SparkML

Created by Mahidhar

Reference:SparkReferenceGuide

Introduction

- Apache Spark MLlib is the Apache Spark scalable machine learning library consisting of common learning algorithms and utilities
  - classification,
  - regression,
  - Clustering,
  - collaborative filtering,
  - dimensionality reduction

Spark MLLib seamlessly integrates with other Spark components such as Spark SQL, Spark Streaming, and DataFrames.

ML in Spark

- Spark has come up with Machine Learning in the 0.8 version
- The first version of Machine learning used the RDD’s and the library was called as MLib.
- It used the jblas library(Java Basic Linear Algebra Subprograms) which depends upon the Fortran.
- **As of Spark 2.0, the RDD-based APIs in the spark.mllib package have entered maintenance mode. The primary Machine Learning API for Spark is now the DataFrame-based API in the spark.ml package.**
- SparkML uses the linear algebra package Breeze, which depends on netlib-java for optimised numerical processing. If native libraries are not available at runtime, you will see a warning message and a pure JVM implementation will be used instead.

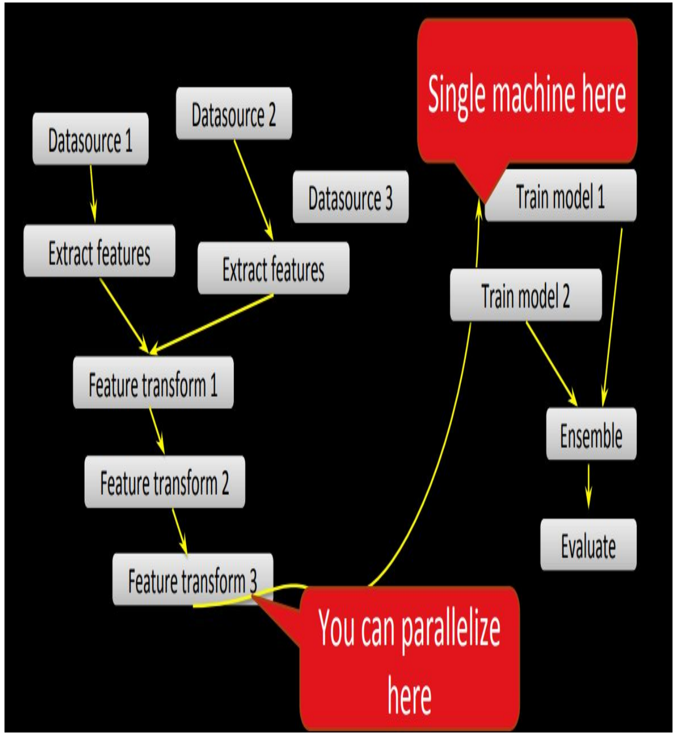
Parallelization Techniques

Feature engineer in Spark but train the model on a single driver machine

○ Gather source data, perform joins, do complex ETL and feature engineer

○ Works well when you have big data for your transactions but smaller aggregated or subsamples once you have features to train on

○ May need to use a larger size driver node that can fit all the training features into memory (may need to adjust spark.driver.maxResultSize)



contd..

Train the entire model with distributed machine learning

○ MLLib built for large scale model creation (millions of instances)

○ You will create a single model with all this data (you use model selection techniques like Train Tune Validation splits or a cross validator)

Create many models one on each worker

○ Works for creating one model per customer, one model per set of features for feature selection, one model per set of hyperparameters for tuning…

○ Spark-sklearn helps facilitate creating one model per worker

○ You can as of Spark 2.3 use the cross validation parallelism parameter to run do model selection / hyperparameter tuning in parallel in MLLib as well

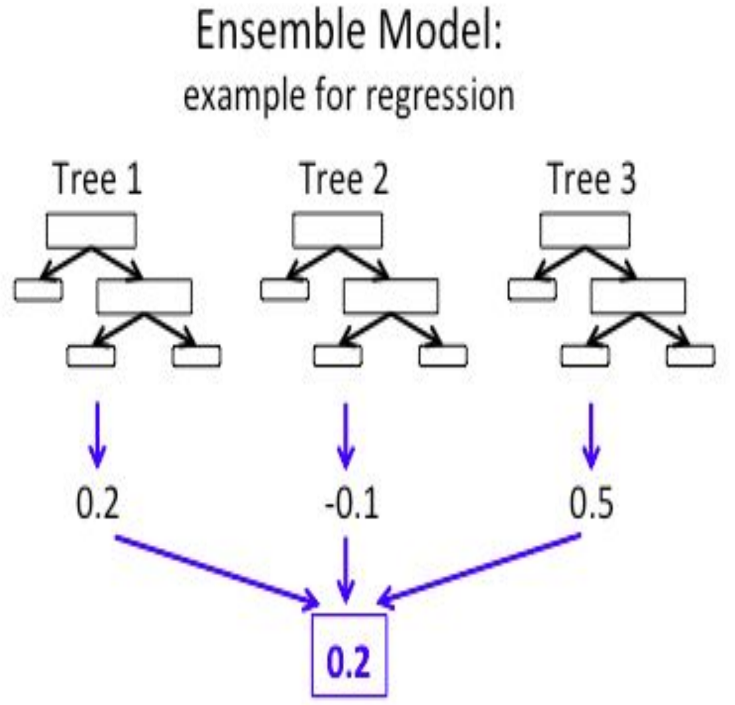
○ Broadcast your data but use Spark to train models on each worker with different sets of hyperparameters to find the optimal model

Distributed Machine Learning

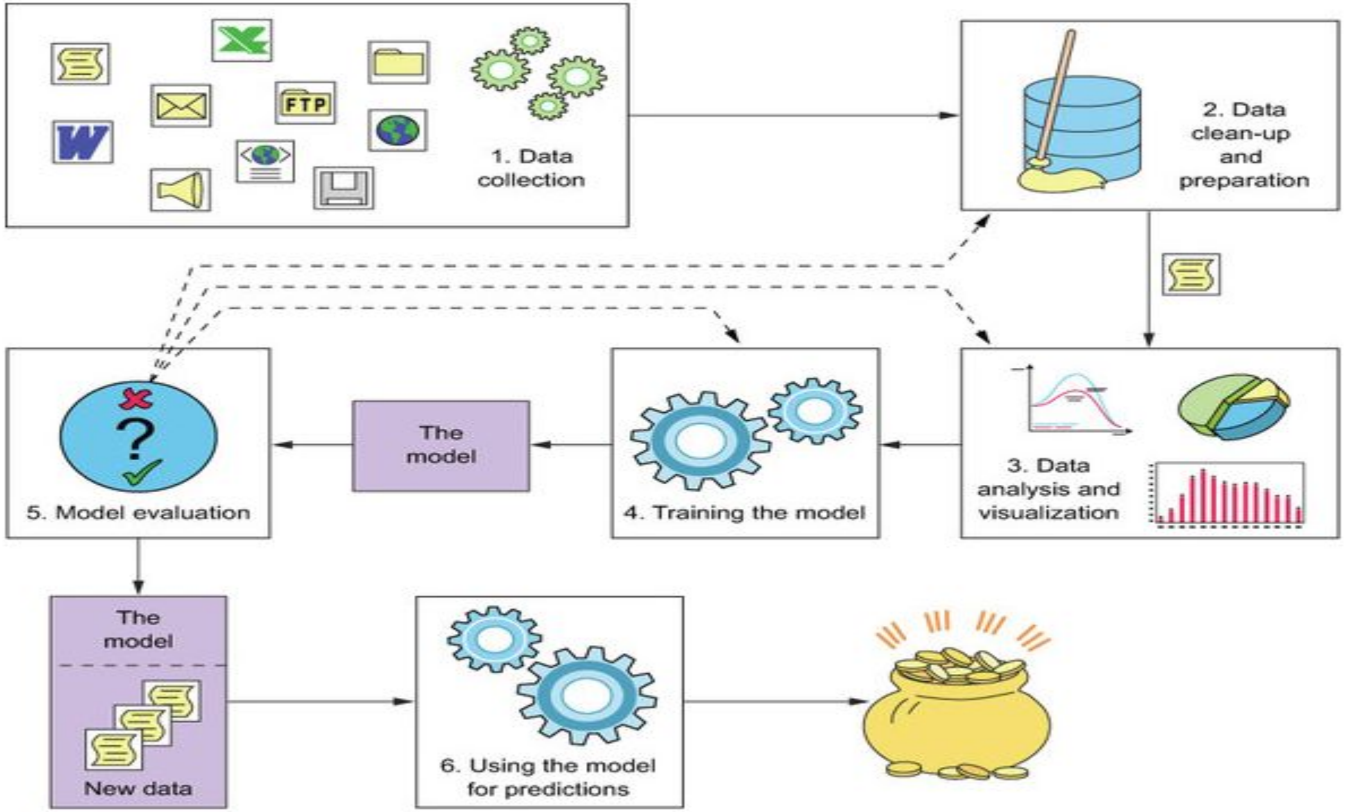
- In Distributed Machine Learning like Spark MLlib, the data is represented as an RDD or a DataFrame typically in distributed file storage such as S3, Blob Store or HDFS .
- As a best practice, you should cache the data in memory across the nodes so that multiple iterations don't have to query the disk each time .
- The calculation of the gradients is distributed across all the nodes using Spark's distributed compute engine (similar to MapReduce)
- After each iteration, the results return to the driver and iterations continue until either max_iter or tol is reached.

Random Forest on Spark

- In Spark because each tree is trained on a subsample of data (and features) and do not depend on the other trees many trees can be trained in parallel.
- The selection of features to split on is also distributed Splits are chosen based on information gain
- Tree creation is stopped when maxDepth is hit, information gain is < minInfoGain or no split candidates have minInstancesPerNode

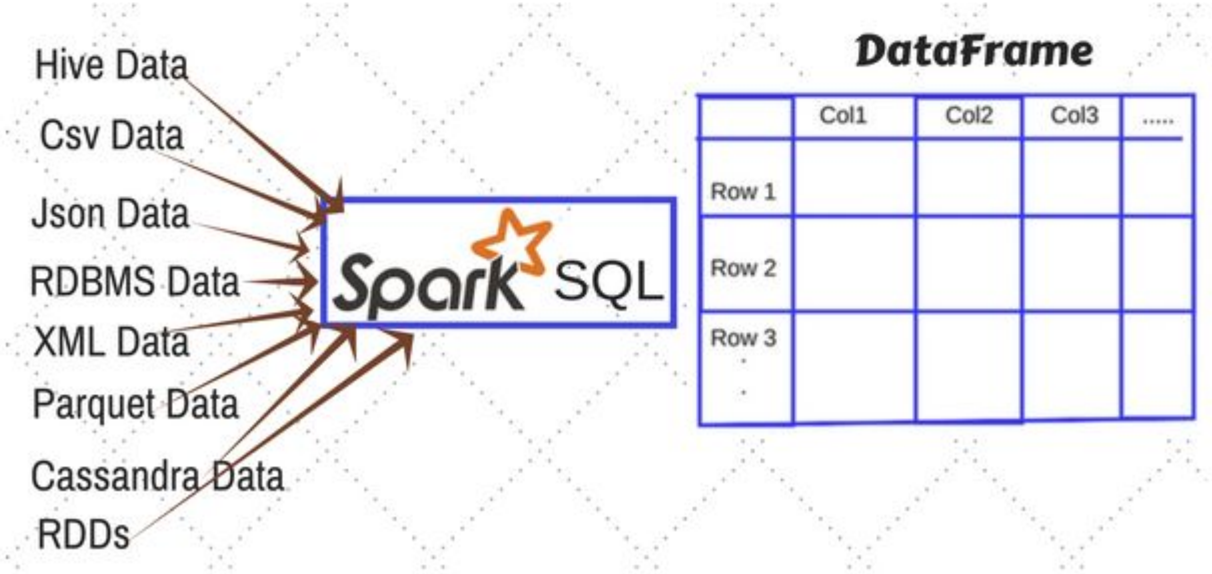


ML contd..

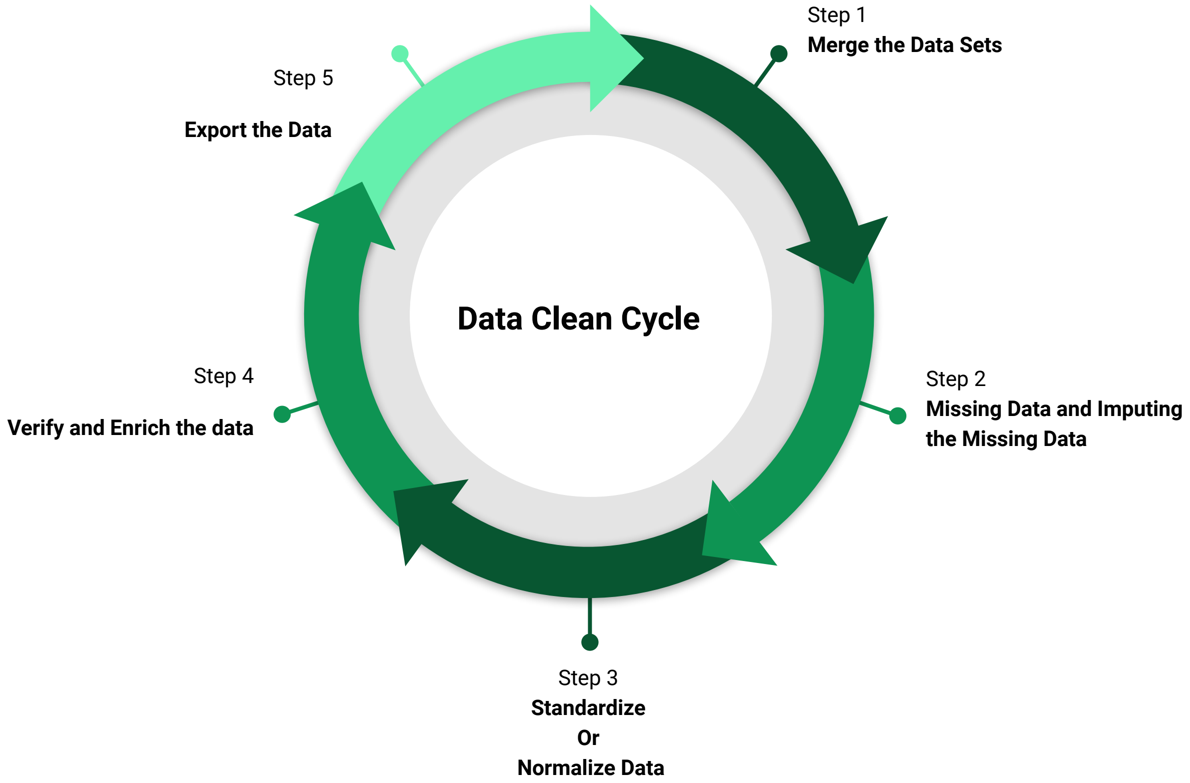

1. Data collection
FTP
2. Data clean-up and preparation
3. Data analysis and visualization
4. Training the model
The model
5. Model evaluation
The model
New data
6. Using the model for predictions

Data Collection

- Spark can read data from multiple sources

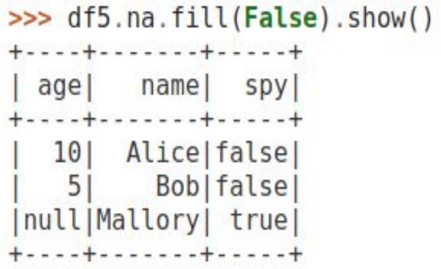


Clean up Data -- Preprocessing

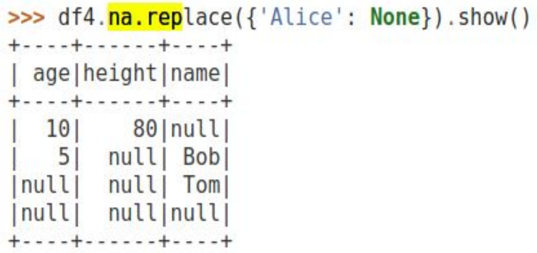


Missing Value Imputation

- Basically we can do with fill.na and na.replace .



```
>>> df5.na.fill(False).show()
+----+-------+-----+
| age|   name|  spy|
+----+-------+-----+
|  10|  Alice|false|
|   5|    Bob|false|
|null|Mallory| true|
+----+-------+-----+
```



```
>>> df4.na.replace({'Alice': None}).show()
+----+------+----+
| age|height|name|
+----+------+----+
|  10|    80|null|
|   5|  null| Bob|
|null|  null| Tom|
|null|  null|null|
+----+------+----+
```

- In current version of our cluster spark 2.3.0 there is a Imputer method in pyspark.ml.feature which helps in imputation and that works for only numerical data.

Standardize/ Normalize Data

- The Spark 2.3.0 provides different ways of Standardizing / Normalizing the data.
  - **Normalizer**
  - **StandardScaler**
  - **MinMaxScaler**
  - **MaxAbsScaler**

Enriching the Data & Dimensionality Reduction

- We can enrich the data by creating the new features using

  df.withColumn()
- We can use PCA in the pyspark.ml.feature

SparkML for Text Data Preprocessing

- The features that are available for the text data are :
  - TF-IDF
  - Word2Vec
  - CountVectorizer
  - FeatureHasher
  - Tokenizer
  - StopWordsRemover
  - n-gram

Analysis and Visualization

- Pyspark documentation did not have any visualization packages or methods
- We can use plotly for visualization which is a python way of doing the visualization
- For Analysis on the basic Statistical bases we have certain statistical methods available in Spark .
    - Correlation
    - Hypothesis Test

Training the Model -- ML Pipeline

- pyspark uses sklearn pipeline concept
- MLlib standardizes APIs for machine learning algorithms to make it easier to combine multiple algorithms into a single pipeline, or workflow.
  - **DataFrame**: This ML API uses DataFrame from Spark SQL as an ML dataset, which can hold a variety of data types. E.g., a DataFrame could have different columns storing text, feature vectors, true labels, and predictions.
  - **Transformer**: A Transformer is an algorithm which can transform one DataFrame into another DataFrame. E.g., an ML model is a Transformer which transforms a DataFrame with features into a DataFrame with predictions.
  - **Estimator**: An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer. E.g., a learning algorithm is an Estimator which trains on a DataFrame and produces a model.

Contd ...

- **Pipeline**: A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow.
- **Parameter**: All Transformers and Estimators now share a common API for specifying parameters.

Transformers

- A Transformer is an abstraction that includes feature transformers and learned models. Technically, a Transformer implements a method transform(), which converts one DataFrame into another, generally by appending one or more columns. For example:
  - A feature transformer might take a DataFrame, read a column (e.g., text), map it into a new column (e.g., feature vectors), and output a new DataFrame with the mapped column appended.
  - A learning model might take a DataFrame, read the column containing feature vectors, predict the label for each feature vector, and output a new DataFrame with predicted labels

Estimators

An Estimator abstracts the concept of a learning algorithm or any algorithm that fits or trains on data. Technically, an Estimatorimplements a method fit(), which accepts a DataFrame and produces a Model, which is a Transformer.

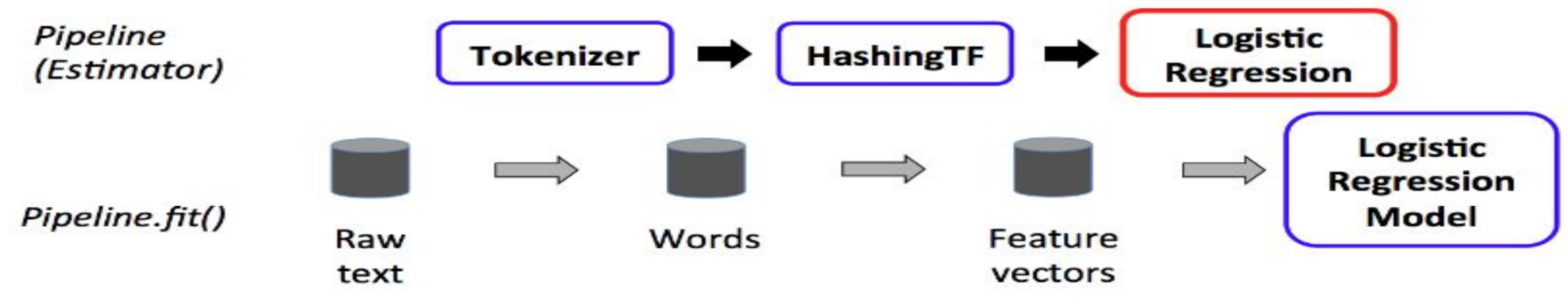
For example, a learning algorithm such as LogisticRegression is an Estimator, and calling fit() trains a LogisticRegressionModel, which is a Model and hence a Transformer.

Pipeline

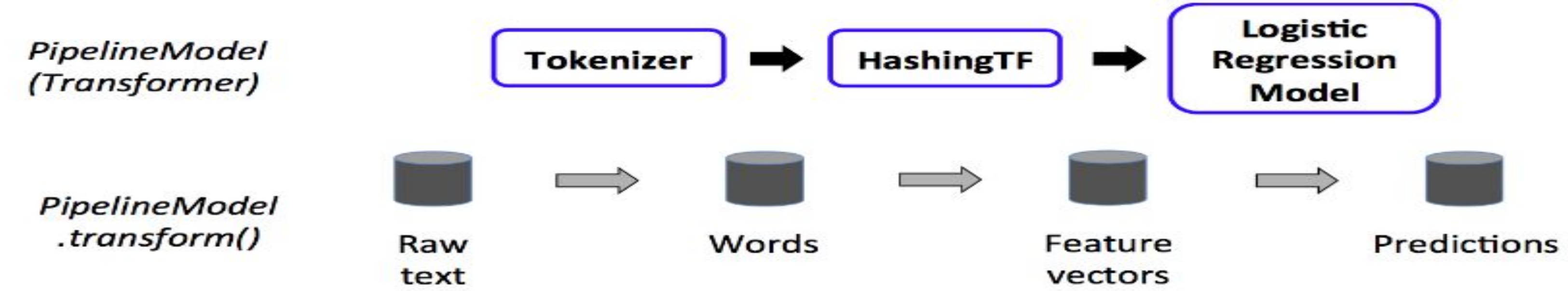
- In machine learning, it is common to run a sequence of algorithms to process and learn from data.
- E.g., a simple text document processing workflow might include several stages:
  - Split each document’s text into words.
  - Convert each document’s words into a numerical feature vector.
  - Learn a prediction model using the feature vectors and labels.

MLlib represents such a workflow as a Pipeline, which consists of a sequence of PipelineStages (Transformers and Estimators) to be run in a specific order. We will use this simple workflow as a running example in this section.

Example



A Pipeline is an Estimator. Thus, after a Pipeline's fit() method runs, it produces a PipelineModel, which is a Transformer. This PipelineModel is used at *test time*; the figure below illustrates this usage.


PipelineModel (Transformer)
Tokenizer
HashingTF
Logistic Regression Model
PipelineModel .transform()
Raw text
Words
Feature vectors
Predictions

Evaluating the Model

In pyspark.ml.evaluation there are different evaluators for the models like

- **BinaryClassificationEvaluator**
- **RegressionEvaluator**
- **MulticlassClassificationEvaluator**
- **ClusteringEvaluator**

Testing Model -- Load/Save -- Persist Pipeline

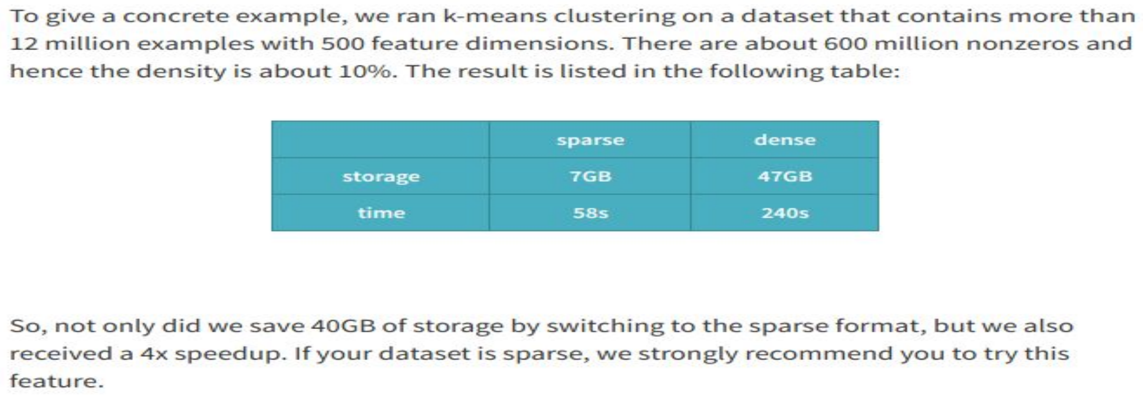
- The pipeline Model that has been created in the previous steps can be saved Whenever the new data comes we need to load the model and get the predictions.

```
>>>pipeline_model.save("Path to the location to save")
>>>model = pipeline_model.load("path where the model is")
```

SparkML - use Sparse Vectors

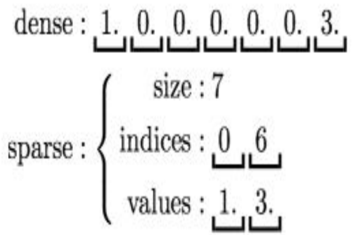
- For performing the machine learning algorithms sparkml uses the Sparse vector representation for the rows

- In Apache Spark 1.0, MLlib adds full support for sparse data in Scala, Java, and Python (previous versions only supported it in specific algorithms like alternating least squares). It takes advantage of sparsity in both storage and computation in methods including SVM, logistic regression, Lasso, naive Bayes, k-means, and summary statistics.


To give a concrete example, we ran k-means clustering on a dataset that contains more than 12 million examples with 500 feature dimensions. There are about 600 million nonzeros and hence the density is about 10%. The result is listed in the following table:

	sparse	dense
storage	7GB	47GB
time	58s	240s

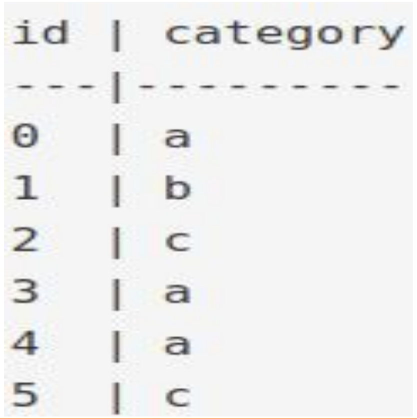
So, not only did we save 40GB of storage by switching to the sparse format, but we also received a 4x speedup. If your dataset is sparse, we strongly recommend you to try this feature.



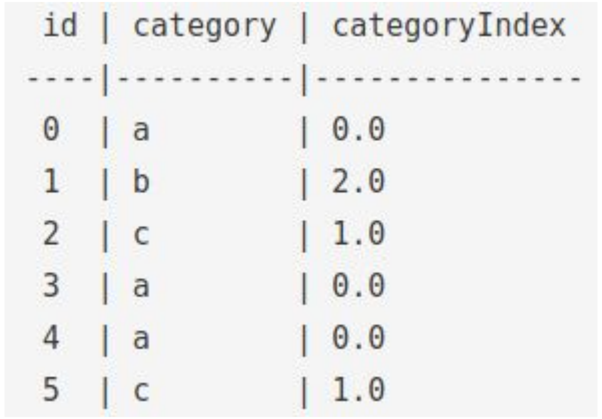
Important Preprocessing Steps

String Indexer :

A label indexer that maps a string column of labels to an ML column of label indices. If the input column is numeric, we cast it to string and index the string values. The indices are in [0, numLabels). By default, this is ordered by label frequencies so the most frequent label gets index 0. The ordering behavior is controlled by setting **stringOrderType**. Its default value is ‘frequencyDesc’.



```
id | category
---|----------
0  | a
1  | b
2  | c
3  | a
4  | a
5  | c
```



```
 id | category | categoryIndex
----|----------|--------------
 0  | a        | 0.0
 1  | b        | 2.0
 2  | c        | 1.0
 3  | a        | 0.0
 4  | a        | 0.0
 5  | c        | 1.0
```

Contd ..

VectorAssembler

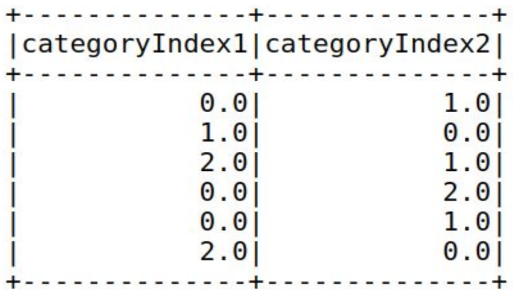
VectorAssembler is a transformer that combines a given list of columns into a single vector column. It is useful for combining raw features and features generated by different feature transformers into a single feature vector, in order to train ML models like logistic regression and decision trees. VectorAssembler accepts the following input column types: all numeric types, boolean type, and vector type. In each row, the values of the input columns will be concatenated into a vector in the specified order.

```
 id | hour | mobile | userFeatures     | clicked
----|------|--------|------------------|---------
 0  | 18   | 1.0    | [0.0, 10.0, 0.5] | 1.0
```

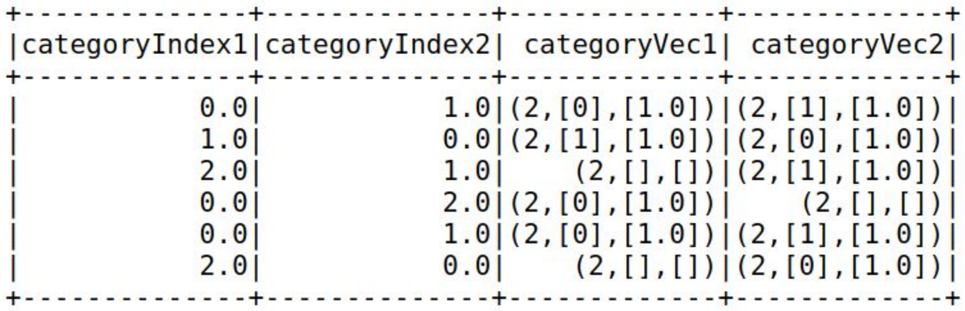
```
 id | hour | mobile | userFeatures     | clicked | features
----|------|--------|------------------|---------|-----------------------------
 0  | 18   | 1.0    | [0.0, 10.0, 0.5] | 1.0     | [18.0, 1.0, 0.0, 10.0, 0.5]
```

Contd..

OneHotEncoderEstimator: One-hot encoding maps a categorical feature, represented as a label index, to a binary vector with at most a single one-value indicating the presence of a specific feature value from among the set of all feature values. This encoding allows algorithms which expect continuous features, such as Logistic Regression, to use categorical features. For string type input data, it is common to encode categorical features using StringIndexer first.OneHotEncoderEstimator supports the handleInvalid parameter to choose how to handle invalid input during transforming data. Available options include ‘keep’ (any invalid inputs are assigned to an extra categorical index) and ‘error’ (throw an error).



```
+--------------+--------------+
|categoryIndex1|categoryIndex2|
+--------------+--------------+
|           0.0|           1.0|
|           1.0|           0.0|
|           2.0|           1.0|
|           0.0|           2.0|
|           0.0|           1.0|
|           2.0|           0.0|
+--------------+--------------+
```



```
+--------------+--------------+-------------+-------------+
|categoryIndex1|categoryIndex2| categoryVec1| categoryVec2|
+--------------+--------------+-------------+-------------+
|           0.0|           1.0|(2,[0],[1.0])|(2,[1],[1.0])|
|           1.0|           0.0|(2,[1],[1.0])|(2,[0],[1.0])|
|           2.0|           1.0|    (2,[],[])|(2,[1],[1.0])|
|           0.0|           2.0|(2,[0],[1.0])|    (2,[],[])|
|           0.0|           1.0|(2,[0],[1.0])|(2,[1],[1.0])|
|           2.0|           0.0|    (2,[],[])|(2,[0],[1.0])|
+--------------+--------------+-------------+-------------+
```

Thank You

Thank You