BTS | Barcelona Technology School

Real Time Data Analysis

Master in Big Data Solutions 2020-2021

Ankit Tewari

ankit.tewari@bts.tech

Agenda

- Narrowing down on the modern analytics framework
- Exploring how various machine machine learning algorithms can be implemented using the MLib framework
- Understanding by hands-on-exercises the various types of algorithms in MLib
- Comparing the model performances of different machine learning models
- Best practices while using such machine learning models

Contents

- Introduction
- Fundamental classifications of various real world problems
- Regression algorithms
- Classification algorithms
- Hyperparameters of machine learning models
- Training parameters
- Model Scalability

Motivation

Some people to whom we are grateful..

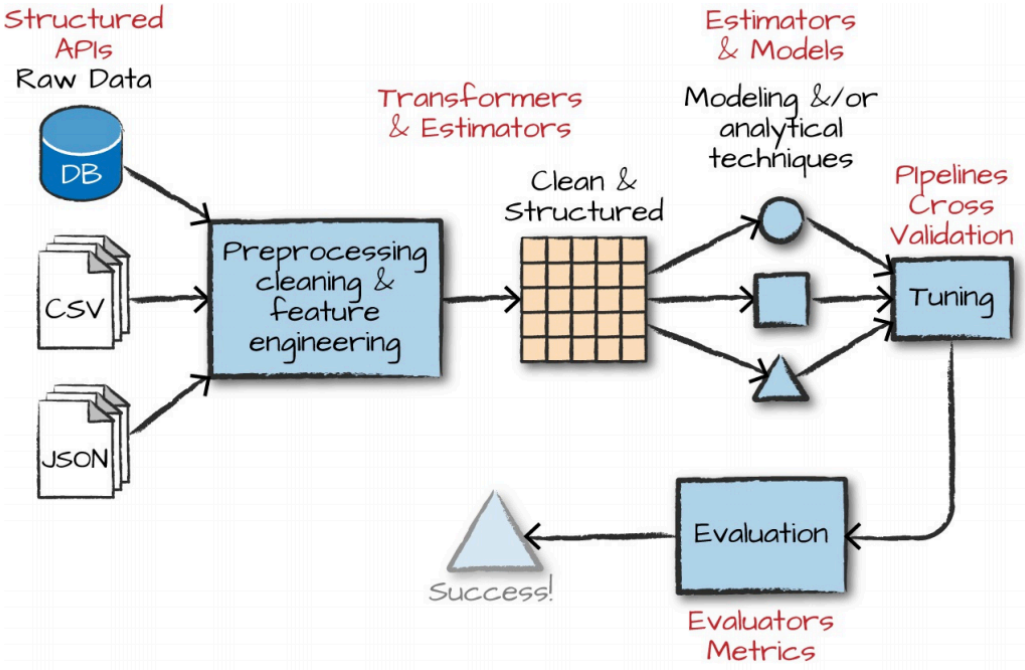
I am extremely grateful to the following for their contribution in shaping my early academic life by teaching me Apache Spark and making me familiar with the cutting edge terminologies in data processing and modelling.

- **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?



- **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

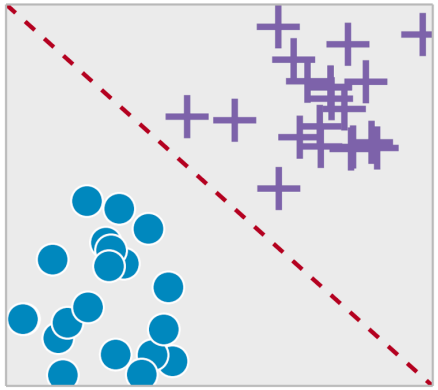
Classification of Machine Learning Algorithms

Modelling the real world phenomena

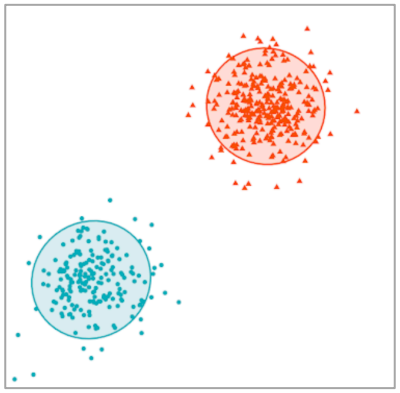
Some of the broadest classes are-

- **Classification**: Predicting the credit risk, topic for a news article, sentiments from social media etc.
- **Regression**: Predicting the viewership of a movie, revenue of a company, annual crop yield etc.
- **Clustering**: Determining the various classes of passenger in an airline data, topics in a given textual data etc.
- **Recommendations**: Recommending relevant movies to users based on previous interest or similar users, suggesting course in an EdTech platform etc.

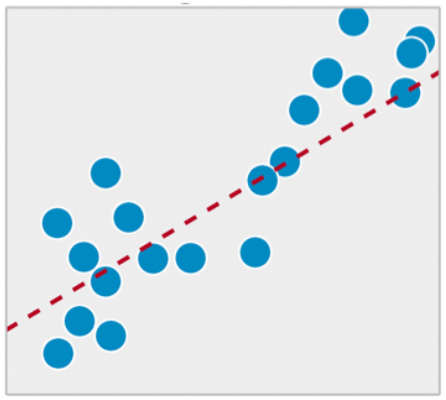
Could you determine the types of problems in the images below?



a)



b)



c)

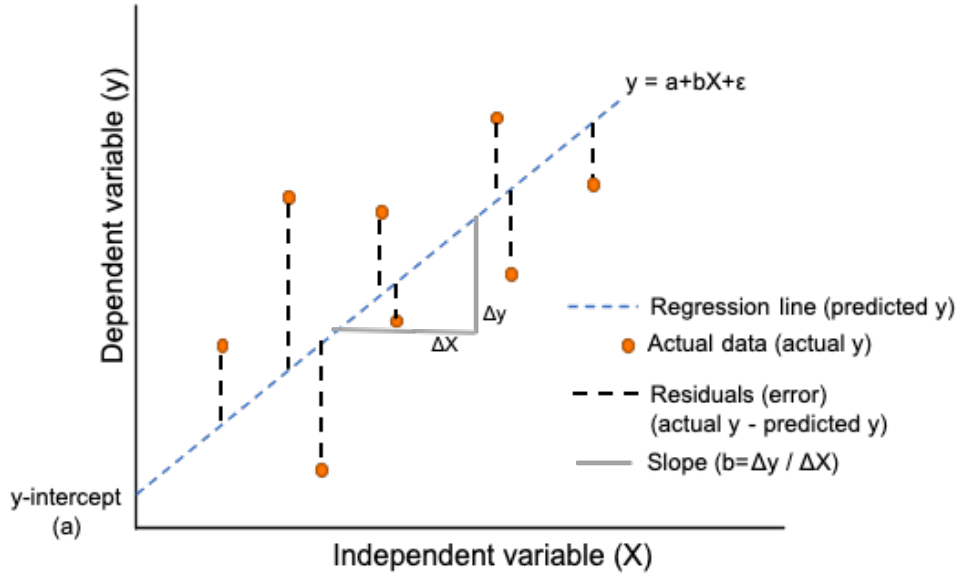
BTS | Barcelona Technology School

Regression

Introduction to Regression

What is Regression?

Regression is a statistical method used in finance, investing, and other disciplines that attempts to determine the strength and character of the relationship between one dependent variable (usually denoted by Y) and a series of other variables (known as independent variables).



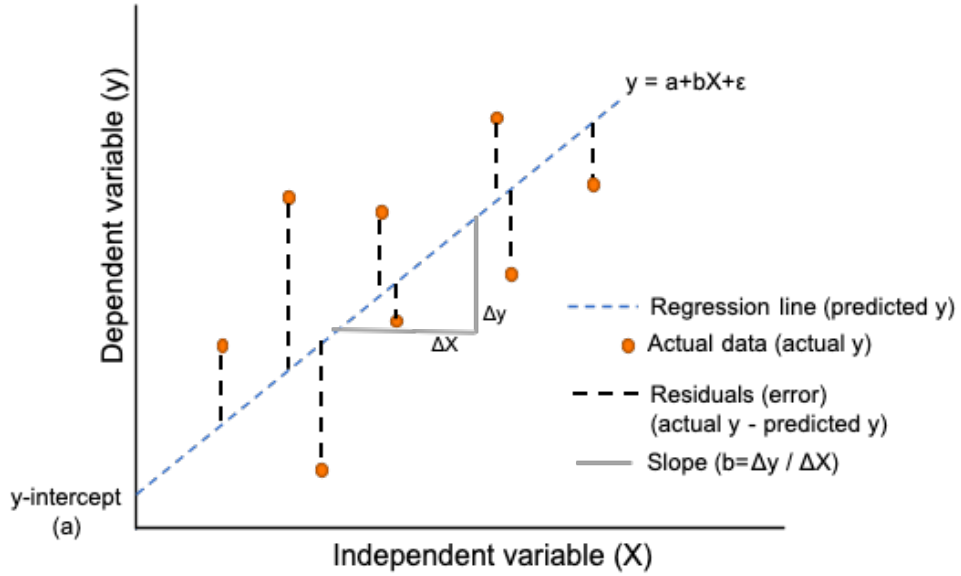
Some common regression models provided by Apache-Spark

- Linear Regression
- Generalised Linear Regression
- Decision Trees
- Random Forest
- Gradient Boosting Trees...

Introduction to Regression

Assumptions in Linear Regression

Regression is a statistical method used in finance, investing, and other disciplines that attempts to determine the strength and character of the relationship between one dependent variable (usually denoted by Y) and a series of other variables (known as independent variables).



There are four assumptions associated with a linear regression model:

- **Linearity**: The relationship between X and the mean of Y is linear.
- **Homoscedasticity**: The variance of residual is the same for any value of X.
- **Independence**: Observations are independent of each other.
- **Normality**: For any fixed value of X, Y is normally distributed.

Introduction to Regression

Types of Regression Model Parameters

The model parameters of any regression algorithms can be divided into- a) Hyperparameters, b) Training parameters, and Prediction Parameters.
For example, Consider the case of **Linear Regression**. The aforesaid parameters are-

- **Hyperparameters**: configurations that determine the basic structure of the model itself. For example, **family**, **regParam** (determines how much weight to give to the regularization term in the objective function) etc.

- **Training Parameters**: used to specify how we perform our training. For example, **maxIter** (Total number of iterations over the data before stopping. Changing this parameter probably won't change your results a ton, so it shouldn't be the first parameter you look to adjust. The default is 100), **weightCol** (name of a weight column used to weigh certain rows more than others. This can be a useful tool if you have some other measure of how important a particular training example is and have a weight associated with it. For example, you might have 10,000 examples where you know that some labels are more accurate than others. You can weigh the labels you know are correct more than the ones you don't)

Introduction to Regression

Beyond Linear Regression..

The standard linear regression is actually a part of a family of algorithms called **generalized linear regression**. The generalized form of linear regression gives you more fine-grained control over what kind of regression model you use.

For instance, these allow you to select the expected noise distribution from a variety of families, including Gaussian (linear regression), binomial (logistic regression), poisson (poisson regression), and gamma (gamma regression). The generalized models also support setting a link function that specifies the relationship between the linear predictor and the mean of the distribution function. The table below shows the available link functions-

Family	Response type	Supported links
Gaussian	Continuous	Identity*, Log, Inverse
Binomial	Binary	Logit*, Probit, CLogLog
Poisson	Count	Log*, Identity, Sqrt
Gamma	Continuous	Inverse*, Idenity, Log
Tweedie	Zero-inflated continuous	Power link function

Introduction to Regression

Reading the data file to a DataFrame

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master('local[*]')\
.appName('airbnb-prediction').getOrCreate()
df = spark.read.parquet('listings.parquet', header = True, inferSchema = True)
df.select("neighbourhood_cleansed", "room_type", "bedrooms", "bathrooms",
 "number_of_reviews", "price").show(5)
```

```
+----------------------+---------------+--------+---------+-----------------+-----+
|neighbourhood_cleansed|      room_type|bedrooms|bathrooms|number_of_reviews|price|
+----------------------+---------------+--------+---------+-----------------+-----+
|      Western Addition|Entire home/apt|     1.0|      1.0|            180.0|170.0|
|        Bernal Heights|Entire home/apt|     2.0|      1.0|            111.0|235.0|
|        Haight Ashbury|   Private room|     1.0|      4.0|             17.0| 65.0|
|        Haight Ashbury|   Private room|     1.0|      4.0|              8.0| 65.0|
|      Western Addition|Entire home/apt|     2.0|      1.5|             27.0|785.0|
+----------------------+---------------+--------+---------+-----------------+-----+
only showing top 5 rows
```

Introduction to Regression

Preparing the training and test data

```
df= df.select("neighbourhood_cleansed", "room_type", "bedrooms", "bathrooms",
 "number_of_reviews", "price")
trainDF, testDF = df.randomSplit([.8, .2], seed=42)
print(f"""There are {trainDF.count()} rows in the training set,
and {testDF.count()} in the test set""")
```

```
There are 5780 rows in the training set,
and 1366 in the test set
```

Introduction to Regression

Creating a Baseline Model

```
from pyspark.ml.feature import VectorAssembler
vecAssembler = VectorAssembler(inputCols=["bedrooms"], outputCol="features")
vecTrainDF = vecAssembler.transform(trainDF)
vecTrainDF.select("bedrooms", "features", "price").show(10)
```

```
+--------+--------+-----+
|bedrooms|features|price|
+--------+--------+-----+
|     0.0|   [0.0]| 99.0|
|     0.0|   [0.0]| 60.0|
|     0.0|   [0.0]|100.0|
|     0.0|   [0.0]|110.0|
|     0.0|   [0.0]|149.0|
|     1.0|   [1.0]|250.0|
|     1.0|   [1.0]| 95.0|
|     1.0|   [1.0]|130.0|
|     1.0|   [1.0]|109.0|
|     1.0|   [1.0]|127.0|
+--------+--------+-----+
only showing top 10 rows
```

```
from pyspark.ml.regression import LinearRegression
lr = LinearRegression(featuresCol="features", labelCol="price")
lrModel = lr.fit(vecTrainDF)
m = round(lrModel.coefficients[0], 2)
b = round(lrModel.intercept, 2)
print(f"""The formula for the linear regression line is
price = {m}*bedrooms + {b}""")
```

```
The formula for the linear regression line is
price = 121.53*bedrooms + 51.8
```

Introduction to Regression

Pipelining and Evaluating Model Performance

```
from pyspark.ml import Pipeline
pipeline = Pipeline(stages=[vecAssembler, lr])
pipelineModel = pipeline.fit(trainDF)

predDF = pipelineModel.transform(testDF)
predDF.select("bedrooms", "features", "price", "prediction").show(5)
```

```
+--------+--------+-----+------------------+
|bedrooms|features|price|        prediction|
+--------+--------+-----+------------------+
|     0.0|   [0.0]|115.0|  51.7988030413165|
|     0.0|   [0.0]|199.0|  51.7988030413165|
|     1.0|   [1.0]| 95.0|173.33189308801815|
|     1.0|   [1.0]| 88.0|173.33189308801815|
|     1.0|   [1.0]| 99.0|173.33189308801815|
+--------+--------+-----+------------------+
only showing top 5 rows
```

```
r2 = regressionEvaluator.setMetricName("r2").evaluate(predDF)
print(f"R2 is {r2}")
```

```
R2 is 0.23959257292844993
```

Introduction to Regression

Improving the Baseline Model: Adding Categorical Features

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

categoricalCols = [field for (field, dataType) in trainDF.dtypes
                   if dataType == "string"]
indexOutputCols = [x + "Index" for x in categoricalCols]
oheOutputCols = [x + "OHE" for x in categoricalCols]

stringIndexer = StringIndexer(inputCols=categoricalCols,
    outputCols=indexOutputCols,
    handleInvalid="skip")
oheEncoder = OneHotEncoder(inputCols=indexOutputCols,
                           outputCols=oheOutputCols)
numericCols = [field for (field, dataType) in trainDF.dtypes
               if ((dataType == "double") & (field != "price"))]

assemblerInputs = oheOutputCols + numericCols
vecAssembler = VectorAssembler(inputCols=assemblerInputs,
 outputCol="features")
```

Introduction to Regression

Improving the Baseline Model: Adding Categorical Features

```
from pyspark.ml.feature import RFormula
rFormula = RFormula(formula="price ~ .",
 featuresCol="features",
 labelCol="price",
 handleInvalid="skip")
```

```
lr = LinearRegression(labelCol="price", featuresCol="features", )
pipeline = Pipeline(stages = [stringIndexer, oheEncoder, vecAssembler, lr])
pipelineModel = pipeline.fit(trainDF)
predDF = pipelineModel.transform(testDF)
predDF.select("features", "price", "prediction").show(5)
```

```
+--------------------+-----+------------------+
|            features|price|        prediction|
+--------------------+-----+------------------+
|(40,[15,35,38,39]...|115.0| 5.219459229122052|
|(40,[15,35,38,39]...|199.0|-5.485956453427008|
|(40,[15,35,37,38,...| 95.0|113.94354528880599|
|(40,[15,35,37,38,...| 88.0|110.37507339462296|
|(40,[15,35,37,38,...| 99.0| 104.1897221113724|
+--------------------+-----+------------------+
only showing top 5 rows
```

Introduction to Regression

Improving the Baseline Model: Adding Categorical Features

```
from pyspark.ml.feature import RFormula
rFormula = RFormula(formula="price ~ .",
 featuresCol="features",
 labelCol="price",
 handleInvalid="skip")
```

```
lr = LinearRegression(labelCol="price", featuresCol="features", )
pipeline = Pipeline(stages = [rFormula, lr])
pipelineModel = pipeline.fit(trainDF)
predDF = pipelineModel.transform(testDF)
predDF.select("features", "price", "prediction").show(5)
```

```
+--------------------+-----+------------------+
|            features|price|        prediction|
+--------------------+-----+------------------+
|(40,[15,35,38,39]...|115.0| 5.219459229122052|
|(40,[15,35,38,39]...|199.0|-5.485956453427008|
|(40,[15,35,37,38,...| 95.0|113.94354528880599|
|(40,[15,35,37,38,...| 88.0|110.37507339462296|
|(40,[15,35,37,38,...| 99.0| 104.1897221113724|
+--------------------+-----+------------------+
only showing top 5 rows
```

Introduction to Regression

Improving the Baseline Model: Adding Categorical Features

```
from pyspark.ml.evaluation import RegressionEvaluator
regressionEvaluator = RegressionEvaluator(
    predictionCol="prediction",
    labelCol="price",
    metricName="rmse")
rmse = regressionEvaluator.evaluate(predDF)
print(f"RMSE is {rmse:.1f}")
```

```
RMSE is 191.1
```

```
r2 = regressionEvaluator.setMetricName("r2").evaluate(predDF)
print(f"R2 is {r2}")
```

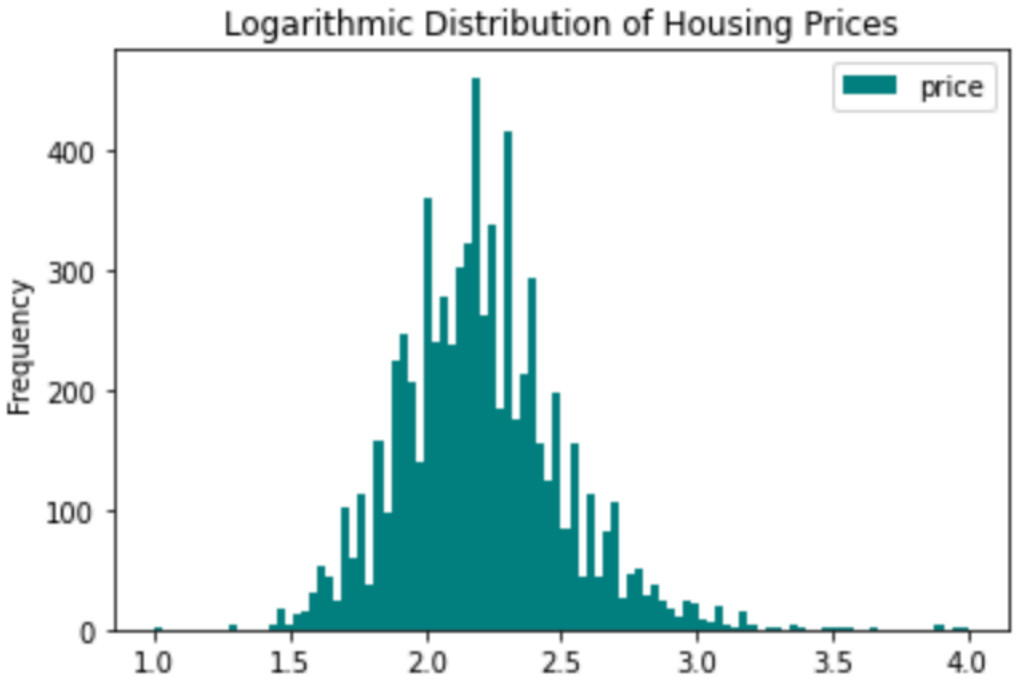
```
R2 is 0.2923250512096982
```

Introduction to Regression

Improving the Baseline Model: Distribution of the housing prices

Introduction to Regression

Improving the Baseline Model: Distribution of the housing prices on logarithmic scale



Introduction to Regression

Improving the Baseline Model: Transforming Distribution of the Housing Prices

```
trainDF.show(5)
```

```
+----------------------+---------------+--------+---------+------------------+-----+
|neighbourhood_cleansed|      room_type|bedrooms|bathrooms|number_of_reviews|price|
+----------------------+---------------+--------+---------+------------------+-----+
|               Bayview|Entire home/apt|     0.0|      1.0|               2.0| 99.0|
|               Bayview|Entire home/apt|     0.0|      1.0|               3.0| 60.0|
|               Bayview|Entire home/apt|     0.0|      1.0|              20.0|100.0|
|               Bayview|Entire home/apt|     0.0|      1.0|              39.0|110.0|
|               Bayview|Entire home/apt|     0.0|      1.0|              95.0|149.0|
+----------------------+---------------+--------+---------+------------------+-----+
only showing top 5 rows
```

```
trainDF.withColumn("price", log10(col("price"))).show(5)
```

```
+----------------------+---------------+--------+---------+------------------+------------------+
|neighbourhood_cleansed|      room_type|bedrooms|bathrooms|number_of_reviews|             price|
+----------------------+---------------+--------+---------+------------------+------------------+
|               Bayview|Entire home/apt|     0.0|      1.0|               2.0|  1.99563519459755|
|               Bayview|Entire home/apt|     0.0|      1.0|               3.0|1.7781512503836436|
|               Bayview|Entire home/apt|     0.0|      1.0|              20.0|               2.0|
|               Bayview|Entire home/apt|     0.0|      1.0|              39.0| 2.041392685158225|
|               Bayview|Entire home/apt|     0.0|      1.0|              95.0| 2.173186268412274|
+----------------------+---------------+--------+---------+------------------+------------------+
only showing top 5 rows
```

Introduction to Regression

Improving the Baseline Model: Transforming Distribution of the Housing Prices

```
trainDF= trainDF.withColumn("price", log10(col("price")))
lr = LinearRegression(labelCol="price", featuresCol="features", )
pipeline = Pipeline(stages = [rFormula, lr])
pipelineModel = pipeline.fit(trainDF)
testDF= testDF.withColumn("price", log10(col("price")))
predDF = pipelineModel.transform(testDF)
predDF.select("features", "price", "prediction").show(5)
```

```
+-------------------+-----------------+------------------+
|           features|            price|        prediction|
+-------------------+-----------------+------------------+
|(40,[15,35,38,39]...| 2.060697840353612| 1.903884654822569|
|(40,[15,35,38,39]...| 2.298853076409707| 1.899111693969867|
|(40,[15,35,37,38,...|1.9777236052888478| 2.052488947194184|
|(40,[15,35,37,38,...|1.9444826721501687| 2.050897960243283|
|(40,[15,35,37,38,...|  1.99563519459755|2.0481402495283887|
+-------------------+-----------------+------------------+
only showing top 5 rows
```

```
r2 = regressionEvaluator.setMetricName("r2").evaluate(predDF)
print(f"R2 is {r2}")
```

```
R2 is 0.5438922389401064
```

Introduction to Regression

Can you compare between the Predictions?

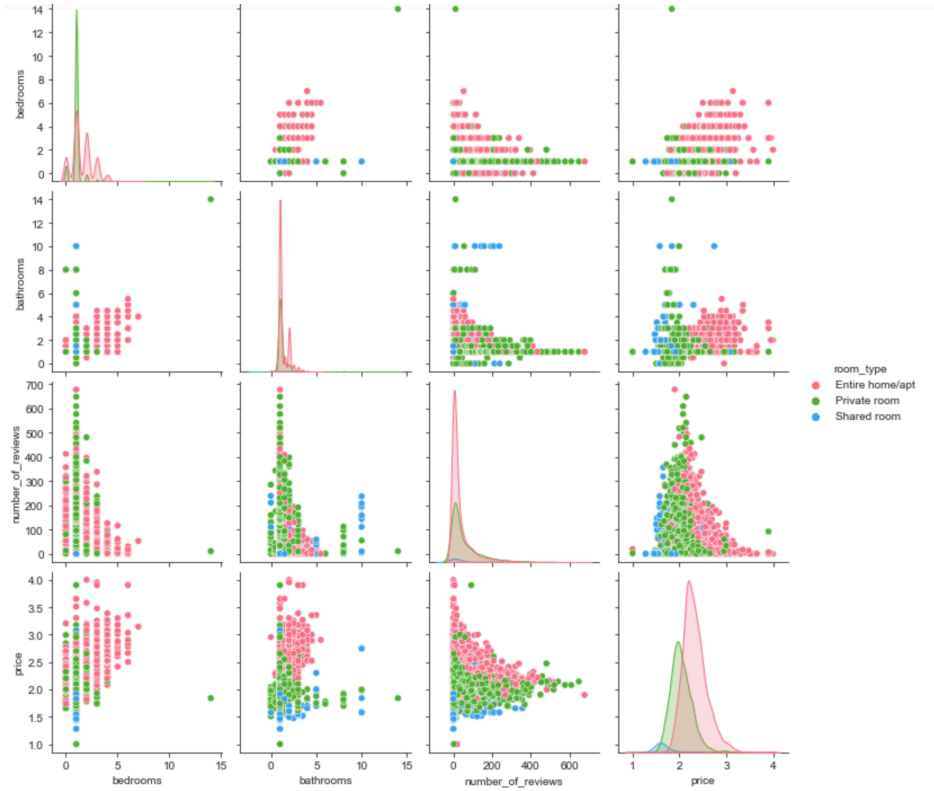
```
+--------------------+-----+-------------------+
|            features|price|         prediction|
+--------------------+-----+-------------------+
|(40,[15,35,38,39]...|115.0| 5.219459229122052|
|(40,[15,35,38,39]...|199.0|-5.485956453427008|
|(40,[15,35,37,38,...| 95.0|113.94354528880599|
|(40,[15,35,37,38,...| 88.0|110.37507339462296|
|(40,[15,35,37,38,...| 99.0| 104.1897221113724|
+--------------------+-----+-------------------+
only showing top 5 rows
```

```
+--------------------+------------------+------------------+
|            features|             price|        prediction|
+--------------------+------------------+------------------+
|(40,[15,35,38,39]...| 2.060697840353612| 1.903884654822569|
|(40,[15,35,38,39]...| 2.298853076409707| 1.899111693969867|
|(40,[15,35,37,38,...|1.9777236052888478| 2.052488947194184|
|(40,[15,35,37,38,...|1.9444826721501687| 2.050897960243283|
|(40,[15,35,37,38,...|  1.99563519459755|2.0481402495283887|
+--------------------+------------------+------------------+
only showing top 5 rows
```

Introduction to Regression

Exploring further..

```
import pandas as pd
import seaborn as sb
from matplotlib import pyplot as plt
sb.set_style("ticks")
sb.pairplot(trainDF.toPandas(), hue= "room_type", diag_kind = "kde",kind = "scatter",palette = "husl")
plt.show()
```



Introduction to Regression

Hyperparameter Optimisation and Cross Validation

```
from pyspark.ml.regression import RandomForestRegressor
rf = RandomForestRegressor(labelCol="price", maxBins=40, seed=42)

trainDF_rf= trainDF\
.withColumn("price", log10(col("price")))\
.withColumn("number_of_reviews", log(col("number_of_reviews"))).dropna()

pipeline = Pipeline(stages = [stringIndexer, oheEncoder, vecAssembler, rf])
pipelineModel = pipeline.fit(trainDF_rf)

testDF_rf= testDF\
.withColumn("price", log10(col("price")))\
.withColumn("number_of_reviews", log(col("number_of_reviews"))).dropna()

predDF = pipelineModel.transform(testDF_rf)
predDF.select("features", "price", "prediction").show(5)
```

```
+--------------------+------------------+-------------------+
|            features|             price|         prediction|
+--------------------+------------------+-------------------+
|(40,[15,35,38,39]...| 0.3140143159483305|0.33164819400294626|
|(40,[15,35,38,39]...|0.36151121566096056| 0.3292419724817349|
|(40,[15,35,37,38,...| 0.2961655971270392|  0.336467845316805|
|(40,[15,35,37,38,...|0.28880407737646113| 0.3368643856092421|
|(40,[15,35,37,38,...|0.30008115445973177|0.33445816408803075|
+--------------------+------------------+-------------------+
only showing top 5 rows
```

```
r2 = regressionEvaluator.setMetricName("r2").evaluate(predDF)
print(f"R2 is {r2}")
```

```
R2 is 0.5652900437054849
```

Introduction to Regression

Hyperparameter Optimisation and Cross Validation

```
pipeline = Pipeline(stages = [stringIndexer, oheEncoder, vecAssembler, lr])

from pyspark.ml.tuning import ParamGridBuilder
paramGrid = (ParamGridBuilder()
    .addGrid(rf.maxDepth, [2, 4, 6])
    .addGrid(rf.numTrees, [10, 100])
    .build())
evaluator = RegressionEvaluator(labelCol="price",
    predictionCol="prediction",
    metricName="r2")
```

```
from pyspark.ml.tuning import CrossValidator
cv = CrossValidator(estimator=pipeline,
 evaluator=evaluator,
 estimatorParamMaps=paramGrid,
 numFolds=3,
 seed=42)
cvModel = cv.fit(trainDF_rf)
```

```
 ({Param(parent='RandomForestRegressor_e8904c07469b', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g.,
depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.'): 2,
   Param(parent='RandomForestRegressor_e8904c07469b', name='numTrees', doc='Number of trees to train (>= 1).'): 100
},
  0.4501165087742496),
 ({Param(parent='RandomForestRegressor_e8904c07469b', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g.,
depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.'): 4,
   Param(parent='RandomForestRegressor_e8904c07469b', name='numTrees', doc='Number of trees to train (>= 1).'): 10}
,
  0.5255637221568302),
 ({Param(parent='RandomForestRegressor_e8904c07469b', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g.,
depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.'): 4,
   Param(parent='RandomForestRegressor_e8904c07469b', name='numTrees', doc='Number of trees to train (>= 1).'): 100
},
  0.52885878186357),
 ({Param(parent='RandomForestRegressor_e8904c07469b', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g.,
depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.'): 6,
   Param(parent='RandomForestRegressor_e8904c07469b', name='numTrees', doc='Number of trees to train (>= 1).'): 10}
,
  0.5530713561690613),
 ({Param(parent='RandomForestRegressor_e8904c07469b', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g.,
depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.'): 6,
   Param(parent='RandomForestRegressor_e8904c07469b', name='numTrees', doc='Number of trees to train (>= 1).'): 100
},
  0.5634186078387311)]
```

Introduction to Regression

Model Scalability

Model scalability is an important consideration when choosing your model. In general, Spark has great support for training large-scale machine learning models (note, these are large scale; on single-node workloads there are a number of other tools that also perform well).

The table below is a simple model scalability scorecard to use to find the best model for your particular task (**if scalability is the core consideration**). The actual scalability will depend on your configuration, machine size, and other specifics but should make for a good heuristic.

Model	Number features	Training examples
Linear regression	1 to 10 million	No limit
Generalized linear regression	4,096	No limit
Isotonic regression	N/A	Millions
Decision trees	1,000s	No limit
Random forest	10,000s	No limit
Gradient-boosted trees	1,000s	No limit
Survival regression	1 to 10 million	No limit

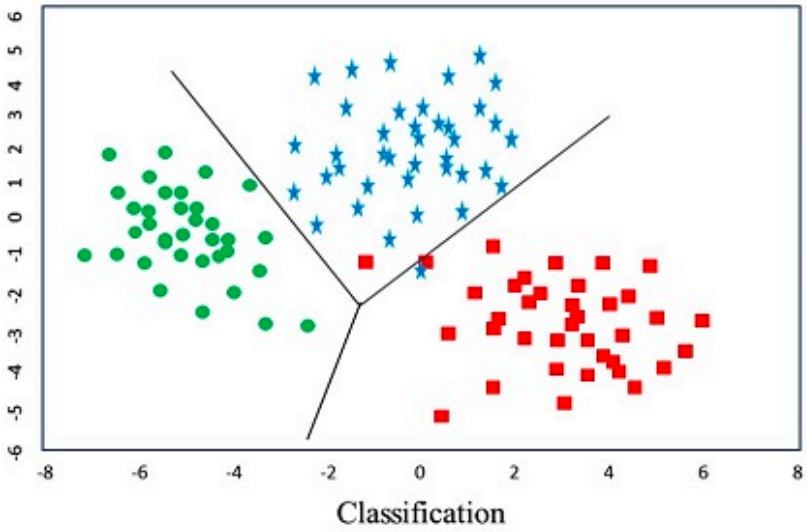
BTS | Barcelona Technology School

Classification

Introduction to Classification

What is Classification?

Classification is a process of categorizing a given set of data into classes, It can be performed on both structured or unstructured data. The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories.



Some common classification models provided by Apache-Spark

- Logistic regression
- Decision trees
- Random forests
- Gradient-boosted trees...

Introduction to Classification

Types of Classification Model Parameters

The model parameters of any classification algorithms can be divided into- a) Hyperparameters, b) Training parameters and c) Prediction parameters.

For example, Consider the case of **Logistic Regression**. The aforesaid parameters are-

- **Hyperparameters**: configurations that determine the basic structure of the model itself. For example, **family** (Can be multinomial (two or more distinct labels; multiclass classification) or binary (only two distinct labels; binary classification), **fitIntercept** (True or False), **regParam** (determines how much weight to give to the regularization term in the objective function)

- **Training Parameters**: used to specify how we perform our training. For example, **maxIter** (Total number of iterations over the data before stopping. Changing this parameter probably won't change your results a ton, so it shouldn't be the first parameter you look to adjust. The default is 100), **weightCol** (name of a weight column used to weigh certain rows more than others. This can be a useful tool if you have some other measure of how important a particular training example is and have a weight associated with it. For example, you might have 10,000 examples where you know that some labels are more accurate than others. You can weigh the labels you know are correct more than the ones you don't)

- **Prediction Parameters**: help determine how the model should actually be making predictions at prediction time, but do not affect training. For example, **threshold** (A Double in the range of 0 to 1. This parameter is the probability threshold for when a given class should be predicted. It is used to balance between false positives and false negatives. For instance, if a mistaken prediction would be costly—you might want to make its prediction threshold very high).

Introduction to Classification

Let's read the data first..

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, FloatType

spark = SparkSession.builder.master('local[*]').appName('term-deposit-prediction').getOrCreate()
df = spark.read.csv('bank.csv', sep= ';', header = True, inferSchema = True)
```

```
df.select(df.columns[:8]).show(4, truncate= False)
```

```
+---+----------+-------+---------+-------+-------+-------+----+
|age|job       |marital|education|default|balance|housing|loan|
+---+----------+-------+---------+-------+-------+-------+----+
|30 |unemployed|married|primary  |no     |1787   |no     |no  |
|33 |services  |married|secondary|no     |4789   |yes    |yes |
|35 |management|single |tertiary |no     |1350   |yes    |no  |
|30 |management|married|tertiary |no     |1476   |yes    |yes |
+---+----------+-------+---------+-------+-------+-------+----+
only showing top 4 rows
```

Introduction to Classification

Some Summary Statistics?

```
df.select(df.columns[:8]).describe().show()
```

```
+-------+------------------+-------+--------+---------+-------+------------------+-------+----+
|summary|               age|    job| marital|education|default|           balance|housing|loan|
+-------+------------------+-------+--------+---------+-------+------------------+-------+----+
|  count|              4521|   4521|    4521|     4521|   4521|              4521|   4521|4521|
|   mean| 41.17009511170095|   null|    null|     null|   null|1422.6578190665782|   null|null|
| stddev|10.576210958711263|   null|    null|     null|   null|3009.6381424673395|   null|null|
|    min|                19| admin.|divorced|  primary|     no|             -3313|     no|  no|
|    max|                87|unknown|  single|  unknown|    yes|             71188|    yes| yes|
+-------+------------------+-------+--------+---------+-------+------------------+-------+----+
```

Introduction to Classification

Exploring Data Types

```
numeric_features = [t[0] for t in df.dtypes if t[1] == 'int']
df.select(numeric_features).describe().toPandas().transpose()
```

summary	count	mean	stddev	min	max
age	4521	41.17009511170095	10.576210958711263	19	87
balance	4521	1422.6578190665782	3009.6381424673395	-3313	71188
day	4521	15.915284229152842	8.247667327229934	1	31
duration	4521	263.96129174961294	259.85663262468216	4	3025
campaign	4521	2.793629727936297	3.1098066601885823	1	50
pdays	4521	39.766644547666445	100.12112444301656	-1	871
previous	4521	0.5425790754257908	1.6935623506071211	0	25

```
df = df.select('age', 'job', 'marital', 'education', 'default',
               'balance', 'housing', 'loan', 'contact', 'duration',
               'campaign', 'pdays', 'previous', 'poutcome', 'deposit')
df.dtypes
```

```
[('age', 'int'),
 ('job', 'string'),
 ('marital', 'string'),
 ('education', 'string'),
 ('default', 'string'),
 ('balance', 'int'),
 ('housing', 'string'),
 ('loan', 'string'),
 ('contact', 'string'),
 ('duration', 'int'),
 ('campaign', 'int'),
 ('pdays', 'int'),
 ('previous', 'int'),
 ('poutcome', 'string'),
 ('deposit', 'string')]
```

Introduction to Classification

Pipelining the Process of Feature Engineering

```
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler

stages = []
numericCols = [field for (field, dataType) in df.dtypes
               if ((dataType == "int") & (field != "deposit"))]
categoricalColumns = [field for (field, dataType) in df.dtypes
                      if dataType == "string"]
categoricalColumns.remove("deposit")

for categoricalCol in categoricalColumns:
    stringIndexer = StringIndexer(inputCol = categoricalCol, outputCol = categoricalCol + 'Index')
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
    stages += [stringIndexer, encoder]

label_stringIdx = StringIndexer(inputCol = 'deposit', outputCol = 'label')
stages += [label_stringIdx]


assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]
```

```
train, test = df.randomSplit([0.7, 0.3], seed = 2018)
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))
```

```
Training Dataset Count: 3204
Test Dataset Count: 1317
```

Introduction to Classification

Creating the Baseline Model

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

lr = LogisticRegression(labelCol="label", featuresCol="features")
stages+= [lr]
pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(train)
pred = pipelineModel.transform(test)
pred.select("features", "label", "prediction").show(5)
```

```
+--------------------+-----+----------+
|            features|label|prediction|
+--------------------+-----+----------+
|(30,[10,12,15,16,...|  1.0|       0.0|
|(30,[4,12,13,16,1...|  0.0|       0.0|
|(30,[10,12,13,16,...|  1.0|       1.0|
|(30,[10,12,13,16,...|  0.0|       0.0|
|(30,[10,12,13,16,...|  0.0|       0.0|
+--------------------+-----+----------+
only showing top 5 rows
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(
    labelCol="label", metricName="areaUnderROC")
areaUnderROC = evaluator.evaluate(pred)
print("Area Under ROC = %g " % (areaUnderROC))
```

```
Area Under ROC = 0.862333
```

Introduction to Classification

Comparing the Baseline Model with Random Forest

```
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml import Pipeline
rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label')
stages.remove(stages[-1])
stages+= [rf]
pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(train)
pred = pipelineModel.transform(test)
pred.select("features", "label", "prediction").show(5)
```

```
+--------------------+-----+----------+
|            features|label|prediction|
+--------------------+-----+----------+
|(30,[10,12,15,16,...|  1.0|       0.0|
|(30,[4,12,13,16,1...|  0.0|       0.0|
|(30,[10,12,13,16,...|  1.0|       0.0|
|(30,[10,12,13,16,...|  0.0|       0.0|
|(30,[10,12,13,16,...|  0.0|       0.0|
+--------------------+-----+----------+
only showing top 5 rows
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(
    labelCol="label", metricName="areaUnderROC")
areaUnderROC = evaluator.evaluate(pred)
print("Area Under ROC = %g " % (areaUnderROC))
```

```
Area Under ROC = 0.86742
```

Introduction to Classification

Hyperparameter Optimisation and Cross Validation

```
from pyspark.ml.tuning import ParamGridBuilder
paramGrid = (ParamGridBuilder()
    .addGrid(rf.maxDepth, [2, 4, 6])
    .addGrid(rf.numTrees, [10, 100])
    .build())
```

```
from pyspark.ml.tuning import CrossValidator
cv = CrossValidator(estimator=pipeline,
    evaluator=evaluator,
    estimatorParamMaps=paramGrid,
    numFolds=3,
    seed=42)
cvModel = cv.fit(train)
list(zip(cvModel.getEstimatorParamMaps(), cvModel.avgMetrics))
```

```
[({Param(parent='RandomForestClassifier_b77223118c4d', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g.
, depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.'): 2,
   Param(parent='RandomForestClassifier_b77223118c4d', name='numTrees', doc='Number of trees to train (>= 1).'): 10
},
  0.6456131717826524),
 ({Param(parent='RandomForestClassifier_b77223118c4d', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g.
, depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.'): 2,
   Param(parent='RandomForestClassifier_b77223118c4d', name='numTrees', doc='Number of trees to train (>= 1).'): 10
0},
  0.8390147890914366),
 ({Param(parent='RandomForestClassifier_b77223118c4d', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g.
, depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.'): 4,
   Param(parent='RandomForestClassifier_b77223118c4d', name='numTrees', doc='Number of trees to train (>= 1).'): 10
},
  0.8340064510451268),
 ({Param(parent='RandomForestClassifier_b77223118c4d', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g.
, depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.'): 4,
   Param(parent='RandomForestClassifier_b77223118c4d', name='numTrees', doc='Number of trees to train (>= 1).'): 10
0},
  0.856124756408414),
 ({Param(parent='RandomForestClassifier_b77223118c4d', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g.
, depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.'): 6,
   Param(parent='RandomForestClassifier_b77223118c4d', name='numTrees', doc='Number of trees to train (>= 1).'): 10
},
  0.8493498678607267),
 ({Param(parent='RandomForestClassifier_b77223118c4d', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g.
, depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.'): 6,
   Param(parent='RandomForestClassifier_b77223118c4d', name='numTrees', doc='Number of trees to train (>= 1).'): 10
0},
  0.865440553397082)]
```

Introduction to Classification

Model Scalability

Model	Features count	Training examples	Output classes
Logistic regression	1 to 10 million	No limit	Features x Classes < 10 million
Decision trees	1,000s	No limit	Features x Classes < 10,000s
Random forest	10,000s	No limit	Features x Classes < 100,000s
Gradient-boosted trees	1,000s	No limit	Features x Classes < 10,000s

BTS | Barcelona Technology School

References

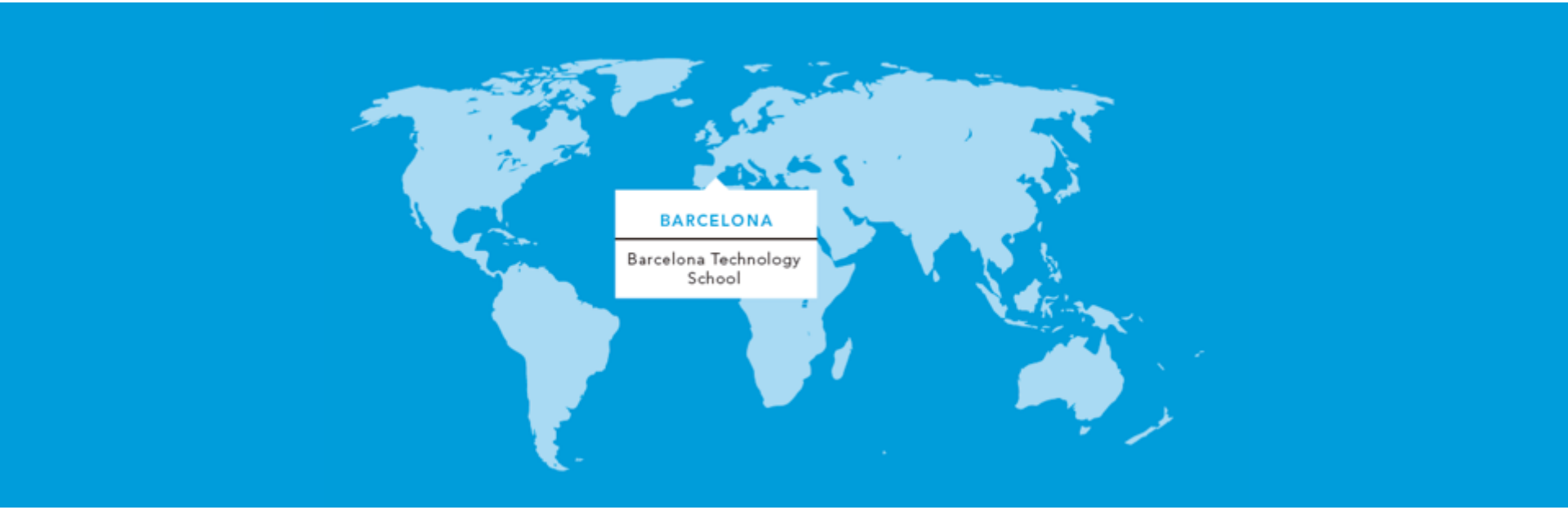
References

Some course that we take inspiration from

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