a spark contextval data = sc.textFile("/path/to/somedir") // Read files from "somedir" into an RDD of (filename, content) pairs.val tokens = data.flatMap(_.split(" ")) // Split each file into a list of tokens (words).val wordFreq = tokens.map((_, 1)).reduceByKey(_ + _) // Add a count of one to each token, then sum the counts per word type.wordFreq.sortBy(s => -s._2).map(x => (x._2, x._1)).top(10) // Get the top 10 words. Swap word and count to sort by count.Spark SQL is a component on top of Spark Core that introduced a data abstraction called DataFrames,[a] which provides support for structured and semi-structured data. Spark SQL provides a domain-specific language (DSL) to manipulate DataFrames in Scala, Java, Python or .NET.[16] It also provides SQL language support, with command-line interfaces and ODBC/JDBC server. Although DataFrames lack the compile-time type-checking afforded by RDDs, as of Spark 2.0, the strongly typed DataSet is fully supported by Spark SQL as well.import org.apache.spark.sql.SparkSessionval url = "jdbc:mysql://yourIP:yourPort/test?user=yourUsername;password=yourPassword" // URL for your database server.val spark = SparkSession.builder().getOrCreate() // Create a Spark session objectval df = spark .read .format("jdbc") .option("url", url) .option("dbtable", "people") .load()df.printSchema() // Looks at the schema of this DataFrame.val countsByAge = df.groupBy("age").count() // Counts people by ageOr alternatively via SQL:df.createOrReplaceTempView("people")val countsByAge = spark.sql("SELECT age, count(*) FROM people GROUP BY age")Spark Streaming uses Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD transformations on those mini-batches of data. This design enables the same set of application code written for batch analytics to be used in streaming analytics, thus facilitating easy implementation of lambda architecture.[19][20] However, this convenience comes with the penalty of latency equal to the mini-batch duration. Other streaming data engines that process event by event rather than in mini-batches include Storm and the streaming component of Flink.[21] Spark Streaming has support built-in to consume from Kafka, Flume, Twitter, ZeroMQ, Kinesis, and TCP/IP sockets.[22]In Spark 2.x, a separate technology based on Datasets, called Structured Streaming, that has a higher-level interface is also provided to support streaming.[23]Spark can be deployed in a traditional on-premises data center as well as in the cloud.[24]Spark MLlib is a distributed machine-learning framework on top of Spark Core that, due in large part to the distributed memory-based Spark architecture, is as much as nine times as fast as the disk-based implementation used by Apache Mahout (according to benchmarks done by the MLlib developers against the alternating least squares (ALS) implementations, and before Mahout itself gained a Spark interface), and scales better than Vowpal Wabbit.[25] Many common machine learning and statistical algorithms have been implemented and are shipped with MLlib which simplifies large scale machine learning pipelines, including:summary statistics, correlations, stratified sampling, hypothesis testing, random data generation[26]classification and regression: support vector machines, logistic regression, linear regression, naive Bayes classification, Decision Tree, Random Forest, Gradient-Boosted Treecollaborative filtering techniques including alternating least squares (ALS)cluster analysis methods including k-means, and latent Dirichlet allocation (LDA)dimensionality reduction techniques such as singular value decomposition (SVD), and principal component analysis (PCA)feature extraction and transformation functionsoptimization algorithms such as stochastic gradient descent, limited-memory BFGS (L-BFGS)GraphX is a distributed graph-processing framework on top of Apache Spark. Because it is based on RDDs, which are immutable, graphs are immutable and thus GraphX is unsuitable for graphs that need to be updated, let alone in a transactional manner like a graph database.[27] GraphX provides two separate APIs for implementation of massively parallel algorithms (such as PageRank): a Pregel abstraction, and a more general MapReduce-style API.[28] Unlike its predecessor Bagel, which was formally deprecated in Spark 1.6, GraphX has full support for property graphs (graphs where properties can be attached to edges and vertices).[29]Like Apache Spark, GraphX initially started as a research project at UC Berkeley's AMPLab and Databricks, and was later donated to the Apache Software Foundation and the Spark project.[30]Apache Spark has built-in support for Scala, Java, SQL, R, and Python with 3rd party support for the .NET CLR,[31] Julia,[32] and more.Spark was initially started by Matei Zaharia at UC Berkeley's AMPLab in 2009, and open sourced in 2010 under a BSD license.[33]In 2013, the project was donated to the Apache Software Foundation and switched its license to Apache 2.0. In February 2014, Spark became a Top-Level Apache Project.[34]In November 2014, Spark founder M. Zaharia's company Databricks set a new world record in large scale sorting using Spark.[35][33]Spark had in excess of 1000 contributors in 2015,[36] making it one of the most active projects in the Apache Software Foundation[37] and one of the most active open source big data projects.VersionOriginal release dateLatest versionRelease dateOld version, not maintained: 0.52012-06-120.5.22012-11-22Old version, not maintained: 0.62012-10-150.6.22013-02-07Old version, not maintained: 0.72013-02-270.7.32013-07-16Old version, not maintained: 0.82013-09-250.8.12013-12-19Old version, not maintained: 0.92014-02-020.9.22014-07-23Old version, not maintained: 1.02014-05-261.0.22014-08-05Old version, not maintained: 1.12014-09-111.1.12014-11-26Old version, not maintained: 1.22014-12-181.2.22015-04-17Old version, not maintained: 1.32015-03-131.3.12015-04-17Old version, not maintained: 1.42015-06-111.4.12015-07-15Old version, not maintained: 1.52015-09-091.5.22015-11-09Old version, not maintained: 1.62016-01-041.6.32016-11-07Old version, not maintained: 2.02016-07-262.0.22016-11-14Old version, not maintained: 2.12016-12-282.1.32018-06-26Old version, not maintained: 2.22017-07-112.2.32019-01-11Old version, not maintained: 2.32018-02-282.3.42019-09-09Old version, not maintained: 2.4 LTS2018-11-022.4.82021-05-17[38]Old version, not maintained: 3.02020-06-183.0.32021-06-01[39]Old version, not maintained: 3.12021-03-023.1.32022-02-18[40]Old version, not maintained: 3.22021-10-133.2.42023-04-13[41]Old version, still maintained: 3.32022-06-163.3.32023-08-21[42]Old version, still maintained: 3.42023-04-133.4.32024-04-18[43]Latest version: 3.52023-09-093.5.22024-08-10[44]Legend:Old version, not maintainedOld version, still maintainedLatest versionLatest preview versionFuture versionSpark 3.5.2 is based on Scala 2.13 (and thus works with Scala 2.12 and 2.13 out-of-the-box), but it can also be made to work with Scala 3.[45]Apache Spark is developed by a community. The project is managed by a group called the "Project Management Committee" (PMC).[46]Feature release branches will, generally, be maintained with bug fix releases for a period of 18 months. For example, branch 2.3.x is no longer considered maintained as of September 2019, 18 months after the release of 2.3.0 in February 2018. No more 2.3.x releases should be expected after that point, even for bug fixes.The last minor release within a major a release will typically be maintained for longer as an LTS release. For example, 2.4.0 was released on November 2, 2018, and had been maintained for 31 months until 2.4.8 was released in May 2021. 2.4.8 is the last release and no more 2.4.x releases should be expected even for bug fixes.[47]Big dataDistributed computingDistributed data processingList of Apache Software Foundation projectsList of concurrent and parallel programming languagesMapReduce^ Called SchemaRDDs before Spark 1.3[18]^ "Spark Release 2.0.0". MLlib in R: SparkR now offers MLlib APIs [..] Python: PySpark now offers many more MLlib algorithms"^ a b c d Zaharia, Matei; Chowdhury, Mosharaf; Franklin, Michael J.; Shenker, Scott; Stoica, Ion. Spark: Cluster Computing with Working Sets (PDF). USENIX Workshop on Hot Topics in Cloud Computing (HotCloud).^ "Spark 2.2.0 Quick Start". apache.org. 2017-07-11. Retrieved 2017-10-19. we highly recommend you to switch to use Dataset, which has better performance than RDD^ "Spark 2.2.0 deprecation list". apache.org. 2017-07-11. Retrieved 2017-10-10.^ Damji, Jules (2016-07-14). "A Tale of Three Apache Spark APIs: RDDs, DataFrames, and Datasets: When to use them and why". databricks.com. Retrieved 2017-10-19.^ Chambers, Bill (2017-08-10). "12". Spark: The Definitive Guide. O'Reilly Media. virtually all Spark code you run, where DataFrames or Datasets, compiles down to an RDD[permanent dead link]^ "What is Apache Spark? Spark Tutorial Guide for Beginner". janbasktraining.com. 2018-04-13. Retrieved 2018-04-13.^ Zaharia, Matei; Chowdhury, Mosharaf; Das, Tathagata; Dave, Ankur; Ma, Justin; McCauley, Murphy; J., Michael; Shenker, Scott; Stoica, Ion (2010). Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing (PDF). USENIX Symp. Networked Systems Design and Implementation.^ Xin, Reynold; Rosen, Josh; Zaharia, Matei; Franklin, Michael; Shenker, Scott; Stoica, Ion (June 2013). Shark: SQL and Rich Analytics at Scale (PDF). SIGMOD 2013. arXiv:1211.6176. Bibcode:2012arXiv1211.6176X.^ Harris, Derrick (28 June 2014). "4 reasons why Spark could jolt Hadoop into hyperdrive". Gigaom. Archived from the original on 24 October 2017. Retrieved 25 February 2016.^ "Cluster Mode Overview - Spark 2.4.0 Documentation - Cluster Manager Types". apache.org. Apache Foundation. 2019-07-09. Retrieved 2019-07-09.^ Figure showing Spark in relation to other open-source Software projects including Hadoop^ MapR ecosystem support matrix^ Doan, DuyHai (2014-09-10). "Re: cassandra + spark / pyspark". Cassandra User (Mailing list). Retrieved 2014-11-21.^ Wang, Yandong; Goldstone, Robin; Yu, Weikuan; Wang, Teng (May 2014). "Characterization and Optimization of Memory-Resident MapReduce on HPC Systems". 2014 IEEE 28th International Parallel and Distributed Processing Symposium. IEEE. pp.799808. doi:10.1109/IPDPS.2014.87. ISBN978-1-4799-3800-1. S2CID11157612.^ a b dotnet/spark, .NET Platform, 2020-09-14, retrieved 2020-09-14^ "GitHub - DFDX/Spark.jl: Julia binding for Apache Spark". GitHub. 2019-05-24.^ "Spark Release 1.3.0 | Apache Spark".^ "Applying the Lambda Architecture with Spark, Kafka, and Cassandra | Pluralsight". www.pluralsight.com. Retrieved 2016-11-20.^ Shapira, Gwen (29 August 2014). "Building Lambda Architecture with Spark Streaming". cloudera.com. Cloudera. Archived from the original on 14 June 2016. Retrieved 17 June 2016. re-use the same aggregates we wrote for our batch application on a real-time data stream^ Chintapalli, Sanket; Dagit, Derek; Evans, Bobby; Farivar, Reza; Graves, Thomas; Holderbaugh, Mark; Liu, Zhuo; Nusbaum, Kyle; Patil, Kishorkumar; Peng, Boyang Jerry; Poulosky, Paul (May 2016). "Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming". 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE. pp.17891792. doi:10.1109/IPDPSW.2016.138. ISBN978-1-5090-3682-0. S2CID2180634.^ Kharbanda, Arush (17 March 2015). "Getting Data into Spark Streaming". sigmoid.com. Sigmoid (Sunnyvale, California IT product company). Archived from the original on 15 August 2016. Retrieved 7 July 2016.^ Zaharia, Matei (2016-07-28). "Structured Streaming In Apache Spark: A new high-level API for streaming". databricks.com. Retrieved 2017-10-19.^ "On-Premises vs. Cloud Data Warehouses: Pros and Cons". SearchDataManagement. Retrieved 2022-10-16.^ Sparks, Evan; Talwalkar, Ameet (2013-08-06). "Spark Meetup: MLbase, Distributed Machine Learning with Spark". slideshare.net. Spark User Meetup, San Francisco, California. Retrieved 10 February 2014.^ "MLlib | Apache Spark". spark.apache.org. Retrieved 2016-01-18.^ Malak, Michael (14 June 2016). "Finding Graph Isomorphisms In GraphX And GraphFrames: Graph Processing vs. Graph Database". slideshare.net. sparksummit.org. Retrieved 11 July 2016.^ Malak, Michael (1 July 2016). Spark GraphX in Action. Manning. p.89. ISBN9781617292521. Pregel and its little sibling aggregateMessages() are the cornerstones of graph processing in GraphX. ... algorithms that require more flexibility for the terminating condition have to be implemented using aggregateMessages()^ Malak, Michael (14 June 2016). "Finding Graph Isomorphisms In GraphX And GraphFrames: Graph Processing vs. Graph Database". slideshare.net. sparksummit.org. Retrieved 11 July 2016.^ Gonzalez, Joseph; Xin, Reynold; Dave, Ankur; Crankshaw, Daniel; Franklin, Michael; Stoica, Ion (Oct 2014). GraphX: Graph Processing in a Distributed Dataflow Framework (PDF). OSDI 2014.^ ".NET for Apache Spark | Big data analytics". 15 October 2019.^ "Spark.jl". GitHub. 14 October 2021.^ a b Clark, Lindsay. "Apache Spark speeds up big data decision-making". ComputerWeekly.com. Retrieved 2018-05-16.^ "The Apache Software Foundation Announces Apache™ Spark™ as a Top-Level Project". apache.org. Apache Software Foundation. 27 February 2014. Retrieved 4 March 2014.^ Spark officially sets a new record in large-scale sorting^ Open HUB Spark development activity^ "The Apache Software Foundation Announces Apache™ Spark™ as a Top-Level Project". apache.org. Apache Software Foundation. 27 February 2014. Retrieved 4 March 2014.^ "Spark 2.4.8 released". spark.apache.org. Archived from the original on 2021-08-25.^ "Spark 3.0.3 released". spark.apache.org.^ "Spark 3.1.3 released". spark.apache.org. Archived from the original on 2022-06-18.^ "Spark 3.2.4 released". spark.apache.org.^ "Spark 3.3.3 released". spark.apache.org.^ "Spark 3.4.3 released". spark.apache.org.^ "Spark 3.5.2 released". spark.apache.org.^ "Using Scala 3 with Spark". 47 Degrees. Retrieved 29 July 2022.^ "Apache Committee Information".^ "Versioning policy". spark.apache.org.Official website Retrieved from " by Markus Winkler on UnsplashThis is the first part of a collection of examples of how to use the MLlib Spark library with Python. The content in this post is a conversion of this Jupyter notebook.This notebook is a collection of examples that illustrate how to use PySpark with MLlib. This whole collection of examples is intended to be a gentle introduction to those interested in the topic, want to have additional examples, or simply are curious about how to start with this library.The following skills are expected to follow these examples:PythonUnderstanding Spark structures (Dataframes, RDD)Basic ML notions. This is not intended to be a ML course although, you can find some theoretical explanations.The examples are designed to work with a simple local environment using the MLlib Dataframe. The MLlib RDD-based API is now in maintenance (see here). This is why you will see that the main import statement comes from pyspark.ml not pyspark.mllib.You will need a spark environment to be available in your local path. Refer here to the official guide for more details.You will need java to be available on your local path. Check it out running java --version. If nothing is displayed, check out how to install Java on your machine (here). Next, you can easily set up a local environment and install the dependencies used in this notebook.virtualenv envsource env/bin/activatepip install urllib3 numpy matplotlib pysparkWait until the dependencies are already satisfied. If you are not reading these lines from a Jupyter Notebook :) install and run it.pip install jupyterjupyter-notebookSpark Dataframes support a collection of rows containing elements with different data types. In the context of ML algorithms, data types such as boolean, string or even integer are not the expected input for most ML algorithms. In this sense, MLlib supports data types such as vectors or matrices.2.1 Dense VectorsA dense vector is an array. PySpark uses numpy to run algebraical operations.Docsfrom pyspark.ml.linalg import DenseVectora = DenseVector([0,1,2,3,4])b = DenseVector([10,10,10,10,10])print('Sum: ', a + b)print('Difference: ', a - b)print('Multiplication: ', a * 2)print('Division: ', b / 2)print('Non-zeros: ', a.nonzero())print('Squared distance: ', a.squared_distance(b))Sum: [10.0,11.0,12.0,13.0,14.0]Difference: [-10.0,-9.0,-8.0,-7.0,-6.0]Multiplication: [0.0,2.0,4.0,6.0,8.0]Division: [5.0,5.0,5.0,5.0,5.0]Non-zeros: (array([1, 2, 3, 4]),)Squared distance: 330.02.2 Sparse VectorsSparse vectors are designed to represent those vectors where a large number of elements is expected to be zero. These vectors are defined by specifying which positions of the array are different from zero and the assigned values. In the following vector:SparseVector (5 ,[0 ,2 ,4] ,[1 ,3 ,5])we have five elements with entries 0, 2, and 4 take values 1, 3, and 5.Docsfrom pyspark.ml.linalg import SparseVectorsparse_vector = SparseVector (5 ,[0 ,2 ,4] ,[1 ,3 ,5])print('Sparse vector: ', sparse_vector.toArray ())print('Indices: ', sparse_vector.indices)print('Non zeros: ', sparse_vector.numNonzeros ())Sparse vector: [1. 0. 3. 0. 5.]Indices: [0 2 4]Non zeros: 3We can expect datasets to be available from different storage sources:Hard disksHDFSDatabasesOthersThe SparkSession object facilitates the load of data from these sources under different formats (CSV, JSON, text, parquet, databases, etc.). We will show examples for CSV, libSVM, and images.3.1 CSVLets assume the following dataset in a CSV format:csvlabel,f1,f2,f3,f40,0,"one",2.0,true1,4,"five",6.0,falseWe instantiate a SparkSession object and load the dataset indicating that we have a header and the separation character.'''For this example we need the dataset.csv file to be available. Copy and paste the following lines:echo "\label,f1,f2,f3,f40,0,"one",2.0,true1,4,"five",6.0,false" > /tmp/dataset.csv'''from pyspark.sql import SparkSession# Get a session object for our current Spark mastersession = SparkSession.builder.appName("Example").master("local").getOrCreate()dataset = session.read.format('csv')\.option('header', 'true')\.option('sep', ',')\.load('/tmp/dataset.csv')dataset.show() # we stop the sessionsession.stop()+-----+---+----+---+-----+|label| f1| f2| f3| f4|+-----+---+----+---+-----+| 0| 0| one|2.0| true|| 1| 4|five|6.0|false|+-----+---+----+---+-----+3.2 libSVMLibSVM is a popular format to represent numeric sparse data.The following dataset:0 128:51 129:1591 130:253 131:159 132:501 155:48 156:238Where the first row 0 128:51 129:159 indicates an observation with label 0 and feature 128th and 129th equal to 51 and 159 respectively. We can load this dataset using the SparkSession object as we did for the CSV format.'''For this example we need the dataset.libsvm file to be available. Copy and paste the following lines:echo "\0 128:51 129:1591 130:253 131:159 132:501 155:48 156:238" > /tmp/dataset.libsvm'''from pyspark.sql import SparkSessionsession = SparkSession.builder.appName("Example").master("local").getOrCreate()dataset = session.read.format('libsvm').option('numFeatures',157).load('/tmp/dataset.libsvm')dataset.show() # we stop the sessionsession.stop()+-----+--------------------+|label| features|+-----+--------------------+| 0.0|(157,[127,128],[5...|| 1.0|(157,[129,130,131...|| 1.0|(157,[154,155],[4...|+-----+--------------------+3.3 ImagesLMlib can load images in variety of formats (jpeg, png, etc.). It also supports compressed formats. The resulting DataFrame has a column image containing information of the schema.More details in the docs.# We download a cat image for laterimport urllib3import tempfilefrom IPython.display import Imageimport sysurl=' = tempfile.gettempdir() + '/kitty.png'http = urllib3.PoolManager()r = http.request('GET', url, preload_content=False) with open(cat_image, 'wb') as f: while True: data = r.read() if not data: break f.write(data)r.release_conn()Image(filename=cat_image)from pyspark.sql import SparkSessionsession = SparkSession.builder.appName('Example').master('local').getOrCreate()df = session.read.format('image').option('dropInvalid', True).load(cat_image)df.select('image.origin', 'image.width', 'image.height', 'image.nChannels', 'image.mode').show(truncate=False)# The image data is stored in the image.data column, one image per row.img_data = df.select('image.data').collect()[0]# Do something with img_data...session.stop()+---------------------+-----+------+---------+----+|origin |width|height|nChannels|mode|+---------------------+-----+------+---------+----+|file:///tmp/kitty.png|640 |960 |3 |16 |+---------------------+-----+------+---------+----+One of the main tasks for any data engineer is data preparation. For two reasons:Raw data is not ready to be consumed by algorithmsPreprocessing data is required to improve algorithms performanceIncoming data has to be processed in different steps until we reach a successful representation to be consumed by algorithms. MLlib offers a collection of feature-related operations. We can distinguish:Extraction: extract features from raw dataTransformation: modifying/converting featuresSelection: select features based on a certain criteriaLocality Sensitive Hashing (LSH): algorithms combining feature transformation with other algorithmsIn general feature processing in MLlib follows these steps:Instantiate the operator indicating the name of the input and output columns and additional params.Fit the model invoking the .fit(...) method to train a model. Some operators may not require this step if they are not associated with a model.Transform the input data using the modelNo need to mention that these steps, the input params, and the input format vary from operator to operator.The following sections present succint examples of different operators.4.1 Normal standardizationScale a feature to obtain a normal distribution with mean 0 and unit-variance.from pyspark.sql import SparkSessionfrom pyspark.ml.linalg import Vectorsfrom pyspark.ml.feature import StandardScalersession = SparkSession.builder.appName('Example').master('local').getOrCreate()values = [[0, Vectors .dense ([1.0,0.1, -1.0])], [1, Vectors .dense ([2.0 ,1.1 ,1.0])] , [2, Vectors .dense ([3.0 ,10.1 ,3.0])]]dataset = session.createDataFrame(values, ['id', 'features'])dataset.show()+---+--------------+| id| features|+---+--------------+| 0|[1.0,0.1,-1.0]|| 1| [2.0,1.1,1.0]|| 2|[3.0,10.1,3.0]|+---+--------------+# Fitscaler = StandardScaler(inputCol = 'features', outputCol='standardized', withMean=True, withStd=True)scalerModel = scaler.fit(dataset)# Print some statisticsprint("Mean is: %s with sd: %s" % (scalerModel.mean, scalerModel.std))# Transformstandardized = scalerModel.transform(dataset)standardized.show()Mean is: [2.0,3.7666666666666666,1.0] with sd: [1.0,5.507570547286102,2.0]+---+--------------+--------------------+| id| features| standardized|+---+--------------+--------------------+| 0|[1.0,0.1,-1.0]|[-1.0,-0.66575028...|| 1| [2.0,1.1,1.0]|[0.0,-0.484182026...|| 2|[3.0,10.1,3.0]|[1.0,1.1499323120...|+---+--------------+--------------------+We can check if the transformed data has the desired distribution using a Summarizer (docs here). For every feature we have mean set to 0 and standard deviation equal to 1.# Let's see what are the mean and the std nowfrom pyspark.ml.stat import Summarizersummarizer = Summarizer.metrics("mean", "std")standardized.select(summarizer.summary(standardized.standardized)).show(truncate=False)session.stop()+-----------------------------------+|aggregate_metrics(standardized, 1.0) |+-----------------------------------+|{[0.0,0.0,0.0],[1.0,0.9999999999999999,1.0]}|+-----------------------------------+4.2 Elementwise productThis transformer multiplies each input vector by a provided vector, using element-wise multiplication. This operation scales each column by a given scalar (Hadamard product).from pyspark.sql import SparkSessionfrom pyspark.ml.linalg import Vectorsfrom pyspark.ml.feature import ElementwiseProductsession = SparkSession.builder.appName('Example').master('local').getOrCreate()a = [[Vectors.dense ([2,3,1])], [Vectors.dense ([0,8,-2])]]b = Vectors.dense ([3,1,4])print('b =',b)df_a = session.createDataFrame(a, ['features'])df_a.show()ewp = ElementwiseProduct(inputCol='features', outputCol='product', scalingVec=b)a_b = ewp.transform(df_a)a_b.show()b = [3.0,1.0,4.0+--------------+| features|+--------------+| [2.0,3.0,1.0]||[0.0,8.0,-2.0]|+--------------++--------------+--------------+| features| product|+--------------+--------------+| [2.0,3.0,1.0]| [6.0,3.0,4.0]||[0.0,8.0,-2.0]|[0.0,8.0,-8.0]|+--------------+--------------+4.3 Principal Component AnalysisWhen dealing with many features we can come across the curse of dimensionality.More than three variables are difficult to plotPerformance issues for a large number of featuresFeatures that only add noise to the problemAlgorithms may find difficult to converge to a solutionPrincipal Component Analysis (PCA) is a dimensionality reduction technique that aims to find the components that maximize the variance. These are the steps to follow:Standardize the dataCompute eigenvectors and eigenvalues of the covariance matrixSort eigenvalues and pick the d largest valuesConstruct matrix W using the d corresponding eigenvectorsTransform dataset X multiplying it by WIn MLlib there is a PCA transformer that implements all these steps. By applying the PCA we can obtain a reduced version of the original that maintains most of the relevant information brought by the features.In the example below, we compute the PCA for a dataset of 5 features we wish to convert in a new 3 features dataset.from pyspark.sql import SparkSessionfrom pyspark.ml.linalg import Vectorsfrom pyspark.ml.feature import PCAsession = SparkSession.builder.appName("Example").master("local").getOrCreate()data = [[Vectors.dense ([1.0 , 0.0, 3.0, 0.0, 7.0])], [Vectors .dense ([2.0 , 0.0, 3.0, 4.0, 5.0])], [Vectors .dense ([4.0 , 0.0, 0.0, 6.0, 7.0])]]dataset = session.createDataFrame(data, ['features'])dataset.show()+--------------------+| features|+--------------------+|[1.0,0.0,3.0,0.0,...||[2.0,0.0,3.0,4.0,...||[4.0,0.0,0.0,6.0,...|+--------------------+# Fit PCApca = PCA(inputCol='features', outputCol='pcaFeatures', k=3)pcaModel = pca.fit(dataset)print("The variance for every new feature %s" % pcaModel.explainedVariance)The variance for every new feature [0.84375,0.15625000000000008,4.509331675237028e-17]The variance for every new feature [0.84375,0.15625000000000008,4.509331675237028e-17# Transform the original datasetpcaDataset = pcaModel.transform(dataset)pcaDataset.show(truncate=False)+---------------------+---+|features |pcaFeatures |+---------------------+---+|[1.0,0.0,3.0,0.0,7.0]|[0.8164965809277265,3.65148371670111,-2.5144734900027204]||[2.0,0.0,3.0,4.0,5.0]|[-2.857738033247042,0.9128709291752779,-2.514473490002722]||[4.0,0.0,0.0,6.0,7.0]|[-6.53197264742181,3.6514837167011094,-2.514473490002719]|+---------------------+---+Observe that the transformed dataset does no longer correspond to any real observation. Any model predictions generated using this transformed data for training, has to be reconstructed. Otherwise, the output will no make any sense.4.4 StringIndexerThis is a label indexer that assigns a label to every string in a column. If the value is numeric, first it is casted to string and then indexed.from pyspark.sql import SparkSessionfrom pyspark.ml.linalg import Vectorsfrom pyspark.ml.feature import StringIndexersession = SparkSession.builder.appName('Example').master('local').getOrCreate()data = session.createDataFrame([['blue'], ['red'], ['red'], ['white'], ['yellow'], ['red']], ['feature'])data.show()si = StringIndexer(inputCol='feature', outputCol='index',)model = si.fit(data)print('Found labels: ', model.labels)model.transform(data).show()session.stop()+-------+|feature|+-------+| blue|| red|| red|| white|| yellow|| red|+-------+Found labels: ['red', 'blue', 'white', 'yellow']+-------+-----+|feature|index|+-------+-----+| blue| 1.0|| red| 0.0|| red| 0.0|| white| 2.0|| yellow| 3.0|| red| 0.0|+-------+-----+4.4 One hot encoderThis encoder maps a column of indices into a single binary vector. If we have 4 labels, for index 3 we will have [0,0,0,1,0]. The output is a SparseVector.Observe that the param dropLast is True by default ignoring the label with index n-1.from pyspark.sql import SparkSessionfrom pyspark.ml.linalg import Vectorsfrom pyspark.ml.feature import StringIndexer, OneHotEncodersession = SparkSession.builder.appName('Example').master('local').getOrCreate()data = session.createDataFrame([['blue'], ['red'], ['red'], ['white'], ['yellow'], ['red']], ['feature'])data.show()# First we need an input column with indices instead of strings.si = StringIndexer(inputCol='feature', outputCol='indexed')si_model = si.fit(data)indexed = si_model.transform(data)indexed.show()# If we let dropLast=True, the index for yellow will be droppedohe = OneHotEncoder(inputCol='indexed', outputCol='encoded', dropLast=False)ohe_model = ohe.fit(indexed)ohe_transformed = ohe_model.transform(indexed)'''You can check how setting dropLast=True, the 4th row will be+-------+-------+-------------+|feature|indexed| encoded|+-------+-------+-------------+| yellow| 3.0|(3,[],[]) |+-------+-------+-------------+'''ohe_transformed.show()+-------+|feature|+-------+| blue|| red|| red|| white|| yellow|| red|+-------+-------+|feature|indexed|+-------+-------+| blue| 1.0|| red| 0.0|| red| 0.0|| white| 2.0|| yellow| 3.0|| red| 0.0|+-------+-------++-------+-------+-------------+|feature|indexed| encoded|+-------+-------+-------------+| blue| 1.0|(4,[1],[1.0])|| red| 0.0|(4,[0],[1.0])|| red| 0.0|(4,[0],[1.0])|| white| 2.0|(4,[2],[1.0])|| yellow| 3.0|(4,[3],[1.0])|| red| 0.0|(4,[0],[1.0])|+-------+-------+-------------+4.5 TokenizationTokenization is the process of splitting a document into a vector of differentiated tokens. The sentence "The quick brown fox jumps over the lazy dog" will be split into tokens like in ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"]. Different approaches may split the document using white spaces, commas, regular expressions, or any other character.In MLlib there is a Tokenizer transformer for this purpose.from pyspark .sql import SparkSessionfrom pyspark .ml. feature import Tokenizersession = SparkSession.builder.appName("Example").master("local").getOrCreate()sentenceDataFrame = session.createDataFrame ([(0, "Hi I heard about Spark"), (1, "I wish Java could use case classes"), (2, "Logistic, regression, models, are, neat")], ['id', 'sentence'])tokenizer = Tokenizer(inputCol='sentence', outputCol='words')tokenizer.transform(sentenceDataFrame).show(truncate=False)+---+-----------------------------------+--+|id |sentence |words |+---+-----------------------------------+--+|0 |Hi I heard about Spark |[hi, i, heard, about, spark] ||1 |I wish Java could use case classes |[i, wish, java, could, use, case, classes] ||2 |Logistic, regression, models, are, neat|[logistic,, regression,, models,, are,, neat]|+---+-----------------------------------+--+4.6 Stop wordsNatural language is redundant and not every term provides the same amount of information. By stop words we refer to the most common words in a given language. These words are so common that result into non-relevant chunks of information. These words are removed previously to any analysis. There is no a single list of stop words and this changes with every language.MLlib implements the StopWordsRemover that filters out stop words using a dictionary.from pyspark .sql import SparkSessionfrom pyspark .ml. feature import StopWordsRemoversession = SparkSession.builder.appName("Example").master("local").getOrCreate()text = session.createDataFrame ([(0, ["I", "saw", "the", "red", " balloon "]), (1, ["Mary", "had", "a", " little ", "lamb"])], ["id", "raw"])text.show(truncate=False)remover = StopWordsRemover(inputCol='raw', outputCol='filtered')remover.transform(text).show(truncate=False)+---+-----------------------------+|id |raw |+---+-----------------------------+|0 |[I, saw, the, red, balloon] ||1 |[Mary, had, a, little , lamb]|+---+-----------------------------++---+-----------------------------+---------------------+|id |raw |filtered |+---+-----------------------------+---------------------+|0 |[I, saw, the, red, balloon] |[saw, red, balloon] ||1 |[Mary, had, a, little , lamb]|[Mary, little , lamb]|+---+-----------------------------+---------------------+4.7 Count VectorizerThis estimator counts the number of occurrences of items in a vocabulary represented in a sparse vector. This is particularly useful to represent a document in terms of the frequency of its elements and it is normally used in probabilistic models.from pyspark .sql import SparkSessionfrom pyspark.ml.feature import CountVectorizersession = SparkSession.builder.appName("Example").master("local").getOrCreate()text = session.createDataFrame ([(0,'yellow red blue'.split()), (1, 'red'.split()), (2, 'blue white blue'.split()),], ["id", "raw"])text.show()cv = CountVectorizer(inputCol='raw', outputCol='frequencies')cv_model = cv.fit(text)print('The vocabulary: ',cv_model.vocabulary)frequencies = cv_model.transform(text)frequencies.show(truncate=False)+---+------------------+| id| raw|+---+------------------+| 0|[yellow, red, blue]|| 1| [red]|| 2|[blue, white, blue]|+---+------------------+The vocabulary: ['blue', 'red', 'white', 'yellow']+---+------------------+-------------------------+|id |raw |frequencies |+---+------------------+-------------------------+|0 |[yellow, red, blue]|(4,[0,1,3],[1.0,1.0,1.0])||1 |[red] |(4,[1],[1.0]) ||2 |[blue, white, blue]|(4,[0,2],[2.0,1.0]) |+---+------------------+-------------------------+4.8 N-gramsN-grams are a common input for many algorithms to understand the probability of n words to occur together. The NGram transformer outputs a collection of these N-grams.from pyspark .sql import SparkSessionfrom pyspark.ml.feature import IDF, Tokenizer, NGramsession = SparkSession.builder.appName("Example").master("local").getOrCreate()text = session.createDataFrame ([(0, "Hi I heard about Spark"), (0, "I wish Java could use case classes "), (0, "Logistic regression models are neat")], ["label", "sentence"])# First we tokenize our datasettokenizer = Tokenizer(inputCol ='sentence', outputCol ='words')words = tokenizer.transform(text)words.show(truncate=False)# Compute 2-gramsngram = NGram(inputCol='words', outputCol='ngrams', n=2)ngrams = ngram.transform(words)ngrams.show(truncate=False)+-----+----------------------------------+--+|label|sentence |words |+-----+----------------------------------+--+|0 |Hi I heard about Spark |[hi, i, heard, about, spark] ||0 |I wish Java could use case classes |[i, wish, java, could, use, case, classes]||0 |Logistic regression models are neat|[logistic, regression, models, are, neat] |+-----+----------------------------------+--++-----+----------------------------------+--+--+|label|sentence |words |ngrams |+-----+----------------------------------+--+--+|0 |Hi I heard about Spark |[hi, i, heard, about, spark] |[hi i, i heard, heard about, about spark] ||0 |I wish Java could use case classes |[i, wish, java, could, use, case, classes]|[i wish, wish java, java could, could use, use case, case classes]||0 |Logistic regression models are neat|[logistic, regression, models, are, neat] |[logistic regression, regression models, models are, are neat] |+-----+----------------------------------+--+--+4.9 Word2VecThe Word2Vec represents the words of a document in a vector. This makes possible to operate with documents as vectors which makes possible to easily computes distances and enables other algorithms specially in NLP. Take a look at the original Google code here.from pyspark.sql import SparkSessionfrom pyspark.ml.feature import Word2Vecsession = SparkSession.builder.appName('Example').master('local').getOrCreate()corpus = session.createDataFrame([('Spark is quite useful'.split(),), ('I can use Spark with Python'.split(),), ('Spark is not so difficult after all'.split(),),], ['words'])corpus.show(truncate=False)w2v = Word2Vec(inputCol='words', outputCol='result', vectorSize=3, minCount=0)w2v_model = w2v.fit(corpus)vectors = w2v_model.transform(corpus)vectors.show(truncate=False)+---+|words |+---+|[Spark, is, quite, useful] ||[I, can, use, Spark, with, Python] ||[Spark, is, not, so, difficult, after, all]|+---++---+---+|words |result |+---+---+|[Spark, is, quite, useful] |[0.08182145655155182,-0.07318692095577717,-0.0631803400174249] ||[I, can, use, Spark, with, Python] |[0.016474373017748196,-1.7273581276337305E-4,-0.04478610997709135]||[Spark, is, not, so, difficult, after, all]|[0.019738022836723497,0.029656097292900085,-0.033315843919159045] |+---+---+Most models are computed as a concatenation of operations, each operation transforming the original dataset. For example, normalization -> component analysis -> regression. MLlib uses the concept of pipelines (similinar to the one used in SciKit) to unify the execution of a sequence of steps into a single object.The pipeline is defined as a sequence of stages connecting transformers and estimators:Transformer: receives an input dataframe and returns a transformed version (standardizers)Estimator: receives an input dataframe and after fitting returns a transformer (linear regression, logistic regression, etc.)Creating a pipeline is equivalent to set the sequence of stages to be executed.from pyspark.ml.pipeline import Pipelinepipeline = Pipeline(stages=[standardizer, pca, lr])Then we fit the model and transform the dataset to get the corresponding results:model = pipeline.fit(dataset)model.transform(dataset).show()from pyspark.sql import SparkSessionfrom pyspark.ml.feature import Tokenizer, StopWordsRemover, CountVectorizerfrom pyspark.ml.pipeline import Pipelinesession = SparkSession.builder.appName('Example').master('local').getOrCreate()corpus = session.createDataFrame([('Spark is quite useful',), ('I can use Spark with Python',), ('Spark is not so difficult after all',),], ['docs'])corpus.show(truncate=False)tokenizer = Tokenizer(inputCol='docs', outputCol='tokens')stop_remover = StopWordsRemover(inputCol='tokens', outputCol='filtered')cv = CountVectorizer(inputCol='filtered', outputCol='frequencies')pipeline = Pipeline(stages=[tokenizer, stop_remover, cv])fitted = pipeline.fit(corpus)result = fitted.transform(corpus)result.show(truncate=False)for m in fitted.stages: print('-->',m.uid) print(m.params)+----------------------------------+|docs |+----------------------------------+|Spark is quite useful ||I can use Spark with Python ||Spark is not so difficult after all|+----------------------------------++----------------------------------+---+---------------------+-------------------------+|docs |tokens |filtered |frequencies |+----------------------------------+---+---------------------+-------------------------+|Spark is quite useful |[spark, is, quite, useful] |[spark, quite, useful]|(6,[0,2,3],[1.0,1.0,1.0])||I can use Spark with Python |[i, can, use, spark, with, python] |[use, spark, python] |(6,[0,1,5],[1.0,1.0,1.0])||Spark is not so difficult after all|[spark, is, not, so, difficult, after, all]|[spark, difficult] |(6,[0,4],[1.0,1.0]) |+----------------------------------+---+---------------------+-------------------------+--> Tokenizer_346e29794e54[Param(parent='Tokenizer_346e29794e54', name='inputCol', doc='input column name.'), Param(parent='Tokenizer_346e29794e54', name='outputCol', doc='output column name.')]--> StopWordsRemover_cdda6836267e[Param(parent='StopWordsRemover_cdda6836267e', name='caseSensitive', doc='whether to do a case sensitive comparison over the stop words'), Param(parent='StopWordsRemover_cdda6836267e', name='inputCol', doc='input column name.'), Param(parent='StopWordsRemover_cdda6836267e', name='inputCols', doc='input column names.'), Param(parent='StopWordsRemover_cdda6836267e', name='locale', doc='locale of the input. ignored when case sensitive is true'), Param(parent='StopWordsRemover_cdda6836267e', name='outputCol', doc='output column name.'), Param(parent='StopWordsRemover_cdda6836267e', name='outputCols', doc='output column names.'), Param(parent='StopWordsRemover_cdda6836267e', name='stopWords', doc='The words to be filtered out')]--> CountVectorizer_983c07eb2c8f[Param(parent='CountVectorizer_983c07eb2c8f', name='binary', doc='Binary toggle to control the output vector values. If True, all nonzero counts (after minTF filter applied) are set to 1. This is useful for discrete probabilistic models that model binary events rather than integer counts. Default False'), Param(parent='CountVectorizer_983c07eb2c8f', name='inputCol', doc='input column name.'), Param(parent='CountVectorizer_983c07eb2c8f', name='maxDF', doc='Specifies the maximum number of different documents a term could appear in to be included in the vocabulary. A term that appears more than the threshold will be ignored. If this is an integer >= 1, this specifies the maximum number of documents the term could appear in; if this is a double in [0,1), then this specifies the maximum fraction of documents the term could appear in. Default (2^63) - 1'), Param(parent='CountVectorizer_983c07eb2c8f', name='minDF', doc='Specifies the minimum number of different documents a term must appear in to be included in the vocabulary. If this is an integer >= 1, this specifies the number of documents the term must appear in; if this is a double in [0,1), then this specifies the fraction of documents. Default 1.0'), Param(parent='CountVectorizer_983c07eb2c8f', name='minTF', doc="Filter to ignore rare words in a document. For each document, terms with frequency/count less than the given threshold are ignored. If this is an integer >= 1, then this specifies a count (of times the term must appear in the document); if this is a double in [0,1), then this specifies a fraction (out of the document's token count). Note that the parameter is only used in transform of CountVectorizerModel and does not affect fitting. Default 1.0"), Param(parent='CountVectorizer_983c07eb2c8f', name='outputCol', doc='output column name.'), Param(parent='CountVectorizer_983c07eb2c8f', name='vocabSize', doc='max size of the vocabulary. Default 1