BTS | Barcelona Technology School

Real Time Data Analysis

Master in Big Data Solutions 2020-2021

Ankit Tewari, Course Instructor

ankit.tewari@bts.tech

Today's Objective

- Understanding the purpose of using Spark's Machine Learning Framework
- Getting conceptual experience of Spark's Machine Learning Capabilities
- Perform important functions regarding data ingestion and transformation
- Creating pipelines to obtain a seamless data workflow for Distributed Machine Learning Operations

Contents

- Introduction
- The limitations of classical machine learning frameworks
- Understanding the modern analytics workflow
- Feature engineering for modern analytics operations
- Pipelining the analytics workflow operations

Motivation

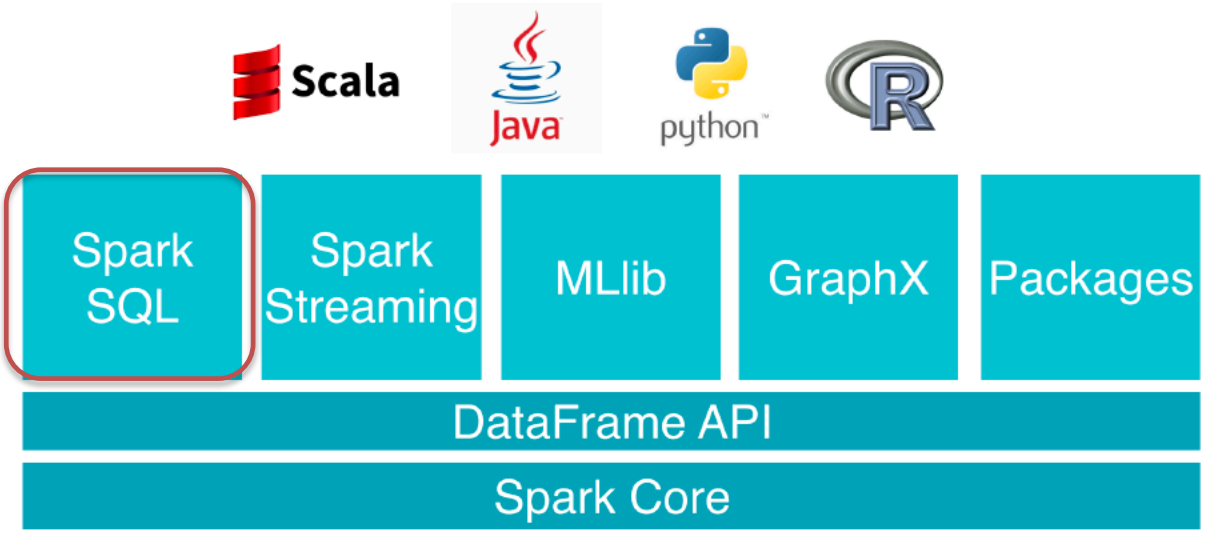
I am extremely grateful to the following for their contribution in shaping my early academic life by teaching me Apache Spark and making me familiar with the cutting edge terminologies in data processing and modelling.

- **Alexandre Perera Lluna**
  - Head, Centre de Recerca en Enginyeria Biomèdica (CREB- UPC)
  - Teaching Courses- Introduction to Bioinformatics and Data Mining for Biomedical Databases

- **Joaquim Gabarro**
  - Associate Professor, UPC (ALBCOM Research Group)
  - Research Interests: Design and Analysis of Parallel Data Structures, Parallel and Concurrent Program Verification.

Let's revisit the past...

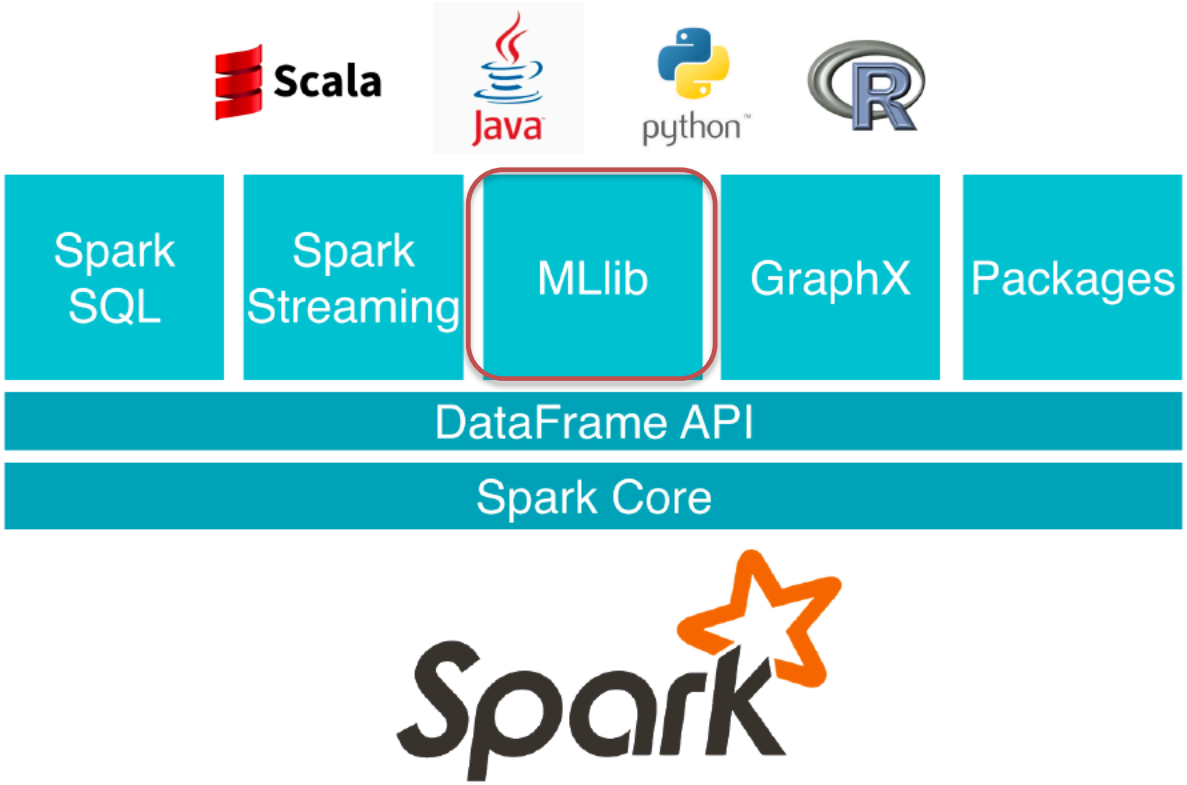
What did we learn previously?



- **Spark SQL**: How to execute SQL queries in a PySpark Dataframe.
- **Using data source**: How to use the Data Sources API to read and ingest various data sources into Spark.
- **Basic statistical computations**: How to perform basic statistical computations on the Spark Dataframe.

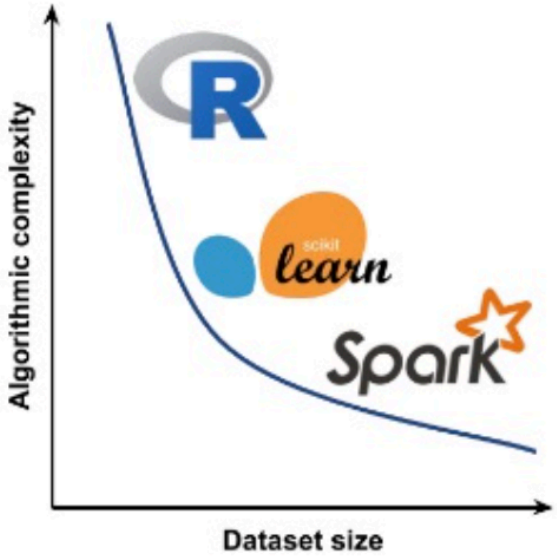
From Reading Data to Analysing Data

Remember, analytics need structured data..



Drawbacks of Conventional Analytics Solutions

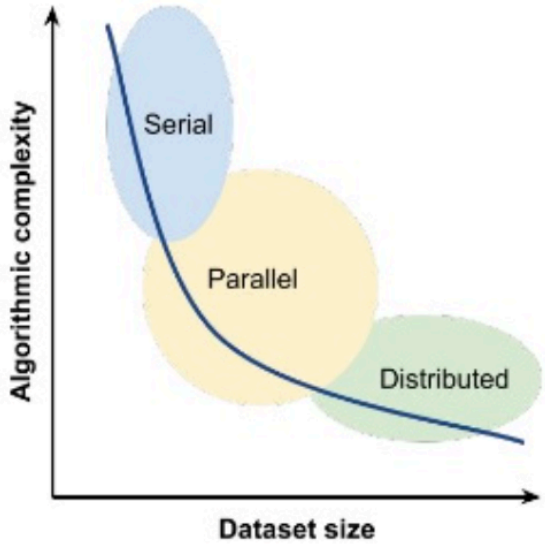
Why do we even need to think about newer solutions?

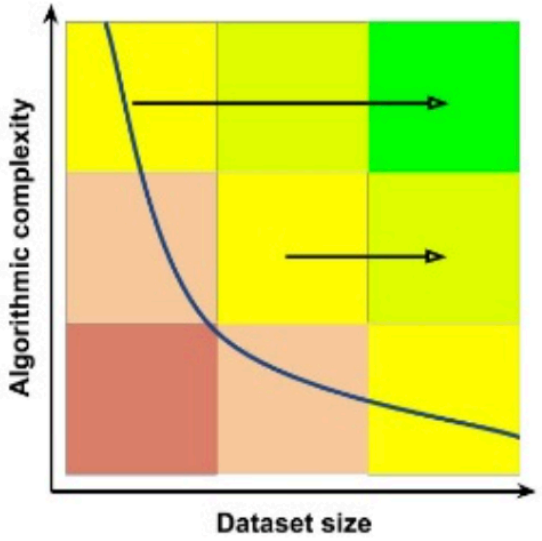


With the improvement in the methods of collection of data, especially due to automation, the size of data being used for analytics has increased enormously. In fact, most of the conventional tools being used for analysing data have turned obsolete because of the demand of large scale data analytics.

Drawbacks of Conventional Analytics Solutions

Think in the terms of backend processes behind analytics





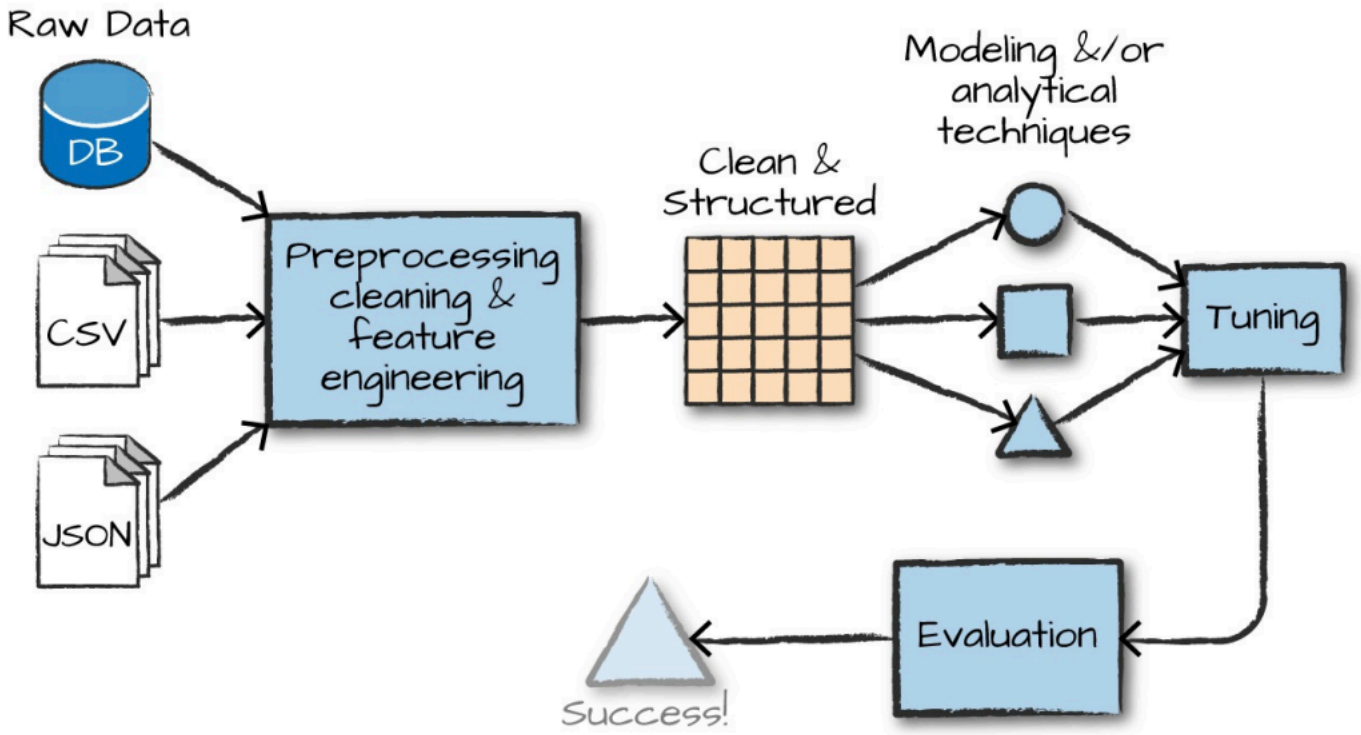
Scaling horizontally is the solution..

- **Serial computing**: sequences of instructions, executed one at a time, preferably on a single CPU.
- **Parallel computing**: multiple instructions, concurrently executed, ability to use multiple CPUs.
- **Distributed computing**: distributed memory, concurrent instructions executed, ability to use multiple CPUs.

Enough concepts. Let's see how modern analytics processes happen at scale with Big Data Solutions.

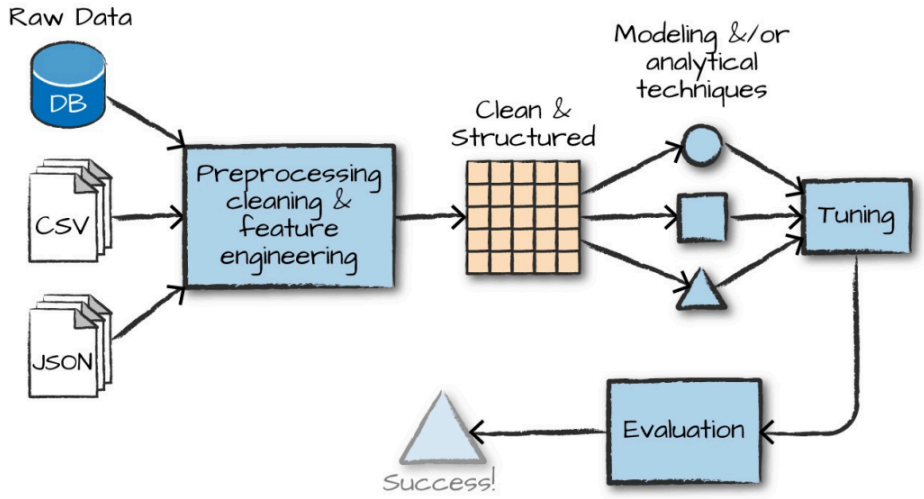
The Modern Analytics Process

Understanding the process step-by-step



The Modern Analytics Process

Understanding the process step-by-step



- **Data Collection**: Gathering and collecting the relevant data for your task based on the problem statement.
- **Data Cleaning**: Cleaning and inspecting the data to better understand it and make it free from any noise etc.
- **Feature Engineering**: to allow the algorithm to leverage the data in a suitable form (e.g., converting the data to numerical vectors)
- **Model training, tuning and evaluation**: Evaluating and comparing models against your success criteria by objectively measuring results on a subset of the same data that was not used for training. This allows you to better understand how your model may perform in the wild
- **Model deployment** : Leveraging the insights from the above process and/or using the model to make predictions, detect anomalies, or solve more general business challenges

The Modern Analytics Process

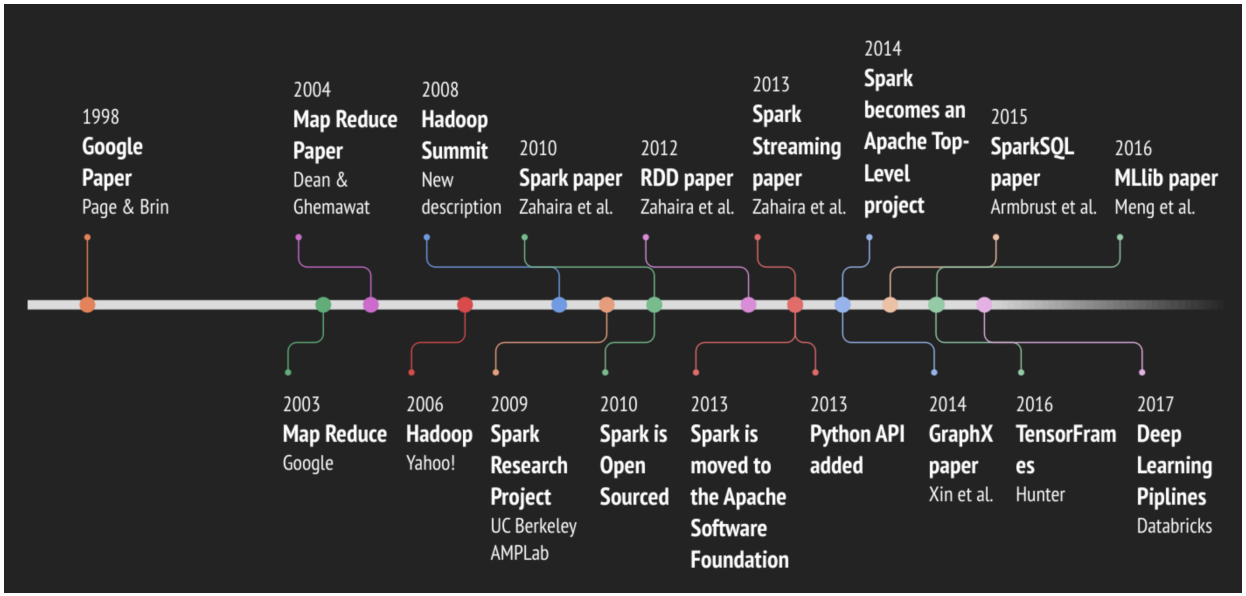
Challenges of doing Machine Learning Operations

With the increase in availability of data, Machine learning models have the ability to generalise well.
But most of the algorithms depend on the amount of data stored at a time in the RAM because by design they operate on a single machine.

Consider some constraints:

1. If the data fits well in RAM: The learning algorithm will be able to learn well but not generalise enough.
2. If the data does not fit: The learning algorithm will force the computer to start to use it's virtual memory to bring the data from Disk to RAM and back to the Disk. This retrieval process is slow. In fact, If the size of data increases, paging becomes more intense- computer keeps waiting for data- performance plummets

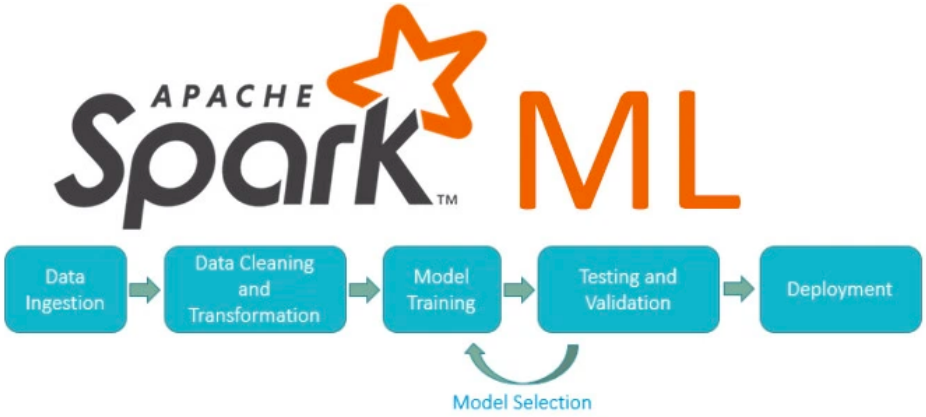
So, what is the solution to the problem?



The Modern Analytics Process

Introducing the Analytical Capabilities of Apache

Apache Spark allows distribution across multiple computers in a cluster rather than handling on a single machine. It's divided up into partitions which are processed separately. Ideally, each partition can fit into RAM on a single computer in the cluster.



WARNING

MLlib actually consists of two packages that leverage different core data structures. The package `org.apache.spark.ml` includes an interface for use with DataFrames. This package also offers a high-level interface for building machine learning pipelines that help standardize the way in which you perform the preceding steps. The lower-level package, `org.apache.spark.mllib`, includes interfaces for Spark's low-level RDD APIs. This book will focus exclusively on the DataFrame API. The RDD API is the lower-level interface, which is in maintenance mode (meaning it will only receive bug fixes, not new features) at this time. It has also been covered fairly extensively in older books on Spark and is therefore omitted here.

The Modern Analytics Process

Advantages of Apache Spark's Machine Learning Framework

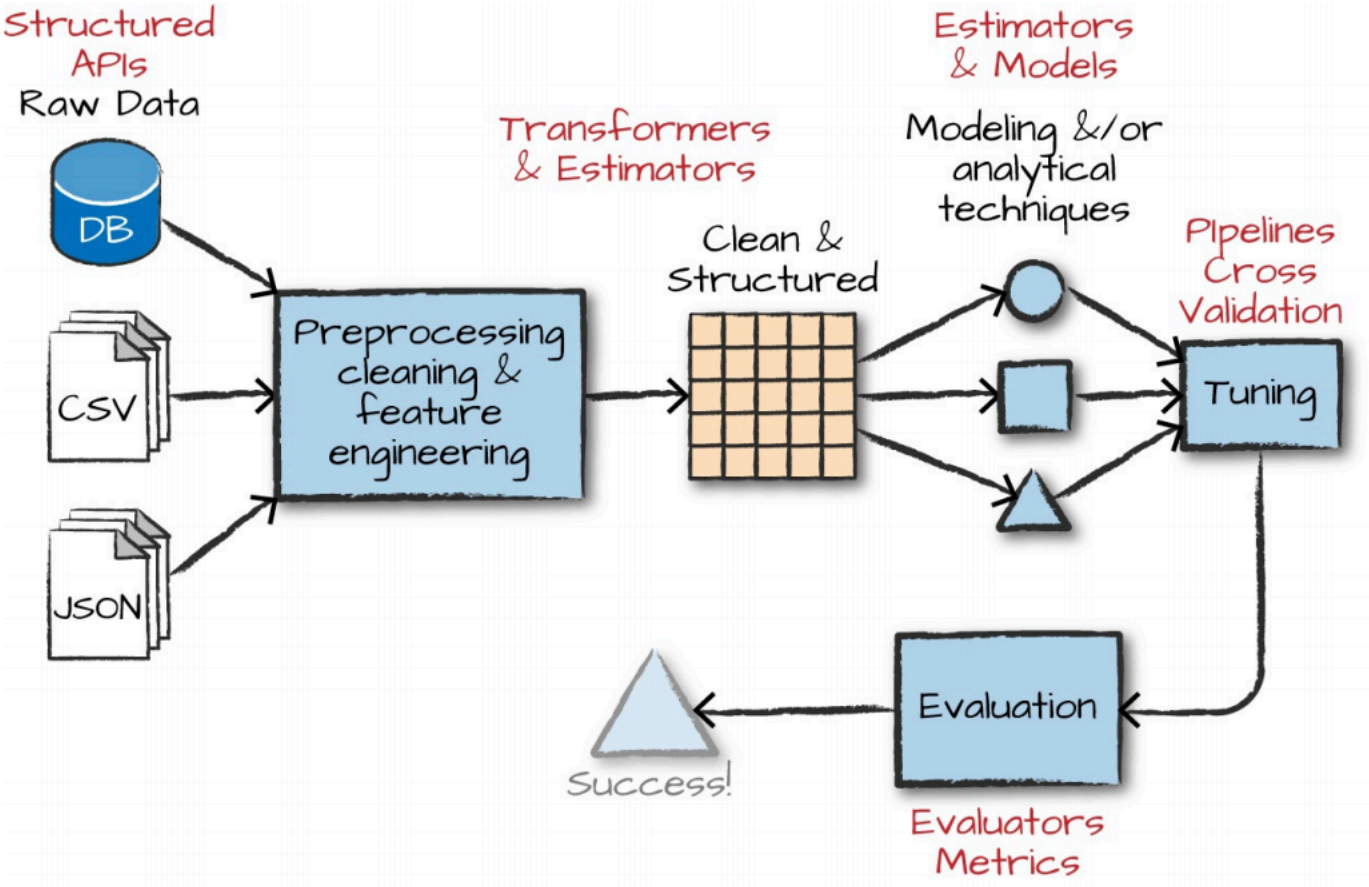
- General purpose framework for cluster computing
- Faster than other frameworks such as Hadoop- because computing happens in-memory.
- Developer friendly interface.
- Cluster itself consists of one or more nodes. Each node consists of CPU+ RAM+ Physical storage. Cluster manager- allocates resources and coordinates activity across cluster. Every application running on spark cluster has a Driver program. The driver uses Spark API to communicate with the cluster manager.
- Spark launches an executer process on each node which persists for the duration of the application.
- The executer process assigns divided up into tasks which are simply units of computation
- Executers run the task in multiple threads across the cores in a node.(Do not worry much about the details of the cluster. Spark manages it for you by itself)

There are **two key use cases where we may leverage Spark's ability to scale.**

- Firstly, you want to leverage Spark for **pre-processing and feature generation to reduce the amount of time it might take to produce training and test sets from a large amount of data**. Then, we might leverage single-machine learning algorithms to train on those given data sets.
- Secondly, **when your input data or model size become too difficult or inconvenient to put on one machine, use Spark to do the heavy lifting. Spark makes big data machine learning simple.**

The Modern Analytics Process

Components of Apache Spark's Machine Learning Framework



BTS | Barcelona Technology School

Feature Engineering

Feature Engineering

Transformers vs Estimators

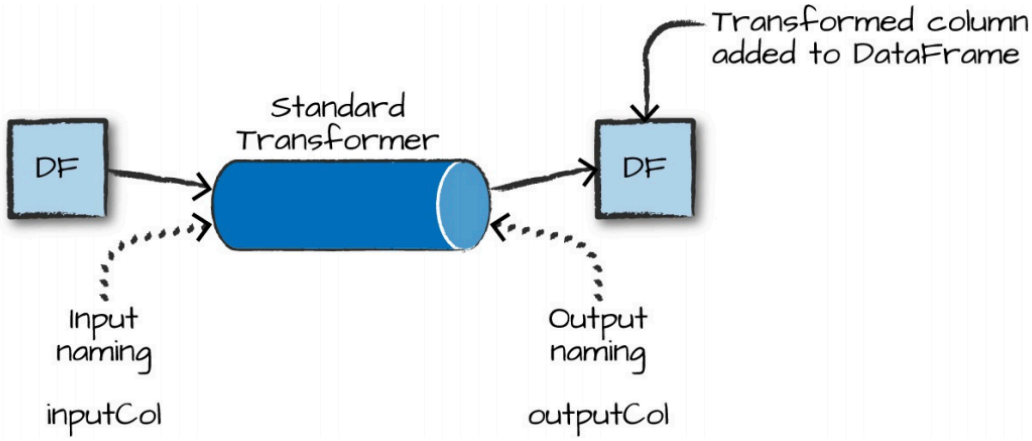
- **Transformers**: functions that accept a DataFrame as an argument and return a new DataFrame as a response after performing the requested transformation. These functions do not require the information about the input DataFrame to perform the transformation task. For example, if we would like to tokenize a sentence using a transformer, It tokenizes the string, splitting on a given character, and has nothing to learn from our data.

- **Estimators**: functions that are used when a transformation you would like to perform must be initialized with data or information about the input column (often derived by doing a pass over the input column itself). For example, if you wanted to scale the values in our column to have mean zero and unit variance, you would need to perform a pass over the entire data in order to calculate the values you would use to normalize the data to mean zero and unit variance.

Feature Engineering

Transformers

- **Transformers**: functions that accept a DataFrame as an argument and return a new DataFrame as a response after performing the requested transformation. These functions do not require the information about the input DataFrame to perform the transformation task. For example, if we would like to tokenize a sentence using a transformer, It tokenizes the string, splitting on a given character, and has nothing to learn from our data.

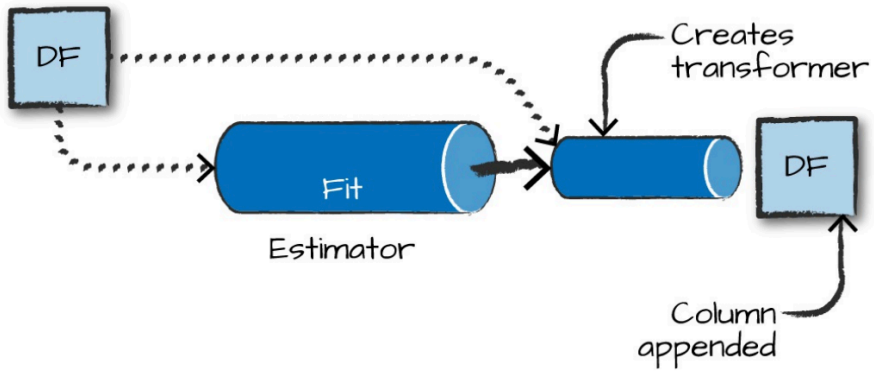


Note: Since Transformers do not need the information about the input DataDrame, they only use the transform attribute.

Feature Engineering

Estimators

- **Estimators**: functions that are used when a transformation you would like to perform must be initialized with data or information about the input column (often derived by doing a pass over the input column itself). For example, if you wanted to scale the values in our column to have mean zero and unit variance, you would need to perform a pass over the entire data in order to calculate the values you would use to normalize the data to mean zero and unit variance.



Note: Since Estimators do need the information about the input DataDrame, they use both the fit as well as the transform attribute.

Feature Transformation

High Level Transformers

High-level transformers allow you to concisely specify a number of transformations in one. These operate at a “high level”, and allow you to avoid doing data manipulations or transformations one by one.

In general, you should try to use the highest level transformers you can, in order to minimize the risk of error and help you focus on the business problem instead of the smaller details of implementation. While this is not always possible, it’s a good objective.

- **RFormula**: allows you to selects columns specified by an R Model Formula. If we use RFormula with a formula string of clicked ~ country + hour, it indicates that we want to predict clicked based on country and hour.

- **SQLTransformer**: allows you to leverage Spark’s vast library of SQL-related manipulations just as you would a MLlib transformation. You might want to use SQLTransformer if you want to formally codify some DataFrame manipulation as a pre-processing step, or try different SQL expressions for features during hyperparameter tuning.

Feature Transformation

High Level Transformers

- RFormula: allows you to selects columns specified by an R Model Formula. If we use RFormula with a formula string of clicked ~ country + hour, it indicates that we want to predict clicked based on country and hour.

- The RFormula allows you to specify your transformations in declarative syntax. It is simple to use once you understand the syntax. Currently, RFormula supports a limited subset of the R operators that in practice work quite well for simple transformations. The basic operators are:
  - **~** Separate target and terms
  - **+** Concatenate terms; “+ 0” means removing the intercept (this means the y-intercept of the line that we will fit will be 0)
  - **-** Remove a term; “- 1” means removing intercept (this means the y-intercept of the line that we will fit will be 0)
  - **:** Interaction (multiplication for numeric values, or binarized categorical values)
  - **.** All columns except the target/dependent variable

Feature Transformation

High Level Transformers

- RFormula: allows you to selects columns specified by an R Model Formula. If we use RFormula with a formula string of clicked ~ country + hour, it indicates that we want to predict clicked based on country and hour.

```
from pyspark.ml.feature import RFormula

dataset = spark.createDataFrame(
    [(7, "US", 18, 1.0),
     (8, "CA", 12, 0.0),
     (9, "NZ", 15, 0.0)],
    ["id", "country", "hour", "clicked"])

formula = RFormula(
    formula="clicked ~ country + hour",
    featuresCol="features",
    labelCol="label")

output = formula.fit(dataset).transform(dataset)
output.show()
```

Observe the formula string of clicked ~ country + hour, which indicates that we wanted to predict clicked based on country and hour

```
+---+-------+----+-------+--------------+-----+
| id|country|hour|clicked|      features|label|
+---+-------+----+-------+--------------+-----+
|  7|     US|  18|    1.0|[0.0,0.0,18.0]|  1.0|
|  8|     CA|  12|    0.0|[1.0,0.0,12.0]|  0.0|
|  9|     NZ|  15|    0.0|[0.0,1.0,15.0]|  0.0|
+---+-------+----+-------+--------------+-----+
```

.

Feature Transformation

High Level Transformers

- SQLTransformer: allows you to leverage Spark's vast library of SQL-related manipulations just as you would a MLlib transformation. You might want to use SQLTransformer if you want to formally codify some DataFrame manipulation as a pre-processing step, or try different SQL expressions for features during hyperparameter tuning.

```
sales = spark.read.csv('online-retail-dataset.csv', header= True, inferSchema= True)
sales.show(5)
```

```
+---------+---------+--------------------+--------+--------------+---------+----------+--------------+
|InvoiceNo|StockCode|         Description|Quantity|   InvoiceDate|UnitPrice|CustomerID|       Country|
+---------+---------+--------------------+--------+--------------+---------+----------+--------------+
|   536365|   85123A|WHITE HANGING HEA...|       6|12/1/2010 8:26|     2.55|     17850|United Kingdom|
|   536365|    71053| WHITE METAL LANTERN|       6|12/1/2010 8:26|     3.39|     17850|United Kingdom|
|   536365|   84406B|CREAM CUPID HEART...|       8|12/1/2010 8:26|     2.75|     17850|United Kingdom|
|   536365|   84029G|KNITTED UNION FLA...|       6|12/1/2010 8:26|     3.39|     17850|United Kingdom|
|   536365|   84029E|RED WOOLLY HOTTIE...|       6|12/1/2010 8:26|     3.39|     17850|United Kingdom|
+---------+---------+--------------------+--------+--------------+---------+----------+--------------+
only showing top 5 rows
```

```
# SQL Transformer
from pyspark.ml.feature import SQLTransformer
basicTransformation = SQLTransformer()\
.setStatement("""
SELECT sum(Quantity), count(*), CustomerID
FROM __THIS__
GROUP BY CustomerID
""")
basicTransformation.transform(sales).show(5)
```

```
+-------------+--------+----------+
|sum(Quantity)|count(1)|CustomerID|
+-------------+--------+----------+
|          265|      30|     17420|
|          133|       8|     16861|
|          567|      86|     16503|
|         3065|     302|     15727|
|         7430|     224|     17389|
+-------------+--------+----------+
only showing top 5 rows
```

Any SELECT statement you can use in SQL is a valid transformation. The only thing you need to change is that instead of using the table name, you should just use the keyword THIS.

Note:

1. The output of this transformation will be appended as a column to the output DataFrame.

Feature Assembling

Vector Assembler

The VectorAssembler is a tool you'll use in nearly every single pipeline you generate. It helps concatenate all your features into one big vector you can then pass into an estimator such as a machine learning algorithm.

It's used typically in the last step of a machine learning pipeline and takes as input a number of columns of Boolean, Double, or Vector.

This is particularly helpful if you're going to perform a number of manipulations using a variety of transformers and need to gather all of those results together

Feature Assembling

Vector Assembler

The VectorAssembler is a tool you'll use in nearly every single pipeline you generate. It helps concatenate all your features into one big vector you can then pass into an estimator.

```
df = spark.read.json("../data/airlines.json")
airport_delays = df.select("Airport.Code",
                           "Statistics.# of Delays.Security",
                           "Time.Month",
                           "Time.Year")
airport_delays.show(5)
```

```
+----+--------+-----+----+
|Code|Security|Month|Year|
+----+--------+-----+----+
| ATL|      17|    6|2003|
| BOS|       3|    6|2003|
| BWI|       8|    6|2003|
| CLT|       2|    6|2003|
| DCA|       4|    6|2003|
+----+--------+-----+----+
only showing top 5 rows
```

```
# Vector Assembler
from pyspark.ml.feature import VectorAssembler
va = VectorAssembler().setInputCols(["Month", "Year"])
va.transform(airport_delays).show(5)
```

```
+----+--------+-----+----+-----------------------------------+
|Code|Security|Month|Year|VectorAssembler_2ed8900e8557__output|
+----+--------+-----+----+-----------------------------------+
| ATL|      17|    6|2003|                       [6.0,2003.0]|
| BOS|       3|    6|2003|                       [6.0,2003.0]|
| BWI|       8|    6|2003|                       [6.0,2003.0]|
| CLT|       2|    6|2003|                       [6.0,2003.0]|
| DCA|       4|    6|2003|                       [6.0,2003.0]|
+----+--------+-----+----+-----------------------------------+
only showing top 5 rows
```

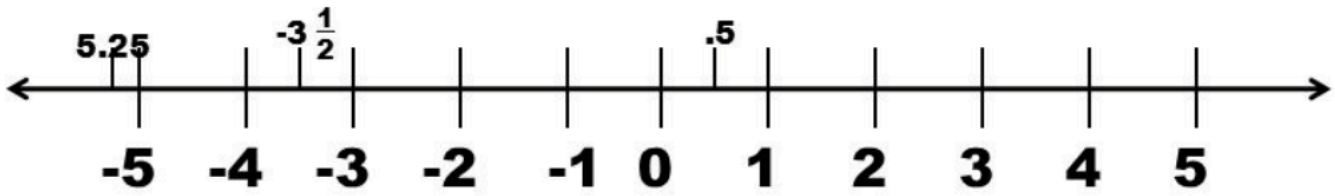
Feature Transformation

Working with Continuous Features

Continuous features are just values on the number line, from positive infinity to negative infinity. There are two common transformers for continuous features.

- First, you can convert continuous features into categorical features via a process called **bucketing**,
- Second, you can **scale and normalize** your features according to several different requirements.

These transformers will only work on Double types, so make sure you've turned any other numerical values to Double



Feature Transformation

Bucketing

Continuous features are just values on the number line, from positive infinity to negative infinity. There are two common transformers for continuous features.

- First, you can convert continuous features into categorical features via a process called **bucketing**

```
from pyspark.ml.feature import Bucketizer

splits = [-float("inf"), -0.5, 0.0, 0.5, float("inf")]
data = [(-999.9,), (-0.5,), (-0.3,), (0.0,), (0.2,), (999.9,)]
dataFrame = spark.createDataFrame(data, ["features"])
bucketizer = Bucketizer(splits=splits, inputCol="features", outputCol="bucketedFeatures")

# Transform original data into its bucket index.
bucketedData = bucketizer.transform(dataFrame)

print("Bucketizer output with %d buckets" % (len(bucketizer.getSplits())-1))
bucketedData.show()
```

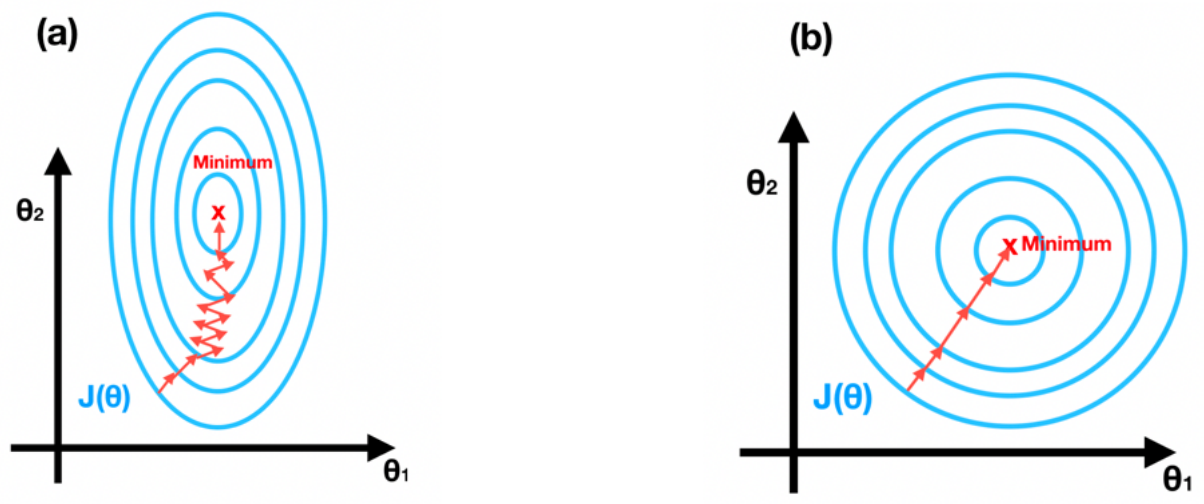
```
Bucketizer output with 4 buckets
+--------+----------------+
|features|bucketedFeatures|
+--------+----------------+
|  -999.9|             0.0|
|    -0.5|             1.0|
|    -0.3|             1.0|
|     0.0|             2.0|
|     0.2|             2.0|
|   999.9|             3.0|
+--------+----------------+
```

Feature Transformation

Scaling and Normalization

Continuous features are just values on the number line, from positive infinity to negative infinity. There are two common transformers for continuous features.

- Second, you can **scale and normalize** your features according to several different requirements.



Visual representation of convergence of a gradient descent algorithm in
a) unscaled data and b) scaled data

Read more about the need of feature scaling here
Also, the impact of various scaling and normalisation methods here

Feature Transformation

Scaling and Normalization

Continuous features are just values on the number line, from positive infinity to negative infinity. There are two common transformers for continuous features.

- Second, you can **scale and normalize** your features according to several different requirements.

```
df = spark.read.csv("diabetes.csv", header= True)
df= df.select(*(col(c).cast("float").alias(c) for c in df.columns))
va = VectorAssembler().setInputCols(['Pregnancies', 'Glucose', 'BloodPressure'])
df= va.transform(df)
vector_column= df.columns[-1]
df_scale= df.select(col(vector_column).alias("features"))
```

```
# Standard Scaler
from pyspark.ml.feature import StandardScaler
sScaler = StandardScaler().setInputCol("features")
sScaler.fit(df_scale).transform(df_scale).show(5, truncate= False)
```

```
+--------------+-----------------------------------------------------------+
|features      |StandardScaler_26be9854c13e__output                        |
+--------------+-----------------------------------------------------------+
|[6.0,148.0,72.0]|[1.7806383732194306,4.628960915766174,3.7198138711154307] |
|[1.0,85.0,66.0] |[0.29677306220323846,2.658524850271114,3.4098293818558116]|
|[8.0,183.0,64.0]|[2.3741844976259077,5.723647618818986,3.306501218769272]  |
|[1.0,89.0,66.0] |[0.29677306220323846,2.783631902048578,3.4098293818558116]|
|[0.0,137.0,40.0]|[0.0,4.284916523378148,2.0665632617307947]                 |
+--------------+-----------------------------------------------------------+
only showing top 5 rows
```

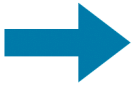
Feature Transformation

Working with Categorical Features

Categorical features are the features that can take a finite set of values. There are two common transformers for continuous features.

- First, we can perform **Indexing** which converts a categorical variable in a column to a numerical one that you can plug into machine learning algorithms. In general, it is recommended to do re-indexing of every categorical variable when pre-processing just for consistency's sake.
- Second, we can apply **One-Hot Encoding** which 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.

Color
Red
Red
Yellow
Green
Yellow



Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

Feature Transformation

StringIndexer

Categorical features are the features that can take a finite set of values. There are two common transformers for continuous features.

- First, we can perform **Indexing**: It converts a categorical variable in a column to a numerical one that you can plug into machine learning algorithms. In general, it is recommended to do re-indexing of every categorical variable when pre-processing just for consistency's sake.

```
from pyspark.ml.feature import OneHotEncoder, StringIndexer
sringIndexer = StringIndexer().setInputCol("job").setOutputCol("job_x")
bank= sringIndexer.fit(bank).transform(bank)
bank.select("job", "job_x").show(10)
```

```
+-------------+-----+
|          job|job_x|
+-------------+-----+
|   unemployed|  8.0|
|     services|  4.0|
|   management|  0.0|
|   management|  0.0|
|  blue-collar|  1.0|
|   management|  0.0|
|self-employed|  6.0|
|   technician|  2.0|
| entrepreneur|  7.0|
|     services|  4.0|
+-------------+-----+
only showing top 10 rows
```

Feature Transformation

One Hot Encoding

Categorical features are the features that can take a finite set of values. There are two common transformers for continuous features.

- Second, we can apply **One-Hot Encoding**: it 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.

```
from pyspark.ml.feature import OneHotEncoder, StringIndexer
sringIndexer = StringIndexer().setInputCol("job").setOutputCol("job_x")
bank= sringIndexer.fit(bank).transform(bank)
encoder = OneHotEncoder(inputCols=["job_x"],
                        outputCols=["job_ohe"])
model = encoder.fit(bank)
encoded = model.transform(bank)
encoded.select("job", "job_x", "job_ohe").show(10, truncate= False)
```

```
+-------------+-----+-------------+
|job          |job_x|job_ohe      |
+-------------+-----+-------------+
|unemployed   |8.0  |(11,[8],[1.0])|
|services     |4.0  |(11,[4],[1.0])|
|management   |0.0  |(11,[0],[1.0])|
|management   |0.0  |(11,[0],[1.0])|
|blue-collar  |1.0  |(11,[1],[1.0])|
|management   |0.0  |(11,[0],[1.0])|
|self-employed|6.0  |(11,[6],[1.0])|
|technician   |2.0  |(11,[2],[1.0])|
|entrepreneur |7.0  |(11,[7],[1.0])|
|services     |4.0  |(11,[4],[1.0])|
+-------------+-----+-------------+
only showing top 10 rows
```

Feature Transformation

Working with Text Features

Text input is always considered to be complex because it often requires lots of manipulation to be done before the text can be served as an input to a machine learning model. There are generally two kinds of texts you'll see: free-form text and string categorical variables. We will primarily focus on freeform text because we already discussed categorical variables. Some steps to process the free form text are-

- First, we can convert free-form text into a list of "tokens" or individual words by means of **Tokenization.**
- Second, we can remove common words that are not relevant in many kinds of analysis and should thus be removed. Frequently occurring stop words in English include "the," "and," and "but." and are called **Stopwords.**
- Third, we can look for creating some **Word combinations** of words called n-grams to better capture sentence structure and more information than can be gleaned by simply looking at all words individually.
- Fourth, we may convert the words into their numerical representations by means of an algorithm called **Term-Frequency-Inverse-Document-Frequency (TF-IDF)**. Based on this, the words that occur in a few documents are given more weight than words that occur in many documents.

Feature Transformation

Working with Text Features

Text input is always considered to be complex because it often requires lots of manipulation to be done before the text can be served as an input to a machine learning model. There are generally two kinds of texts you'll see: free-form text and string categorical variables. We will primarily focus on freeform text because we already discussed categorical variables. Some steps to process the free form text are-

- First, we can convert free-form text into a list of "tokens" or individual words by means of **Tokenization** (these tokenization are based on whitespace).

```
# Tokenization
from pyspark.ml.feature import Tokenizer
tkn = Tokenizer().setInputCol("Description").setOutputCol("Tokenized_Description")
tokenized = tkn.transform(sales.select("Description"))
tokenized.show(5, False)
```

```
+-----------------------------------+------------------------------------------+
|Description                        |Tokenized_Description                     |
+-----------------------------------+------------------------------------------+
|WHITE HANGING HEART T-LIGHT HOLDER |[white, hanging, heart, t-light, holder]  |
|WHITE METAL LANTERN                |[white, metal, lantern]                   |
|CREAM CUPID HEARTS COAT HANGER     |[cream, cupid, hearts, coat, hanger]      |
|KNITTED UNION FLAG HOT WATER BOTTLE|[knitted, union, flag, hot, water, bottle]|
|RED WOOLLY HOTTIE WHITE HEART.     |[red, woolly, hottie, white, heart.]      |
+-----------------------------------+------------------------------------------+
only showing top 5 rows
```

Note: We can also create a Tokenizer that is not just based white space but a regular expression with the RegexTokenizer. See the notebook for a detailed example.

Feature Transformation

Working with Text Features

Text input is always considered to be complex because it often requires lots of manipulation to be done before the text can be served as an input to a machine learning model. There are generally two kinds of texts you'll see: free-form text and string categorical variables. We will primarily focus on freeform text because we already discussed categorical variables. Some steps to process the free form text are-

- Second, we can remove common words that are not relevant in many kinds of analysis and should thus be removed. Frequently occurring stop words in English include "the," "and," and "but." and are called **Stopwords.**

```
# Stopwords removal
from pyspark.ml.feature import StopWordsRemover
englishStopWords = StopWordsRemover.loadDefaultStopWords("english")
stops = StopWordsRemover()\
.setStopWords(englishStopWords)\
.setInputCol("Tokenized_Description")
stops.transform(tokenized).show(5)
```

```
+--------------------+--------------------+------------------------------------+
|         Description|Tokenized_Description|StopWordsRemover_b1e9d2b661bf__output|
+--------------------+--------------------+------------------------------------+
|WHITE HANGING HEA...| [white, hanging, ...|                [white, hanging, ...|
| WHITE METAL LANTERN| [white, metal, la...|                [white, metal, la...|
|CREAM CUPID HEART...| [cream, cupid, he...|                [cream, cupid, he...|
|KNITTED UNION FLA...| [knitted, union, ...|                [knitted, union, ...|
|RED WOOLLY HOTTIE...| [red, woolly, hot...|                [red, woolly, hot...|
|SET 7 BABUSHKA NE...| [set, 7, babushka...|                [set, 7, babushka...|
|GLASS STAR FROSTE...| [glass, star, fro...|                [glass, star, fro...|
|HAND WARMER UNION...| [hand, warmer, un...|                [hand, warmer, un...|
|HAND WARMER RED P...| [hand, warmer, re...|                [hand, warmer, re...|
|ASSORTED COLOUR B...| [assorted, colour...|                [assorted, colour...|
|POPPY'S PLAYHOUSE...| [poppy's, playhou...|                [poppy's, playhou...|
|POPPY'S PLAYHOUSE...| [poppy's, playhou...|                [poppy's, playhou...|
|FELTCRAFT PRINCES...| [feltcraft, princ...|                [feltcraft, princ...|
|IVORY KNITTED MUG...| [ivory, knitted, ...|                [ivory, knitted, ...|
|BOX OF 6 ASSORTED...| [box, of, 6, asso...|                [box, 6, assorted...|
|BOX OF VINTAGE JI...| [box, of, vintage...|                [box, vintage, ji...|
|BOX OF VINTAGE AL...| [box, of, vintage...|                [box, vintage, al...|
|HOME BUILDING BLO...| [home, building, ...|                [home, building, ...|
|LOVE BUILDING BLO...| [love, building, ...|                [love, building, ...|
|RECIPE BOX WITH M...| [recipe, box, wit...|                [recipe, box, met...|
+--------------------+--------------------+------------------------------------+
only showing top 20 rows
```

Feature Transformation

Working with Text Features

Text input is always considered to be complex because it often requires lots of manipulation to be done before the text can be served as an input to a machine learning model. There are generally two kinds of texts you'll see: free-form text and string categorical variables. We will primarily focus on freeform text because we already discussed categorical variables. Some steps to process the free form text are-

- Third, we can look for creating some **Word combinations** of words called n-grams to better capture sentence structure and more information than can be gleaned by simply looking at all words individually**.** When N=2, the bi-grams of "Big Data Processing Made Simple" are- "Big Data", "Data Processing", "Processing Made", "Made Simple".

```
# Creating word combinations- NGrams
from pyspark.ml.feature import NGram
bigram = NGram().setInputCol("Tokenized_Description").setN(2)
bigram.transform(tokenized.select("Tokenized_Description")).show(False)
```

```
+----------------------------------------+-----------------------------------
DescOut                                  |ngram_6e68fb3a642a__output       ...
+----------------------------------------+-----------------------------------
|[rabbit, night, light]                  |[rabbit night, night light]      ...
|[doughnut, lip, gloss]                  |[doughnut lip, lip gloss]        ...
...
|[airline, bag, vintage, world, champion]  |[airline bag, bag vintage, vintag...
|[airline, bag, vintage, jet, set, brown]  |[airline bag, bag vintage, vintag...
+----------------------------------------+-----------------------------------
```

Feature Transformation

Working with Text Features

Text input is always considered to be complex because it often requires lots of manipulation to be done before the text can be served as an input to a machine learning model. There are generally two kinds of texts you'll see: free-form text and string categorical variables. We will primarily focus on freeform text because we already discussed categorical variables. Some steps to process the free form text are-

- Fourth, we may convert the words into their numerical representations by means of an algorithm called **Term-Frequency-Inverse-Document-Frequency (TF-IDF)**. Based on this, the words that occur in a few documents are given more weight than words that occur in many documents.

```
from pyspark.ml.feature import HashingTF, IDF
tf = HashingTF()\
.setInputCol("Tokenized_Description")\
.setOutputCol("TFOut")\
.setNumFeatures(10000)

idf = IDF()\
.setInputCol("TFOut")\
.setOutputCol("IDFOut")\
.setMinDocFreq(2)

idf.fit(tf.transform(tfIdfIn))\
.transform(tf.transform(tfIdfIn))\
.select("Tokenized_Description", "TFOut", "IDFOut")\
.show(10)
```

```
+--------------------+--------------------+--------------------+
|Tokenized_Description|               TFOut|               IDFOut|
+--------------------+--------------------+--------------------+
| [red, woolly, hot...|(10000,[52,388,69...|(10000,[52,388,69...|
| [hand, warmer, re...|(10000,[52,3197,3...|(10000,[52,3197,3...|
| [red, coat, rack,...|(10000,[52,477,20...|(10000,[52,477,20...|
| [alarm, clock, ba...|(10000,[52,4995,8...|(10000,[52,4995,8...|
| [set/2, red, retr...|(10000,[52,129,29...|(10000,[52,129,29...|
| [red, toadstool, ...|(10000,[52,773,17...|(10000,[52,773,17...|
| [hand, warmer, re...|(10000,[52,3197,3...|(10000,[52,3197,3...|
| [edwardian, paras...|(10000,[52,2397,5...|(10000,[52,2397,5...|
| [red, woolly, hot...|(10000,[52,388,69...|(10000,[52,388,69...|
| [edwardian, paras...|(10000,[52,2397,5...|(10000,[52,2397,5...|
+--------------------+--------------------+--------------------+
```

Feature Manipulation

What is Feature Manipulation?

Feature manipulation algorithms are automated means of either **expanding the input feature vectors or reducing them to a lower number of dimensions**. Usually, it is the latter that we need while building a machine learning model.

Some examples of feature manipulation algorithms supported by Apache-Spark are-

1. **Principal Component Analysis**: It is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

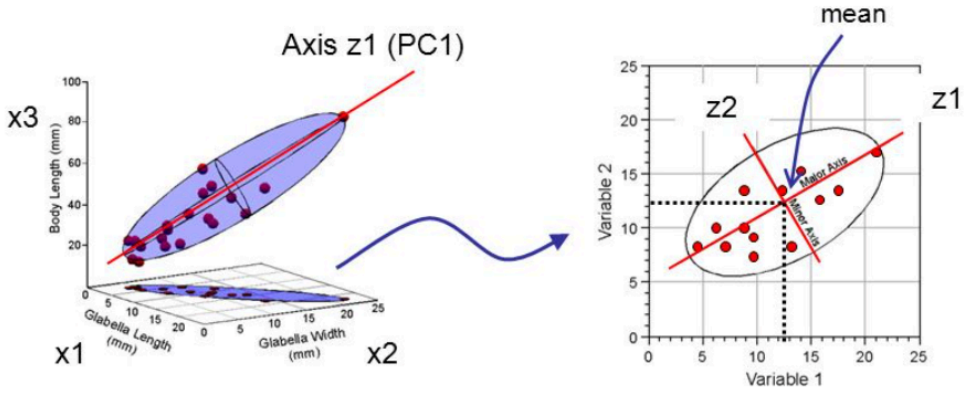
2. **Interactions**: Interaction allows you to create an interaction between two variables manually. It just multiplies the two features together. This transformer is currently only available directly in Scala but can be called from any language using the RFormula.

3. **Polynomial Expansion**: It is used to generate interaction variables of all the input columns. For example, for a degree-2 polynomial, Spark takes every value in our feature vector, multiplies it by every other value in the feature vector, and then stores the results as features. For instance, if we have two input features, we'll get four output features if we use a second degree polynomial (2x2). If we have three input features, we'll get nine output features (3x3).

Feature Manipulation

Example: Principal Component Analysis

- It is a mathematical technique for finding the most important aspects of our data (the principal components).
- It changes the feature representation of our data by creating a new set of features (“aspects”).
- Each new feature is a combination of the original features.
- The power of PCA is that it can create a smaller set of more meaningful features to be input into your model, at the potential cost of interpretability.



You’d want to use PCA if you have a large input dataset and want to reduce the total number of features you have. This frequently comes up in text analysis where the entire feature space is massive and many of the features are largely irrelevant. Using PCA, we can find the most important combinations of features and only include those in our machine learning model. PCA takes a parameter K, specifying the number of output features to create. Generally, this should be much smaller than your input vectors’ dimension.

Feature Manipulation

Example: Principal Component Analysis

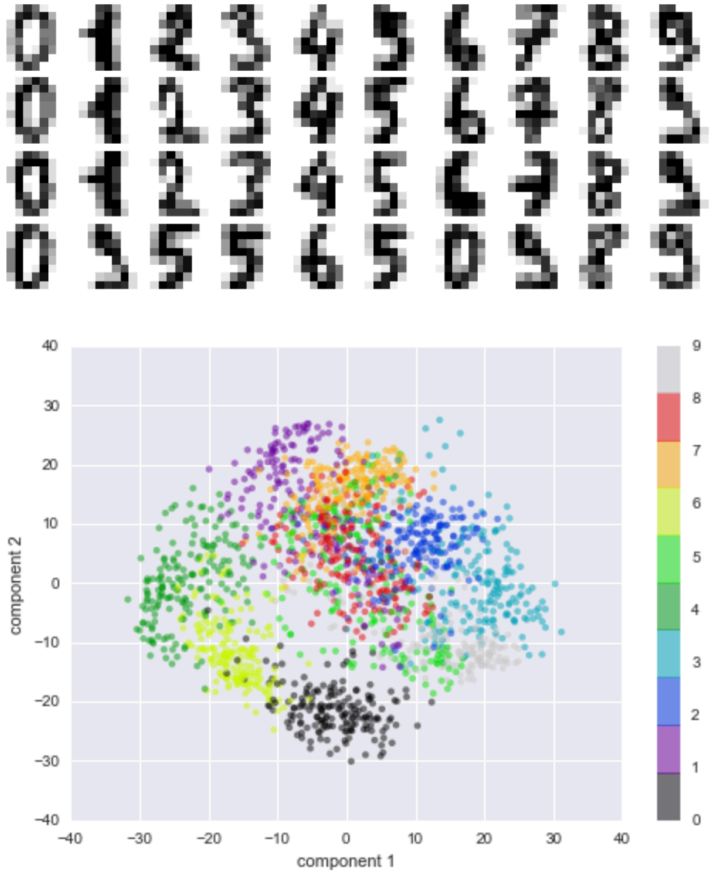
```
from pyspark.ml.feature import VectorAssembler, StandardScaler

df = spark.read.csv("diabetes.csv", header= True)
df= df.select(*(col(c).cast("float").alias(c) for c in df.columns))

# Vector Assembler to assemble all the features into a single vector
va = VectorAssembler().setInputCols(['Pregnancies', 'Glucose', 'BloodPressure'])
df= va.transform(df)
vector_column= df.columns[-1]
df_scale= df.select(col(vector_column).alias("features"))

# Standard Scaler to scale the data
sScaler = StandardScaler().setInputCol("features")
df_scale= sScaler.fit(df_scale).transform(df_scale)
scaled_column= df.columns[-1]
df_scale= df.select(col(scaled_column).alias("scaled_features"))
```

```
from pyspark.ml.feature import PCA
pca = PCA().setInputCol("scaled_features").setK(2)
pca= pca.fit(df_scale).transform(df_scale)
pca_features= pca.columns[-1]
pca.select(col("scaled_features"), col(pca_features).alias("principal_components")).show(5, False)
```

```
+----------------+----------------------------------------+
|scaled_features |principal_components                    |
+----------------+----------------------------------------+
|[6.0,148.0,72.0]|[-156.77339030024655,50.38873651658909] |
|[1.0,85.0,66.0] |[-93.49160365513833,53.28385715428625]  |
|[8.0,183.0,64.0]|[-190.31428823265458,37.548082093334486]|
|[1.0,89.0,66.0] |[-97.45089137263243,52.716540398185955] |
|[0.0,137.0,40.0]|[-141.2674182991563,20.15663231791124]  |
+----------------+----------------------------------------+
only showing top 5 rows
```

Feature Manipulation

Visual Example: Principal Component Analysis



Can you do this by your own? I would like to invite you to take the challenge. Here's the link to the data. All you have to do is to download the dataset, extract he features, assemble them, scale them and apply the Principal Component Analysis to obtain lower level representations.

Feature Selection

What is Feature Selection?

It is often observed while doing analytics specially in context of machine learning that we have a large range of possible features and want to select a smaller subset to use for training. For example, just think of a lot images, each 64x64 in size, representing animals.

If we would like to build an image classifier to classify various animals into their correct classes, not all of the features will be relevant. It may happen because-

1. Correlation: many features might be correlated.
2. Noise: many features might be consisting of a lot of background noise.
3. Overfitting: if we would be using too many features then it might lead to overfitting.

There are a number of ways to evaluate feature importance once you've trained a model but another option is to do some rough filtering beforehand. This process is called feature selection. Spark has some simple options for doing that, such as ChiSqSelector.

Don't believe me? Yeah, it happens.
Find out more about image correlations here

Feature Selection

Example: ChiSqSelector

ChiSqSelector leverages a statistical test to identify features that are not independent from the label we are trying to predict, and drop the uncorrelated features. It's often used with categorical data in order to reduce the number of features you will input into your model, as well as to reduce the dimensionality of text data (in the form of frequencies or counts).

Since this method is based on the Chi-Square test, there are several different ways we can pick the "best" features. The methods are-

- **numTopFeatures** method, which is ordered by p-value;
- **percentile**, which takes a proportion of the input features (instead of just the top N features); and
- **fpr**, which sets a cut off p-value

Read more in-depth about feature selectors from Apache Spark Official Documentation here

Feature Selection

Example: ChiSqSelector

ChiSqSelector leverages a statistical test to identify features that are not independent from the label we are trying to predict, and drop the uncorrelated features. It's often used with categorical data in order to reduce the number of features you will input into your model, as well as to reduce the dimensionality of text data (in the form of frequencies or counts). Since this method is based on the Chi-Square test, there are several different ways we can pick the "best" features. Following is an example of feature selection using the **numTopFeatures** method, which is ordered by p-value;

```
from pyspark.ml.feature import ChiSqSelector
from pyspark.ml.linalg import Vectors

df = spark.createDataFrame([
    (7, Vectors.dense([0.0, 0.0, 18.0, 1.0]), 1.0,),
    (8, Vectors.dense([0.0, 1.0, 12.0, 0.0]), 0.0,),
    (9, Vectors.dense([1.0, 0.0, 15.0, 0.1]), 0.0,)], ["id", "features", "clicked"])

selector = ChiSqSelector(numTopFeatures=1, featuresCol="features",
                         outputCol="selectedFeatures", labelCol="clicked")

result = selector.fit(df).transform(df)

print("ChiSqSelector output with top %d features selected"
      % selector.getNumTopFeatures())
result.show()
```

```
ChiSqSelector output with top 1 features selected
+---+------------------+-------+----------------+
| id|          features|clicked|selectedFeatures|
+---+------------------+-------+----------------+
|  7|[0.0,0.0,18.0,1.0]|    1.0|          [18.0]|
|  8|[0.0,1.0,12.0,0.0]|    0.0|          [12.0]|
|  9|[1.0,0.0,15.0,0.1]|    0.0|          [15.0]|
+---+------------------+-------+----------------+
```

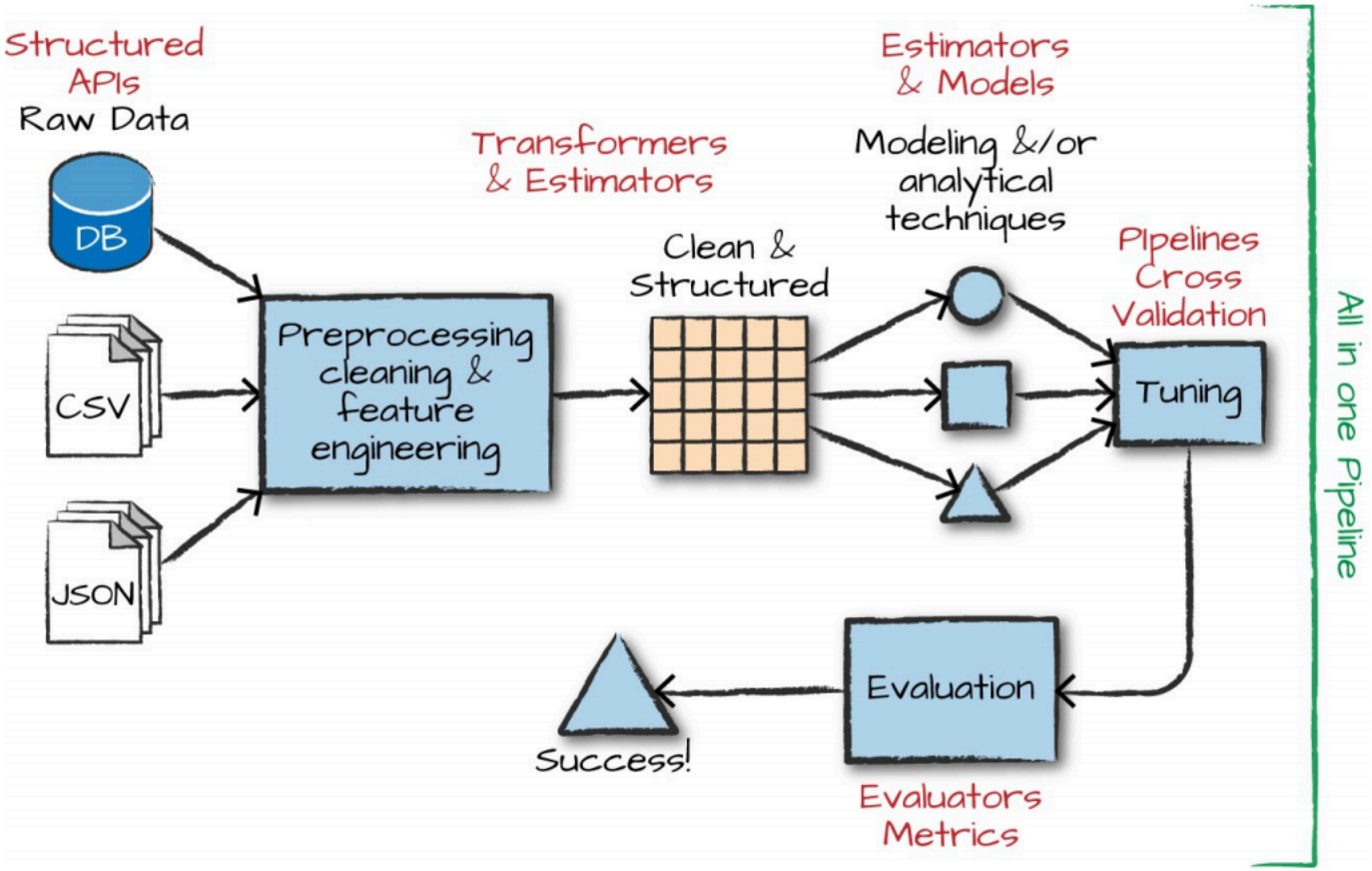
Read more in-depth about feature selectors from Apache Spark Official Documentation here

BTS | Barcelona Technology School

Building Workflow Pipelines

Building Workflow Pipelines

What is the purpose of a Pipeline?

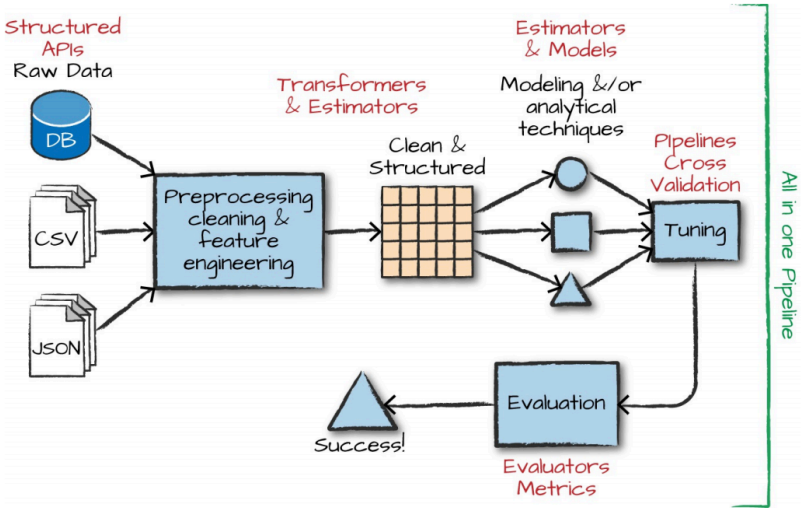


Building Workflow Pipelines

What is the purpose of a Pipeline?

A pipeline allows you to set up a dataflow of the relevant transformations that usually ends with an estimator which is automatically tuned according to our specifications, resulting in a tuned model ready for use. Pipelines are extremely useful in the following cases-

- if we are performing a lot of transformations, writing all the steps and keeping track of DataFrames ends up being quite tedious
- allows us to organise the dataflow of the relevant transformations and feed it straight to the machine learning estimator



Note: it is essential that instances of transformers or models are not reused across different pipelines. Always create a new instance of a model before creating another pipeline.

Building Workflow Pipelines

What is the purpose of a Pipeline?

A pipeline allows you to set up a dataflow of the relevant transformations that usually ends with an estimator which is automatically tuned according to our specifications, resulting in a tuned model ready for use.

```
from pyspark.ml import Pipeline
tkn = Tokenizer().setInputCol("Description").setOutputCol("Tokenized_Description")
englishStopWords = StopWordsRemover.loadDefaultStopWords("english")
stops = StopWordsRemover().setStopWords(englishStopWords)\
.setInputCol("Tokenized_Description")\
.setOutputCol("Removed_Stopwords")

tf = HashingTF()\
.setInputCol("Removed_Stopwords")\
.setOutputCol("TFOut")\
.setNumFeatures(10000)

stages= [tkn, stops, tf]
pipeline = Pipeline(stages = stages)
```

```
pipelineModel = pipeline.fit(sales)
df = pipelineModel.transform(sales)
df = df.select("Tokenized_Description", "Removed_Stopwords", "TFOut").show(5)
```

```
+--------------------+--------------------+--------------------+
|Tokenized_Description|   Removed_Stopwords|               TFOut|
+--------------------+--------------------+--------------------+
| [white, hanging, ...|[white, hanging, ...|(10000,[4618,4667...|
| [white, metal, la...|[white, metal, la...|(10000,[426,671,5...|
| [cream, cupid, he...|[cream, cupid, he...|(10000,[477,6575,...|
| [knitted, union, ...|[knitted, union, ...|(10000,[2352,2589...|
| [red, woolly, hot...|[red, woolly, hot...|(10000,[52,388,69...|
+--------------------+--------------------+--------------------+
only showing top 5 rows
```

BTS | Barcelona Technology School

References

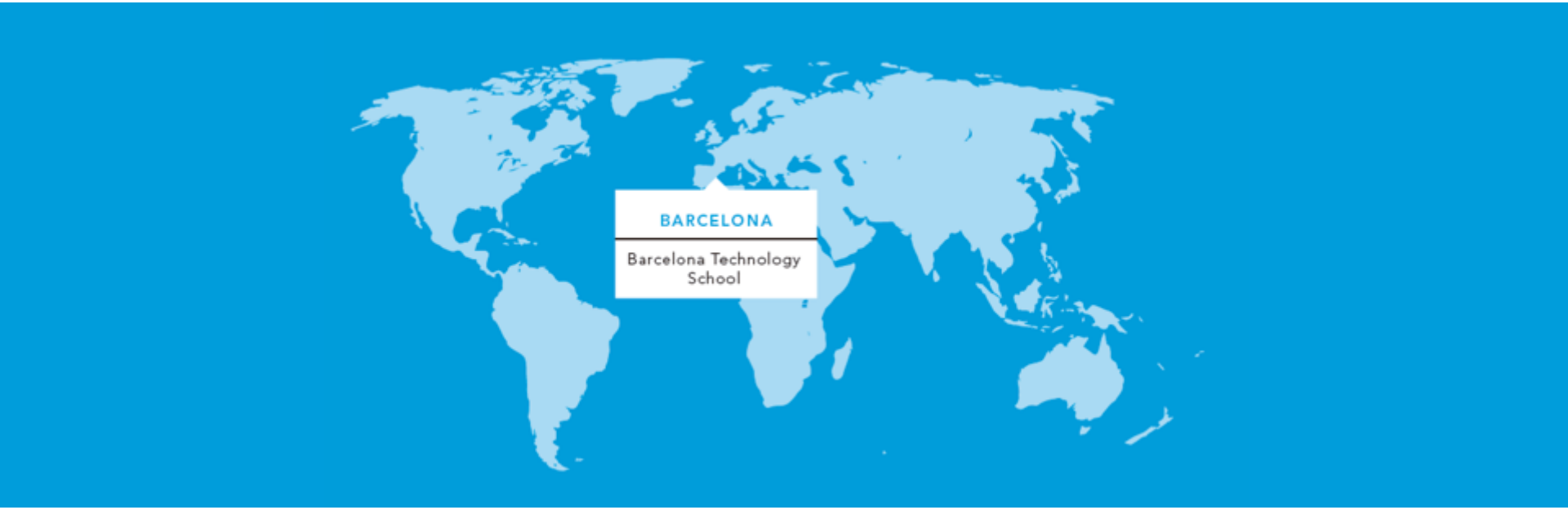
References

Similar Courses

This course has been prepared by drawing some inspiration and references from many similar course. Some of the most similar have been highlighted below for the ready reference-

- CS105X BerkeleyX: Introduction to Apache Spark by UC Berkeley
- COMPSCI516: Apache Spark 101 by Duke University
- CS 4240: Large Scale Parallel Data Processing by Northeastern University
- EECS E6893: Big Data Analytics by Columbia University
- EECS E6895: Advanced Big Data Analytics by Columbia University
- STA 663: Computational Statistics and Statistical Computing (2018) by Duke University

Thanks a lot for your time!


BARCELONA
Barcelona Technology School