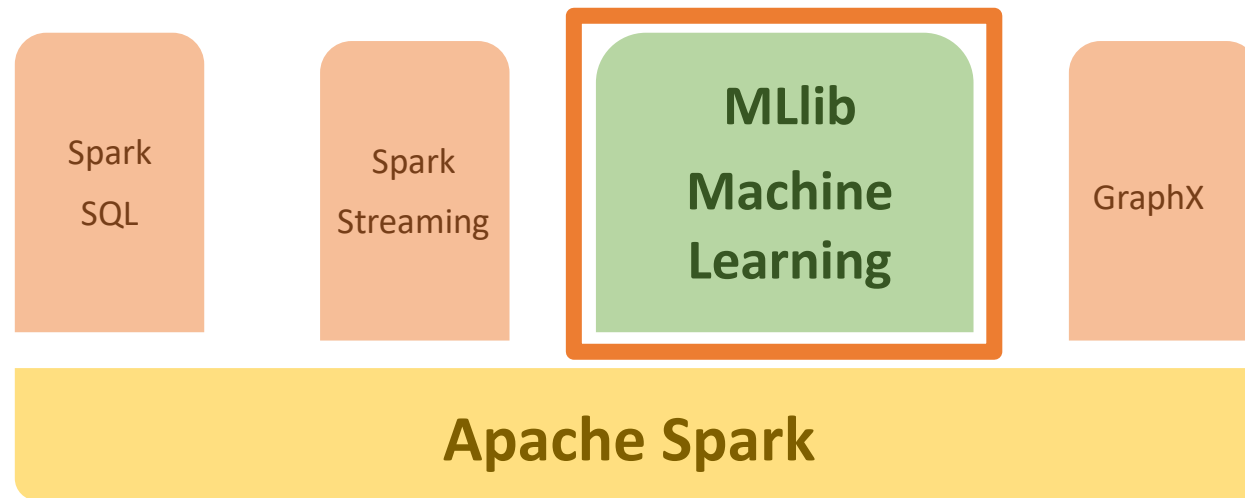


Spark MLlib for Scalable Machine Learning



Spark MLlib



Apache Spark comes with a library named **MLlib** to perform Machine Learning tasks using the Spark framework.

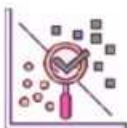


Spark MLlib Tools



ML Algorithms

classification, regression, clustering, and collaborative filtering



Featurization

feature extraction, transformation, dimensionality reduction, and selection



Pipelines

tools for constructing, evaluating, and tuning ML pipelines



Persistence

saving and loading algorithms, models and pipelines



Utilities

linear algebra, statistics, data handling

MLlib: Supported Algorithms

- **Data types**
- **Basic statistics**
 - Summary Statistics
 - Correlations
 - Stratified Sampling
 - Hypothesis Testing
 - Random Data Generation
- **Classification and regression**
 - **Linear Models (SVMs, logistic regression, linear regression)**
 - **Naive Bayes**
 - **Decision Trees**
 - **Ensembles of Trees** (Random Forests and Gradient - Boosted Trees)
- **Collaborative filtering**
 - Alternating Least Squares (ALS)
- **Clustering**
 - k-Means
 - Gaussian Mixture
 - Power Iteration
- **Dimensionality reduction**
 - Singular Value Decomposition (SVD)
 - Principal Component Analysis (PCA)
- **Feature extraction and transformation**
- **Optimization (developer)**
 - Stochastic Gradient Descent
 - Limited-Memory BFGS (L-BFGS)

MLlib: Basic Statistics, Summary statistics

MLlib in spark provide summary statistics for RDD vector through the function `colStats()` in Statistics package

```
# Install PySpark in Colab
!pip install pyspark

# Import necessary libraries
from pyspark import SparkConf, SparkContext
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.stat import Statistics

# Create a SparkConf and SparkContext
conf = SparkConf().setAppName("SummaryStatisticsExample")
sc = SparkContext(conf=conf)
```



```
# Create an RDD of vectors
data = [
    Vectors.dense([1.0, 2.0, 3.0]),
    Vectors.dense([4.0, 5.0, 6.0]),
    Vectors.dense([7.0, 8.0, 9.0])
]

rdd = sc.parallelize(data)

# Calculate summary statistics using colStats()
summary = Statistics.colStats(rdd)

# Print the summary statistics
print("Mean:", summary.mean()) # Mean of each column
print("Variance:", summary.variance()) # Variance of each column
print("Non-zero entries:", summary.numNonzeros()) # Number of non-zero entries in each column

# Stop the SparkContext
sc.stop()
```



```
Mean: [4. 5. 6.]
Variance: [9. 9. 9.]
Non-zero entries: [3. 3. 3.]
```

MLlib: Basic Statistics, Correlations

MLlib can perform **correlation** between two series of data using the statistics package

```
# Install PySpark in Colab (if not already installed)
!pip install pyspark

# Import necessary libraries
from pyspark.sql import SparkSession
from pyspark.mllib.stat import Statistics

# Create a Spark session (which includes a SparkContext)
spark = SparkSession.builder.appName("CorrelationExample").getOrCreate()

# Create two RDDs representing two series of data
data1 = [1.0, 2.0, 3.0, 4.0, 5.0]
data2 = [2.0, 3.0, 4.0, 5.0, 6.0]

rdd1 = spark.sparkContext.parallelize(data1)
rdd2 = spark.sparkContext.parallelize(data2)

# Zip the two RDDs together into pairs
paired_rdd = rdd1.zip(rdd2)

# Calculate the correlation between the two series of data
correlation = Statistics.corr(paired_rdd, method="pearson")

# Print the correlation coefficient
print("Pearson Correlation Coefficient:", correlation)
```



Pearson Correlation Coefficient: $\begin{bmatrix} 1. & 1. \\ 1. & 1. \end{bmatrix}$

MLlib: Basic Statistics, Random data generation

Random data generation in MLlib spark is used for randomized algorithms, prototyping, and performance testing

```
# Install PySpark in Colab (if not already installed)
!pip install pyspark

# Import necessary libraries
from pyspark.sql import SparkSession
from pyspark.mllib.random import RandomRDDs

# Create a Spark session (which includes a SparkContext)
spark = SparkSession.builder.appName("RandomDataExample").getOrCreate()

# Generate random data using RandomRDDs
num_samples = 1000 # Number of random data points to generate
random_data = RandomRDDs.normalRDD(spark.sparkContext, num_samples)

# Show a sample of the generated random data
sample_data = random_data.take(3) # Take the first 10 data points as a sample
print("Sample of Random Data:")
for value in sample_data:
    print(value)

# Stop the SparkSession (which will also stop the underlying SparkContext)
spark.stop()
```



Sample of Random Data:

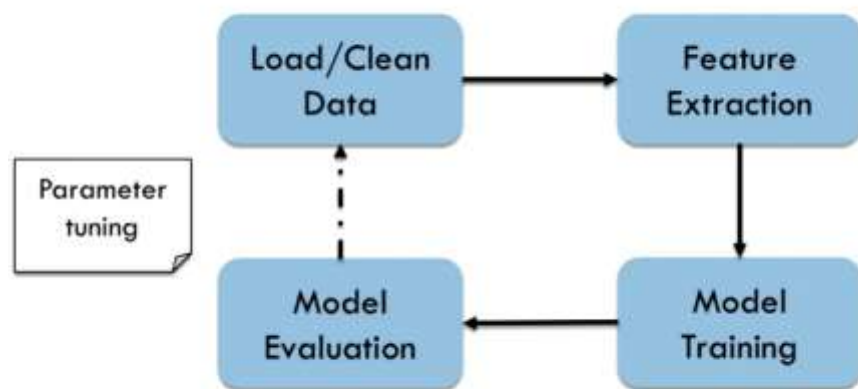
```
-1.4055953972015585
1.1191225288126616
1.3541200242279714
```

Spark ML Concepts

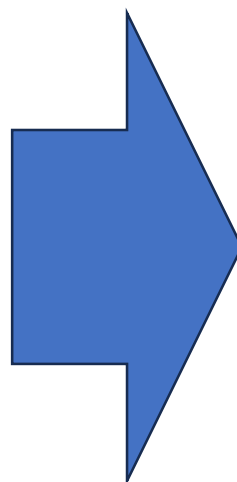
Concept	Explanation
DataFrame	Spark ML uses DataFrame from Spark SQL as an ML dataset, which can hold a variety of data types. E.g., a DataFrame could have different columns storing text, feature vectors, true labels, and predictions
Transformer	A Transformer is an algorithm which can transform one DataFrame into another DataFrame. E.g., an ML model is a Transformer which transforms 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
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

MLlib Pipeline Concepts

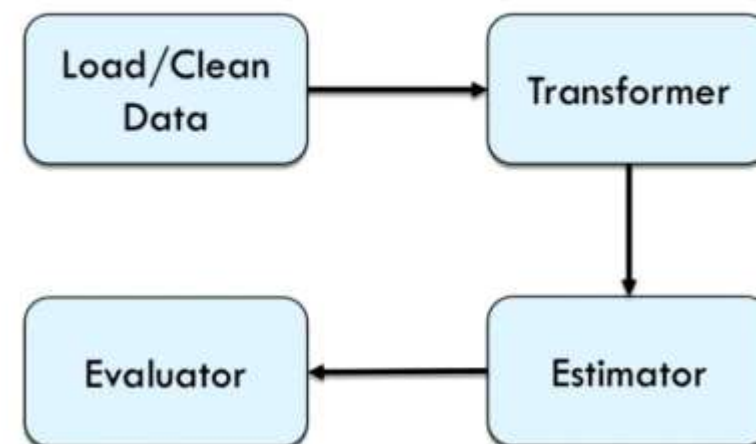
Machine Learning Pipeline



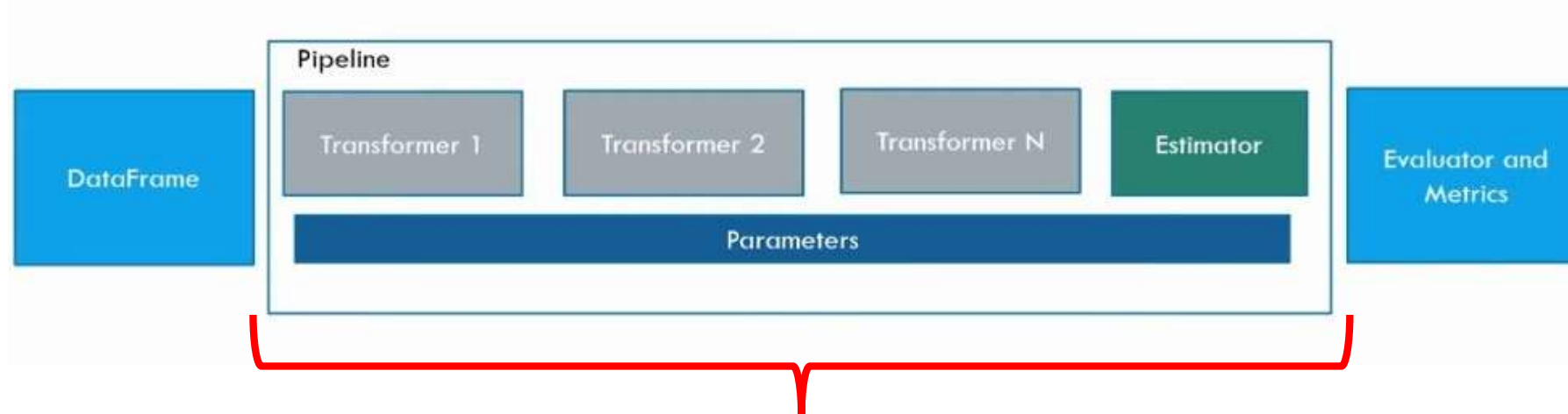
Repeatable Workflow Process



Mllib Pipeline Concepts



MLlib Pipeline



Pipeline is combination of Transformers and one Estimator

The input of a transformer is a dataframe and the output of the transformer is a dataframe.
The input of the estimator is a dataframe and the output of the dataframe is a model.



Transformer and estimator

TRANSFORMER

- Feature Transformers
 - Tokenizer
 - StopWordsRemover
 - n -gram
 - Binarizer
 - PCA
 - PolynomialExpansion
 - Discrete Cosine Transform (DCT)
 - StringIndexer
 - IndexToString
 - OneHotEncoder (Deprecated since 2.3.0)
 - OneHotEncoderEstimator
 - VectorIndexer
 - Interaction
 - Normalizer
 - StandardScaler
 - MinMaxScaler
 - MaxAbsScaler
 - Bucketizer
 - ElementwiseProduct
 - SQLTransformer
 - VectorAssembler
 - VectorSizeHint
 - QuantileDiscretizer
 - Imputer

ESTIMATOR

- Classification
 - Logistic regression
 - Binomial logistic regression
 - Multinomial logistic regression
 - Decision tree classifier
 - Random forest classifier
 - Gradient-boosted tree classifier
 - Multilayer perceptron classifier
 - Linear Support Vector Machine
 - One-vs-Rest classifier (a.k.a. One-vs-All)
 - Naive Bayes
- Regression
 - Linear regression
 - Generalized linear regression
 - Available families
 - Decision tree regression
 - Random forest regression
 - Gradient-boosted tree regression
 - Survival regression
 - Isotonic regression
- Linear methods

Machine Learning Project with PySpark MLlib

(First project: without pipeline, Second project: with Mlib pipeline)

Contents of implementation (without pipeline)

- Setting up Spark Environment
- Data loading
- Data Preprocessing using SparkML
- Model Training and Testing using SparkML
- Prediction on Test data
- Evaluation of predictions

Setting up the PySpark environment

```
# Setting up the PySpark environment

# Download Java Virtual Machine (JVM)
!apt-get install openjdk-8-jdk-headless -qq > /dev/null

# Replace 'path_to_spark_archive' with the actual path to your uploaded Spark archive
path_to_spark_archive = '/content/drive/MyDrive/big data course/spark-3.4.1-bin-hadoop3.tgz'

# Unzip the spark file
!tar xf "{path_to_spark_archive}"

## Add environmental variables
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = '/content/spark-3.2.1-bin-hadoop3.2'

# Install library for finding Spark
!pip install -q findspark

# Import the library
import findspark

# Replace 'path_to_spark' with the actual path to your Spark installation directory
path_to_spark = '/content/spark-3.4.1-bin-hadoop3'

# Initiate findspark with the correct path
findspark.init(path_to_spark)

# Check the location for Spark
findspark.find()
```

Initialize SparkSession

```
# Import SparkSession
from pyspark.sql import SparkSession

# Create a Spark Session
spark = SparkSession.builder \
    .master("local") \
    .appName("Titanic data") \
    .getOrCreate()

# Check Spark Session Information
spark
```

SparkSession - in-memory

SparkContext

[Spark UI](#)

Version

v3.4.1

Master

local

AppName

Titanic data

Reading Data

- Dataset (Titanic): <https://www.kaggle.com/c/titanic/data>

```
df = (spark.read
      .format("csv")
      .option("header", "true")
      .load("/content/drive/MyDrive/big data course/titanic dataset/train.csv"))
```

```
df.show(5)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen ...	male	22	1	0	A/5 21171	7.25	null	S
2	1	1	Cumings, Mrs. Joh...	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. ...	female	26	0	0	STON/O2. 3101282	7.925	null	S
4	1	1	Futrelle, Mrs. Ja...	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. Willia...	male	35	0	0	373450	8.05	null	S

only showing top 5 rows

Selecting some columns (if needed)

- From `pyspark.sql.functions` import `col` and then select columns

```
from pyspark.sql.functions import col
```

```
dataset = df.select(col('Survived').cast('float'),  
                    col('Pclass').cast('float'),  
                    col('Sex'),  
                    col('Age').cast('float'),  
                    col('Fare').cast('float'),  
                    col('Embarked')  
                    )
```

```
dataset.show(4)
```

Survived	Pclass	Sex	Age	Fare	Embarked
0.0	3.0	male	22.0	7.25	S
1.0	1.0	female	38.0	71.2833	C
1.0	3.0	female	26.0	7.925	S
1.0	1.0	female	35.0	53.1	S

only showing top 4 rows

Removing null values (if needed)

```
from pyspark.sql.functions import isnull, when, count, col
```


```
dataset.select([count(when(isnull(c), c)).alias(c) for c in dataset.columns]).show()
```

```
+-----+-----+-----+-----+-----+-----+
|Survived|Pclass|Sex|Age|Fare|Embarked|
+-----+-----+-----+-----+-----+-----+
|      0|      0|  0|177|    0|        2|
+-----+-----+-----+-----+-----+-----+
```

```
dataset = dataset.replace('?', None)\
    .dropna(how='any')
```

```
dataset.select([count(when(isnull(c), c)).alias(c) for c in dataset.columns]).show()
```

```
+-----+-----+-----+-----+-----+-----+
|Survived|Pclass|Sex|Age|Fare|Embarked|
+-----+-----+-----+-----+-----+-----+
|      0|      0|  0|  0|    0|        0|
+-----+-----+-----+-----+-----+-----+
```



Converting categorical variables to numeric values

- Spark only supports numeric values and is incapable of handling categorical variables. For modeling, all categorical variables must be converted to numeric values. To achieve this, **StringIndexer** is employed.

1

```
dataset.show(3)
```

Should be converted

Survived	Pclass	Sex	Age	Fare	Embarked
0.0	3.0	male	22.0	7.25	S
1.0	1.0	female	38.0	71.2833	C
1.0	3.0	female	26.0	7.925	S

only showing top 3 rows

4

```
dataset.show(2)
```

Survived	Pclass	Sex	Age	Fare	Embarked	Gender	Boarded
0.0	3.0	male	22.0	7.25	S	0.0	0.0
1.0	1.0	female	38.0	71.2833	C	1.0	1.0

only showing top 2 rows

2

```
from pyspark.ml.feature import StringIndexer
```

3

```
dataset = StringIndexer(  
    inputCol='Sex',  
    outputCol='Gender',  
    handleInvalid='keep').fit(dataset).transform(dataset)
```

```
dataset = StringIndexer(  
    inputCol='Embarked',  
    outputCol='Boarded',  
    handleInvalid='keep').fit(dataset).transform(dataset)
```

Finally, we can drop 'sex' and 'embarked' columns

5

```
#Drop unnecessary columns  
dataset = dataset.drop('Sex')  
dataset = dataset.drop('Embarked')
```

Feature engineering

- Spark learns through two columns, **label** and **feature**. Therefore, all columns except the target column must be combined into a single column. This is accomplished via **VecotrAssembler**.

```
#Assemble all the feautures with VectorAssembler
from pyspark.ml.feature import VectorAssembler
```

```
require_featured = ['Pclass', 'Age', 'Fare', 'Gender', 'Boarded']
assembler = VectorAssembler(inputCols=require_featured, outputCol='features')
transformed_data = assembler.transform(dataset)
```

```
transformed_data.show(5)
```

Label
(target variables)

Survived	Pclass	Age	Fare	Gender	Boarded	features
0.0	3.0	22.0	7.25	0.0	0.0	[3.0,22.0,7.25,0....]
1.0	1.0	38.0	71.2833	1.0	1.0	[1.0,38.0,71.2833...]
1.0	3.0	26.0	7.925	1.0	0.0	[3.0,26.0,7.92500...]
1.0	1.0	35.0	53.1	1.0	0.0	[1.0,35.0,53.0999...]
0.0	3.0	35.0	8.05	0.0	0.0	[3.0,35.0,8.05000...]

only showing top 5 rows

feature

Modeling

```
#splitting dataset into train and test
(training_data, test_data) = transformed_data.randomSplit([0.8,0.2])
print("Number of train samples: " + str(training_data.count()))
print("Number of train samples: " + str(test_data.count()))
```

Number of train samples: 572

Number of train samples: 140

```
from pyspark.ml.classification import RandomForestClassifier
```

```
rf = RandomForestClassifier(labelCol='Survived',
                           featuresCol='features',
                           maxDepth=5)
```

```
model = rf.fit(training_data)
```

```
predictions = model.transform(test_data)
```

Evaluation

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
evaluator = MulticlassClassificationEvaluator(  
    labelCol='Survived',  
    predictionCol='prediction',  
    metricName = 'accuracy')
```

```
accuracy = evaluator.evaluate(predictions_training)  
print('Training Accuracy = ', accuracy)
```

Training Accuracy = 0.8444055944055944

```
accuracy = evaluator.evaluate(predictions_test)  
print('Test Accuracy = ', accuracy)
```

Test Accuracy = 0.8928571428571429

```
evaluator2 = MulticlassClassificationEvaluator(labelCol="Survived")  
area_under_curve_training = evaluator2.evaluate(predictions_training)  
  
print("area_under_curve: " , area_under_curve_training)
```

area_under_curve: 0.8412953711334141

```
evaluator3 = MulticlassClassificationEvaluator(labelCol="Survived")  
area_under_curve_test = evaluator3.evaluate(predictions_test)  
  
print("area_under_curve: " , area_under_curve_test)
```

area_under_curve: 0.8913308913308915

Second project (using MLlib Pipeline)

Contents:

1. Setting up the environment
2. Read data
3. Leverage Spark ML pipeline

Setting up the environment

```
# Setting up the PySpark environment

# Download Java Virtual Machine (JVM)
!apt-get install openjdk-8-jdk-headless -qq > /dev/null

# Replace 'path_to_spark_archive' with the actual path to your uploaded Spark archive
path_to_spark_archive = '/content/drive/MyDrive/big data course/spark-3.4.1-bin-hadoop3.tgz'

# Unzip the spark file
!tar xf "{path_to_spark_archive}"

## Add environmental variables
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = '/content/spark-3.2.1-bin-hadoop3.2'

# Install library for finding Spark
!pip install -q findspark

# Import the library
import findspark

# Replace 'path_to_spark' with the actual path to your Spark installation directory
path_to_spark = '/content/spark-3.4.1-bin-hadoop3'

# Initiate findspark with the correct path
findspark.init(path_to_spark)

# Check the location for Spark
findspark.find()
```

Initialize SparkSession

```
# Import SparkSession
from pyspark.sql import SparkSession

# Create a Spark Session
spark = SparkSession.builder \
    .master("local") \
    .appName("Titanic data") \
    .getOrCreate()

# Check Spark Session Information
spark
```

SparkSession - in-memory

SparkContext

[Spark UI](#)

Version

v3.4.1

Master

local

AppName

Titanic data

Reading data

```
df = (spark.read
      .format("csv")
      .option("header", "true")
      .load("/content/drive/MyDrive/big data course/titanic dataset/train.csv"))
```

```
df.show(3)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen ...	male	22	1	0	A/5 21171	7.25	null	S
2	1	1	Cumings, Mrs. Joh...	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. ...	female	26	0	0	STON/O2. 3101282	7.925	null	S

only showing top 3 rows

Importing functions

```
from pyspark.sql import functions as F
from pyspark.sql import types as T

# StringIndexer is similar to labelencoder which gives a label to each category
# OneHotEncoder created onehot encoding vector
from pyspark.ml.feature import StringIndexer, OneHotEncoder

# VectorAssembler is used to create vector from the features. Modeling takes vector as an input
from pyspark.ml.feature import VectorAssembler

# DecisionTreeClassifier is used for classification problems
from pyspark.ml.classification import RandomForestClassifier
```

Using pipeline

```
# Import pipeline from PySpark ML
from pyspark.ml import Pipeline
```

```
(train_df, test_df) = dataset.randomSplit([0.8, 0.2], 11)
print("Number of train samples: " + str(train_df.count()))
print("Number of test samples: " + str(test_df.count()))
```

```
Number of train samples: 562
Number of test samples: 150
```

```
# Label Encoding of categorical variables without any .fit or .transform
Sex_indexer = StringIndexer(inputCol="Sex", outputCol="Gender")
Embarked_indexer = StringIndexer(inputCol="Embarked", outputCol="Boarded")

# Assemble all the features with VectorAssembler
inputCols = ['Pclass', 'Age', 'Fare', 'Gender', 'Boarded']
outputCol = "features"
vector_assembler = VectorAssembler(inputCols = inputCols, outputCol = outputCol)

# Modeling using DecisionTreeClassifier
dt_model = RandomForestClassifier(labelCol="Survived", featuresCol="features")
```

Using pipeline

Transformers


Estimator

```
# Setup the pipeline
pipeline = Pipeline(stages=[Sex_indexer, Embarked_indexer, vector_assembler, dt_model])

# Fit the pipeline model
final_pipeline = pipeline.fit(train_df)

# Predict on test data
test_predictions_from_pipeline = final_pipeline.transform(test_df)

test_predictions_from_pipeline.show(5, truncate=False)
```



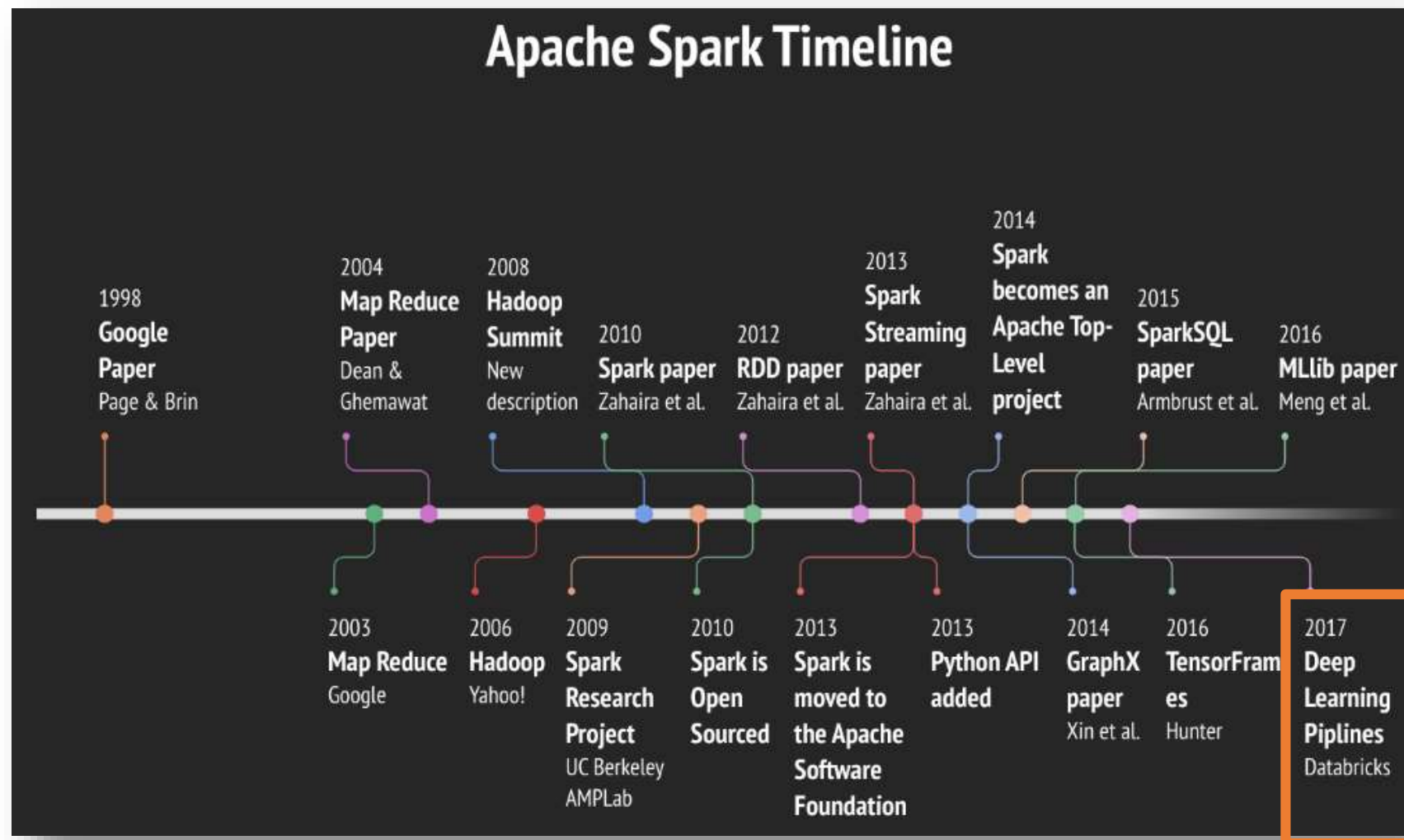
Deep learning with PySpark Mllib

Deep Learning Pipelines for Apache Spark

Deep Learning Pipelines is a new library published by Databricks to provide deep learning models and transfer learning via integration of popular deep learning libraries with MLib Pipelines and Spark SQL.

Deep Learning Pipelines provides a suite of tools around working with and processing images using deep learning. The tools can be categorized as:

- Working with images
- Transfer learning
- Applying deep learning models at scale



Using Spark to load images as a DataFrame

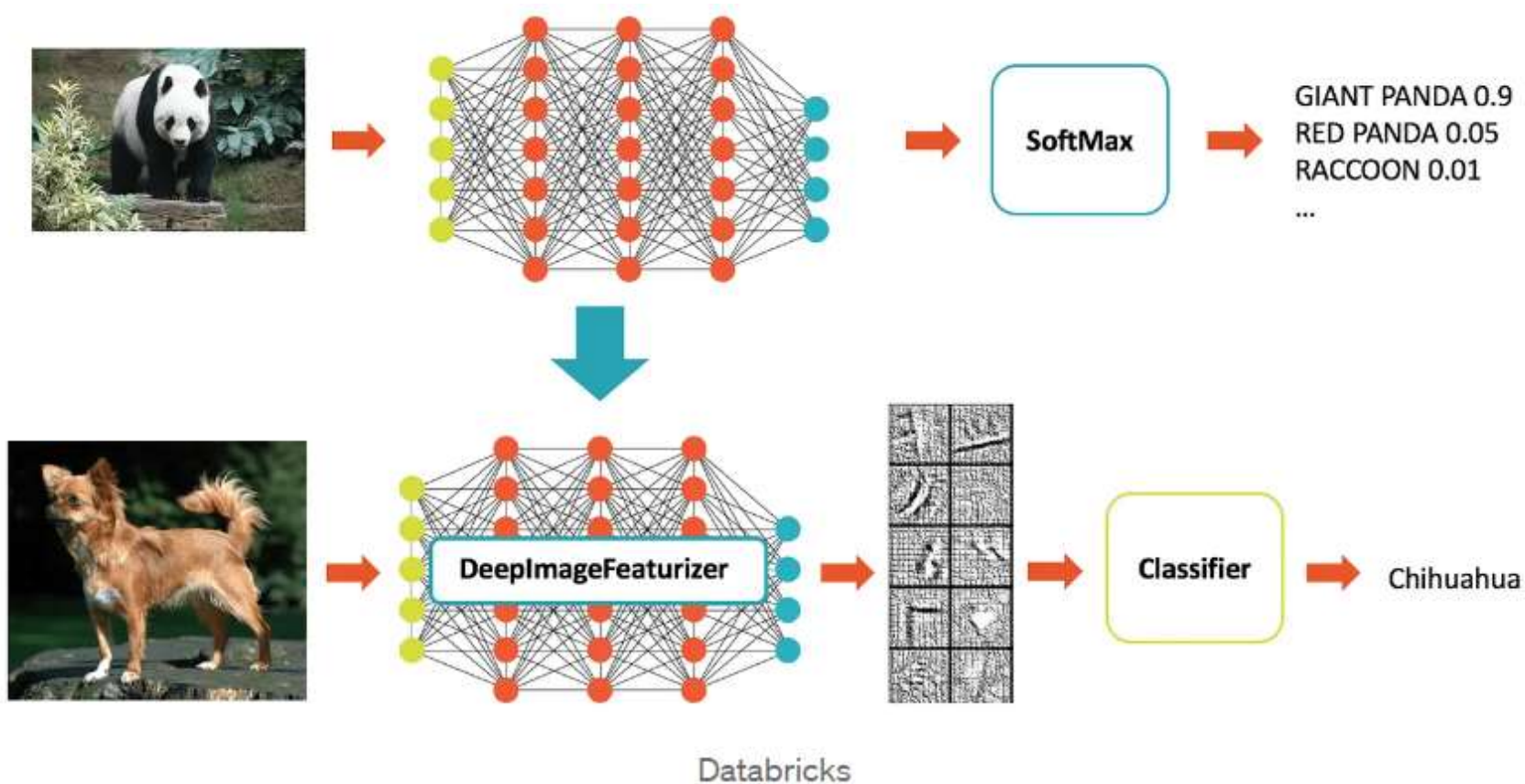
```
pip install sparkdl  
from sparkdl import readImages
```

```
[21] image_df.show()
```

```
+-----+-----+  
|          filePath|          image|  
+-----+-----+  
|file:/content/flo...|{RGB, 263, 320, 3...|  
|file:/content/flo...|{RGB, 313, 500, 3...|  
|file:/content/flo...|{RGB, 209, 320, 3...|  
+-----+-----+
```

Transfer learning

Deep Learning Pipelines enables fast transfer learning with the concept of a *Featurizer*.




Transfer learning


```
from pyspark.ml.classification import LogisticRegression
```

```
from pyspark.ml import Pipeline
```

```
from sparkdl import DeepImageFeaturizer
```



```
featurizer = DeepImageFeaturizer(inputCol="image", outputCol="features", modelName="InceptionV3")
```



```
lr = LogisticRegression(maxIter=10, regParam=0.05, elasticNetParam=0.3, labelCol="label")
```

```
p = Pipeline(stages=[featurizer, lr])
```

```
p_model = p.fit(train_df)
```

Machine Learning



References

- <https://spark.apache.org/docs/latest/ml-guide.html>
- <https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bfcf/5669198905533692/3647723071348946/3983381308530741/latest.html>
- <https://github.com/FavioVazquez/deep-learning-pyspark>
- <https://towardsdatascience.com/deep-learning-with-apache-spark-part-2-2a2938a36d35>



Deep Learning