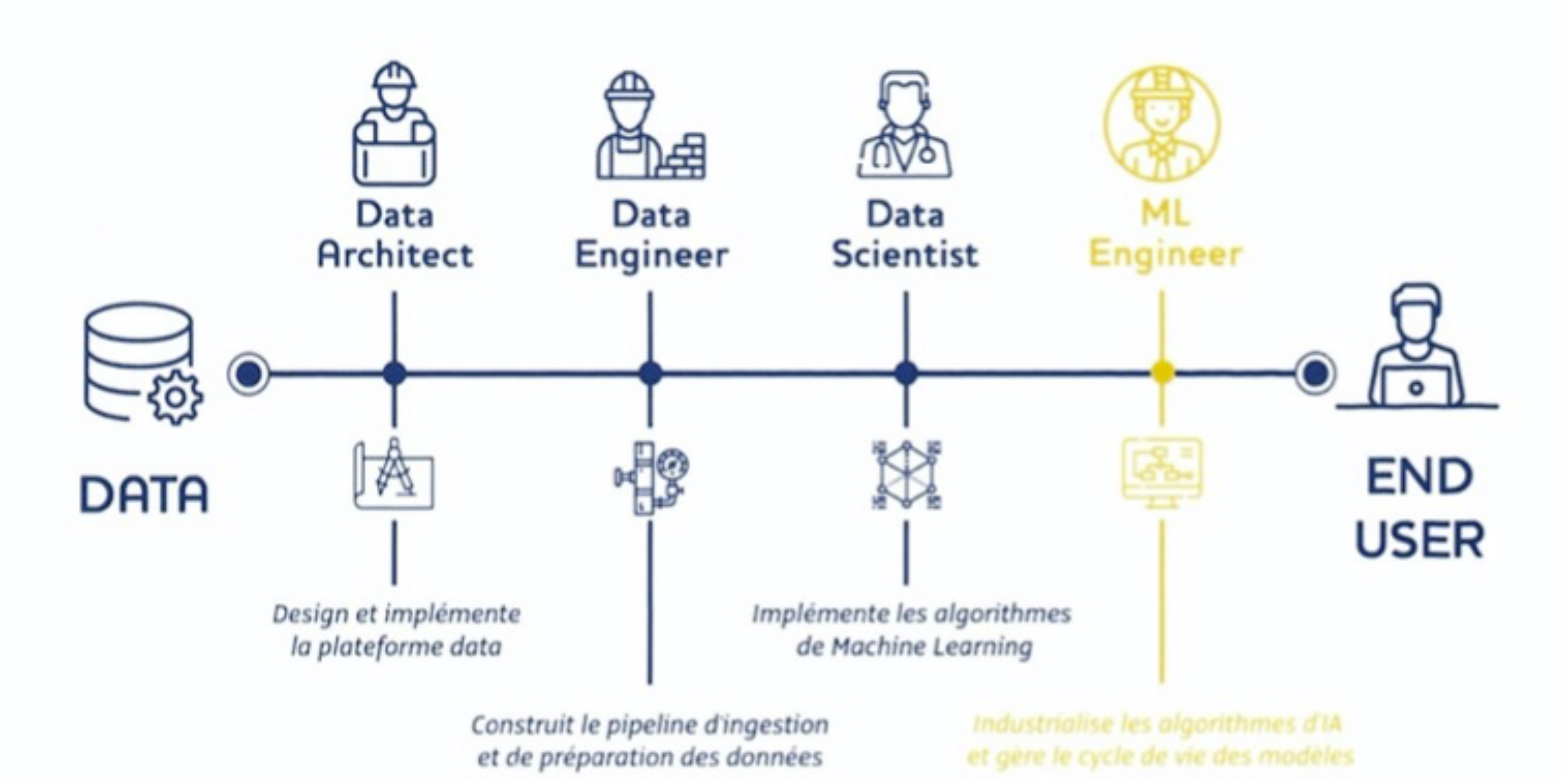
Aperçu sur la préparation de données pour le ML en Spark

Master DAC – Bases de Données Large Echelle

Mohamed-Amine Baazizi
mohamed-amine.baazizi@lip6.fr

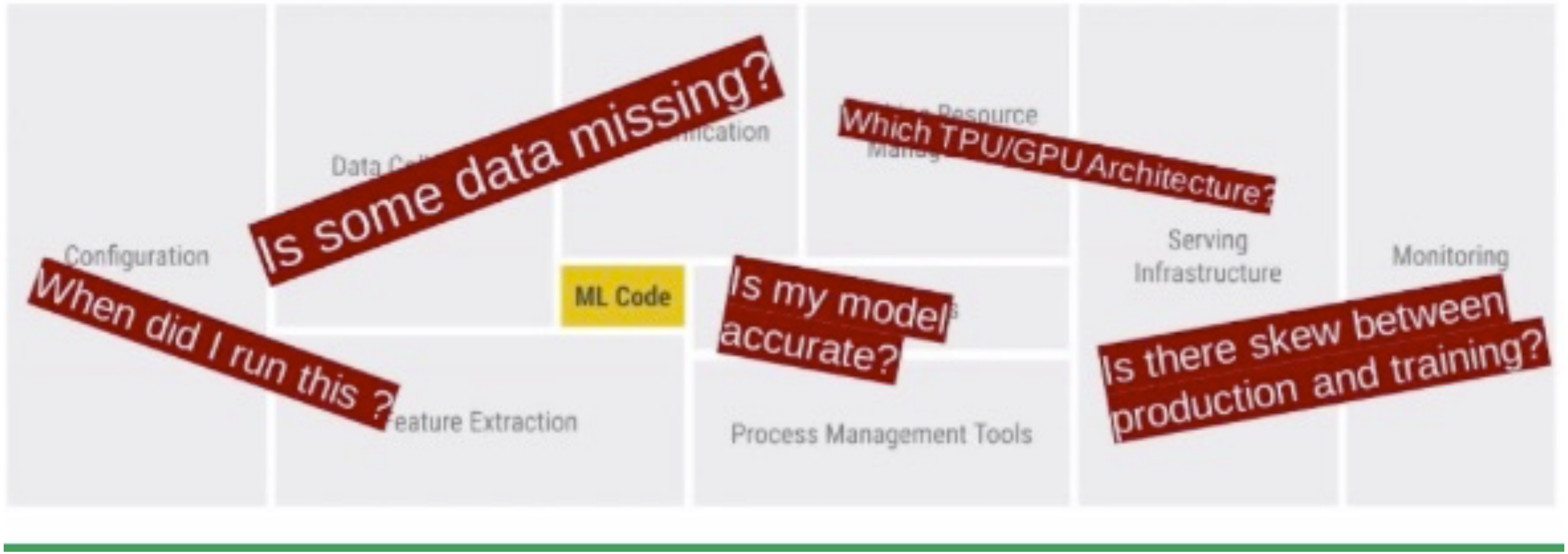
2020-2021

The data journey



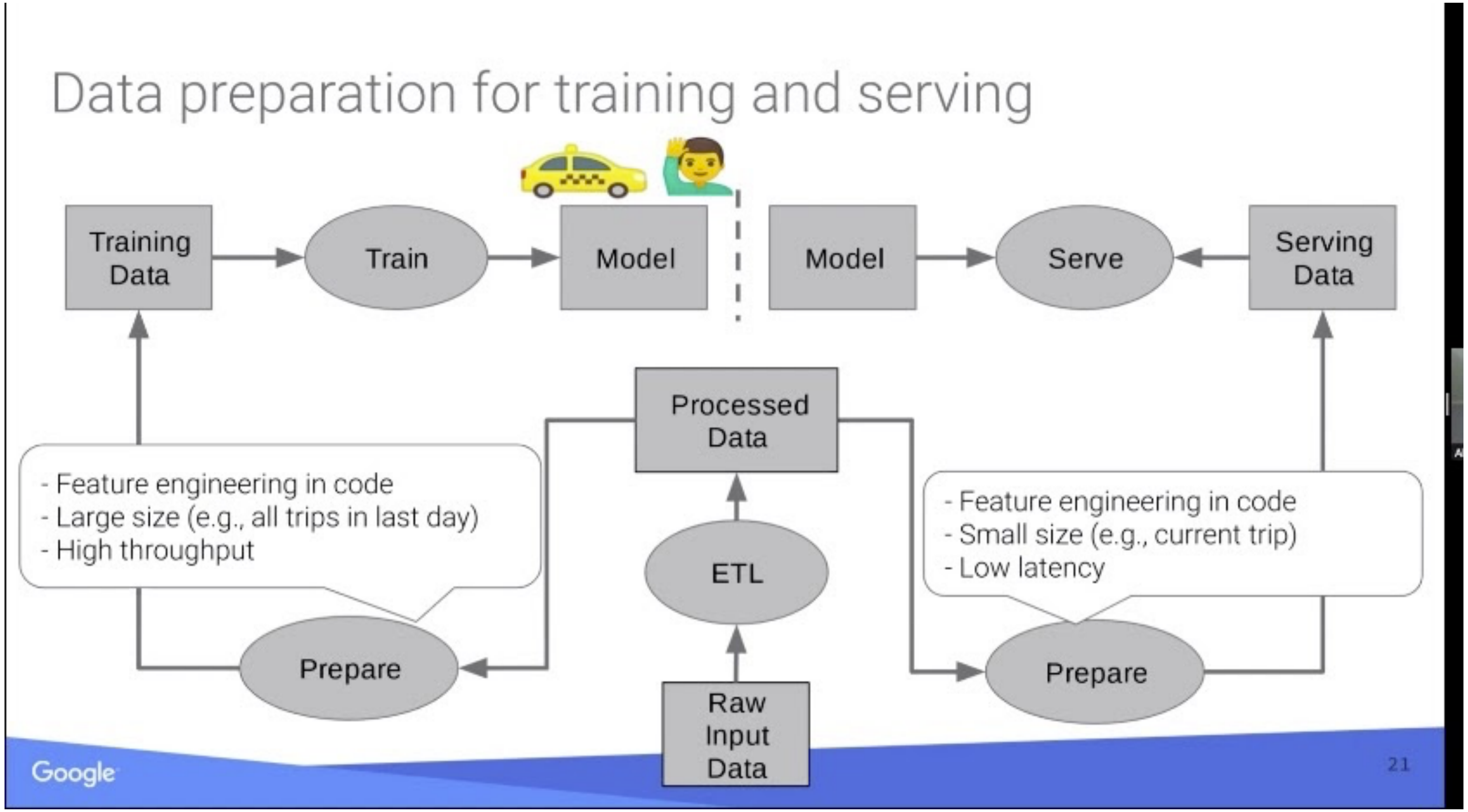
Plaquette Formation Quantmetry

Big data meets Machine learning



From EDBT 2020 Keynote 1

Typical ML pipeline



From EDBT 2020 Keynote 1

Why a Spark-based solution?

- Streamlined integration with data-prep pipeline
- Distributed processing
  - Manage large datasets
  - Parallel training large set of parameters
- Native Stream processing
  - Prediction in continuous for unseen data
- Main-memory and caching capabilities
- Existence of High-level APIs (e.g. Dataset)
  - backed with highly efficient lower API e.g. RDD

Spark Machine Learning Library

- Largely inspired by / relying on existing centralized libraries
  - Feature extraction, transformation and selection from Sikcit-Learn
  - Natlib library …
- Two layers
  - An Dataset-based library exposed to the end-user
  - An RDD-based library encapsulating major alogrithm
- Model selection and tuning
  - Grid search, cross validation

Feature extraction, transformation and selection

- Real data comes uses a rich set of types
  - text, number, booleans, timestamps, ...
- ML algorithms expect numeric data
  - Ex. libsvm
- Encoding real data may be challenging
  - Fixing/cleaning dirty data, deal with missing values, outliers
  - Collect additional data
  - Decide whether a feature is categorical or continuous
- Model inference (and prediction) quality relies on the encoding!
  - Recall the garbage-in garbage-out principle

Spark ML main ingredients

- Transformer   Transformer
  - Create features or perform prediction (using a trained model)
  - Endowed with transform() method
  - Ex. feature transformation:
    - Input : Dataframe with n columns of numbers -> a dataframe with one column of vectors
  - Ex. prediction
    - Input : Dataframe with a features vector -> the input dataframe augmented with predictions column
- Estimator   Estimator
  - trains an ML model on the data (ex. logistic regression)
  - The fit() method

Spark ML main ingredients

- Parameter
  - A uniform class for describing parameters passed to an estimator or extracted from a transformer
  - Ex. for decision tree inference: the number of nodes, the selection criterion (info gain or Gini index), ..
- Pipeline
  - Sequence of stages performing a specific ML algorithm
  - A stage = an estimator or a transformers
  - Linear only, DAG-based on the way?
- Evaluator
  - Several metrics (MAE, RMSE, …)

Spark ML Data model

- Builds on the Dataset
  - Basic types: boolean, numeric (integer, decimal, …), String, null, timestamp
  - Complex types: arrays, structures, maps
  - User-defined types
- Support for the Vector type
  - Part of the org.apache.spark.ml.linalg package
  - Seen as a UDT
  - An n-dimensional structure of *Doubles*
  - Possibility to use the **dense** or the **sparse** variant
  - And to convert dense to spare or vice versa

Dense vs Sparse Vectors

- Dense
  - Sequence of values [v1, v2, ….]
  - E.g [0,1,3,0]
- Sparse
  - Optimized storage by storing non-0 values only!
  - Tuple indicating
    - the vector size
    - indices of non-0 values
    - sequence of non-0 values
  - E.g (4, [1,2], [1,3])

Dense vs Sparse Vectors

```
import org.apache.spark.ml.linalg.{Vectors,Vector}

case class tuple (vec: Vector)
val sparse_sample = spark.createDataFrame(
    Seq( tuple(Vectors.dense(1.0, 1.0, 18.0) )
        , tuple(Vectors.dense(0.0, 2.0, 20.0) )
        , tuple(Vectors.dense(1.0, 0.0, 18.0).toSparse )
        , tuple(Vectors.dense(2.0, 3.0, 11.0).toSparse )
       )).toDF("input_vec")

sparse_sample.show(truncate=false)
```

```
+--------------------------+
|input_vec                 |
+--------------------------+
|[1.0,1.0,18.0]            |
|[0.0,2.0,20.0]            |
|(3,[0,2],[1.0,18.0])      |
|(3,[0,1,2],[2.0,3.0,11.0])|
+--------------------------+
```

Spark ML algorithms

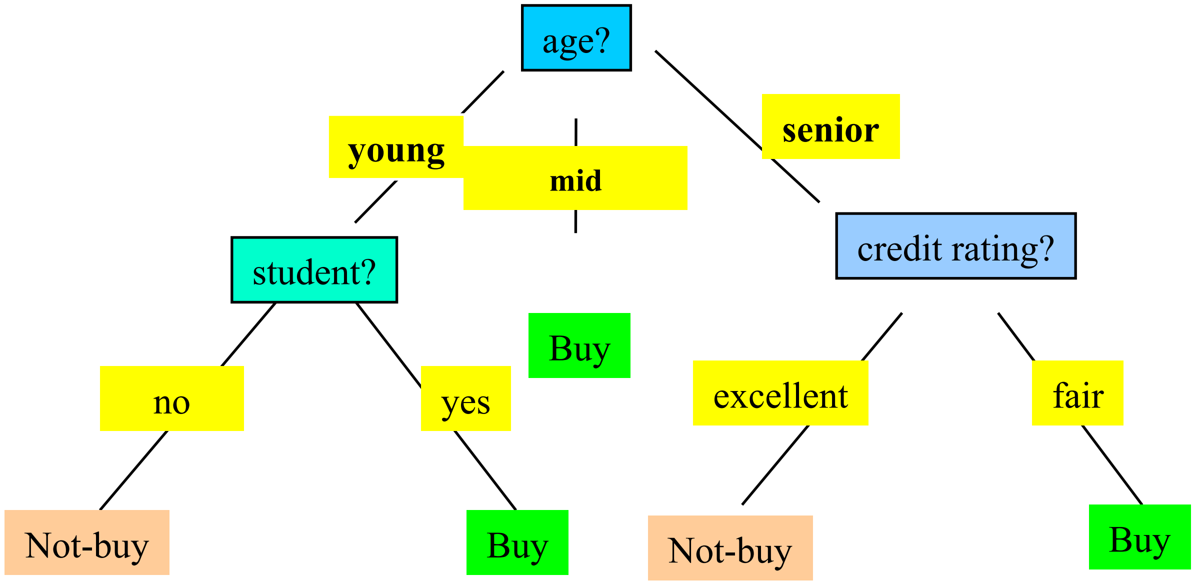
- Common algorithms for supervised and unsupervised learning
- Classification
  - Tree-based family: decision tree, random forest, gradient-boosted
  - Linear SVM, logistic regression, …
- Regression
  - Linear regression
  - Tree-based (same as above for regression)
- Clustering
  - K-means, LDA, ..
- Frequent pattern mining

Case study: decision tree inference

Original data

age	income	student	credit_rating	buys_computer
young	high	no	fair	no
young	high	no	excellent	no
middle	high	no	fair	yes
senior	medium	no	fair	yes
senior	low	yes	fair	yes
senior	low	yes	excellent	no
middle	low	yes	excellent	yes
young	medium	no	fair	no
young	low	yes	fair	yes
senior	medium	yes	fair	yes
young	medium	yes	excellent	yes
middle	medium	no	excellent	yes
middle	high	yes	fair	yes
senior	medium	no	excellent	no

Training data set: Who buys computer?



Adapted from
Data Mining: concepts and techniques by
J.Han, M. Kamber et J. Pei

Case study: decision tree inference

Original data

age	income	student	credit_rating	buys_computer
young	high	no	fair	no
young	high	no	excellent	no
middle	high	no	fair	yes
senior	medium	no	fair	yes
senior	low	yes	fair	yes
senior	low	yes	excellent	no
middle	low	yes	excellent	yes
young	medium	no	fair	no
young	low	yes	fair	yes
senior	medium	yes	fair	yes
young	medium	yes	excellent	yes
middle	medium	no	excellent	yes
middle	high	yes	fair	yes
senior	medium	no	excellent	no

Encoded features (what Spark ML expects)

```
 |-- features: vector (nullable = true)
 |-- indexed_label: double (nullable = false)

+-----------------+-------------+
|         features|indexed_label|
+-----------------+-------------+
|[1.0,1.0,0.0,0.0]|          1.0|
|[1.0,1.0,0.0,1.0]|          1.0|
|[2.0,1.0,0.0,0.0]|          0.0|
|      (4,[],[])|          0.0|
|[0.0,2.0,1.0,0.0]|          0.0|
|[0.0,2.0,1.0,1.0]|          1.0|
|[2.0,2.0,1.0,1.0]|          0.0|
|    (4,[0],[1.0])|          1.0|
|[1.0,2.0,1.0,0.0]|          0.0|
|    (4,[2],[1.0])|          0.0|
|[1.0,0.0,1.0,1.0]|          0.0|
|[2.0,0.0,0.0,1.0]|          0.0|
|[2.0,1.0,1.0,0.0]|          0.0|
|    (4,[3],[1.0])|          1.0|
+-----------------+-------------+
```

Case study: decision tree inference

age	income	student	credit_rating	buys_computer
young	high	no	fair	no
young	high	no	excellent	no
middle	high	no	fair	yes
senior	medium	no	fair	yes
senior	low	yes	fair	yes
senior	low	yes	excellent	no
middle	low	yes	excellent	yes
young	medium	no	fair	no
young	low	yes	fair	yes
senior	medium	yes	fair	yes
young	medium	yes	excellent	yes
middle	medium	no	excellent	yes
middle	high	yes	fair	yes
senior	medium	no	excellent	no

data.csv


```
|-- features: vector (nullable = true)
|-- indexed_label: double (nullable = false)

+-----------------+-------------+
|         features|indexed_label|
+-----------------+-------------+
|[1.0,1.0,0.0,0.0]|          1.0|
|[1.0,1.0,0.0,1.0]|          1.0|
|[2.0,1.0,0.0,0.0]|          0.0|
|        (4,[],[])|          0.0|
|[0.0,2.0,1.0,0.0]|          0.0|
|[0.0,2.0,1.0,1.0]|          1.0|
|[2.0,2.0,1.0,1.0]|          0.0|
|    (4,[0],[1.0])|          1.0|
|[1.0,2.0,1.0,0.0]|          0.0|
|    (4,[2],[1.0])|          0.0|
|[1.0,0.0,1.0,1.0]|          0.0|
|[2.0,0.0,0.0,1.0]|          0.0|
|[2.0,1.0,1.0,0.0]|          0.0|
|    (4,[3],[1.0])|          1.0|
+-----------------+-------------+
```

String Indexer

- Maps a column of strings to a column of longs corresponding to indices
- Indices from [0, numLabels[
- 4 ordering options:
  - Descending or ascending combined with frequency or alphabetical
- 3 possible outcomes for unseen labels:
  - Raise exception (default)
  - Skip row
  - Keep row with label = numLabels
- passing arg to *setHandleInvalid( )*

String Indexer illustrated

age	income	student	credit_rating	buys_computer
young	high	no	fair	no
young	high	no	excellent	no
middle	high	no	fair	yes
senior	medium	no	fair	yes
senior	low	yes	fair	yes
senior	low	yes	excellent	no
middle	low	yes	excellent	yes
young	medium	no	fair	no
young	low	yes	fair	yes
senior	medium	yes	fair	yes
young	medium	yes	excellent	yes
middle	medium	no	excellent	yes
middle	high	yes	fair	yes
senior	medium	no	excellent	no

data.csv

```
import org.apache.spark.ml.feature.StringIndexer

val field = "age"
val ageIndexer =  new StringIndexer()
        .setInputCol(field)
        .setOutputCol("indexed_"+field)

val ageIndexed = ageIndexer.fit(data).transform(data)
ageIndexed.show()
```

‘trains’ an estimator
based on the frequencies

```
+------+------+-------+-------------+-----+-----------+
|   age|income|student|credit_rating|label|indexed_age|
+------+------+-------+-------------+-----+-----------+
| young|  high|     no|         fair|   no|        1.0|
| young|  high|     no|    excellent|   no|        1.0|
|middle|  high|     no|         fair|  yes|        2.0|
|senior|medium|     no|         fair|  yes|        0.0|
|senior|   low|    yes|         fair|  yes|        0.0|
|senior|   low|    yes|    excellent|   no|        0.0|
|middle|   low|    yes|    excellent|  yes|        2.0|
| young|medium|     no|         fair|   no|        1.0|
| young|   low|    yes|         fair|  yes|        1.0|
|senior|medium|    yes|         fair|  yes|        0.0|
| young|medium|    yes|    excellent|  yes|        1.0|
|middle|medium|     no|    excellent|  yes|        2.0|
|middle|  high|    yes|         fair|  yes|        2.0|
|senior|medium|     no|    excellent|   no|        0.0|
+------+------+-------+-------------+-----+-----------+
```

```
age: string
income: string
student: string
credit_rating: string
label: string
```

schema

age	Count(*)	*Label*
Senior	5	0.0
Young	5	1.0
Middle	4	2.0

```
age: string
income: string
student: string
credit_rating: string
label: string
indexed_age: double
```

schema

IndexToString

- Retrieves the original labels from a string indexed column
- Helps in explainaing the inferred models

```
import org.apache.spark.ml.feature.IndexToString

val inputColSchema = ageIndexed.schema(ageIndexer.getOutputCol)

val ageConverter = new IndexToString()
  .setInputCol(ageIndexer.getOutputCol)
  .setOutputCol("originalAge")

val ageConverted = ageConverter.transform(ageIndexed)
ageConverted.show()
```

No training, simply back-transformation

age	indexed_age	originalAge
young	1.0	young
young	1.0	young
middle	2.0	middle
senior	0.0	senior
senior	0.0	senior
senior	0.0	senior
middle	2.0	middle
young	1.0	young
young	1.0	young
senior	0.0	senior
young	1.0	young
middle	2.0	middle
middle	2.0	middle
senior	0.0	senior

OneHot Encoder

- Maps categorical features to a binary vector indicating the presence of a value for a given feature
- Useful for algorithms requiring continuous features like Logistic Regression
- It is possible to merge several *oneHotEncoded* features using *VectorAssembler*
- Pre-requisite: index categorical features using *StringIndexer*

OneHot Encoder illustrated

```
import org.apache.spark.ml.feature.OneHotEncoder

val oneHotEncoder = new OneHotEncoder()
  .setInputCols(Array("indexed_age", "indexed_income"))
  .setOutputCols(Array("category_age", "category_income"))
  .setDropLast(false)

val encoded = oneHotEncoder.fit(data).transform(data)
```

indexed_age	indexed_income	category_age	category_income
1.0	1.0	(3,[1],[1.0])	(3,[1],[1.0])
1.0	1.0	(3,[1],[1.0])	(3,[1],[1.0])
2.0	1.0	(3,[2],[1.0])	(3,[1],[1.0])
0.0	0.0	(3,[0],[1.0])	(3,[0],[1.0])
0.0	2.0	(3,[0],[1.0])	(3,[2],[1.0])
0.0	2.0	(3,[0],[1.0])	(3,[2],[1.0])
2.0	2.0	(3,[2],[1.0])	(3,[2],[1.0])
1.0	0.0	(3,[1],[1.0])	(3,[0],[1.0])
1.0	2.0	(3,[1],[1.0])	(3,[2],[1.0])
0.0	0.0	(3,[0],[1.0])	(3,[0],[1.0])
1.0	0.0	(3,[1],[1.0])	(3,[0],[1.0])
2.0	0.0	(3,[2],[1.0])	(3,[0],[1.0])
2.0	1.0	(3,[2],[1.0])	(3,[1],[1.0])
0.0	0.0	(3,[0],[1.0])	(3,[0],[1.0])

Vector assembler

- Combines a list of columns C1,…, Cn into a single column of vectors obtained by concatenating vallues/vecotrs in C_i

indexed_age	indexed_income
1.0	1.0
1.0	1.0
2.0	1.0
0.0	0.0
0.0	2.0
0.0	2.0
2.0	2.0
1.0	0.0
1.0	2.0
0.0	0.0
1.0	0.0
2.0	0.0
2.0	1.0
0.0	0.0

```
import org.apache.spark.ml.feature.VectorAssembler

val vecAssembler = new VectorAssembler()
                 .setInputCols(Array("indexed_age"
                               ,"indexed_income"))
                 .setOutputCol("ageIncomeVec")

val ageIncomeIndexedVec = vecAssembler
                          .transform(ageIncomeIndexed)
ageIncomeIndexedVec. show()
```

Transformation only

ageIncomeVec
[1.0,1.0]
[1.0,1.0]
[2.0,1.0]
(2,[],[])
[0.0,2.0]
[0.0,2.0]
[2.0,2.0]
[1.0,0.0]
[1.0,2.0]
(2,[],[])
[1.0,0.0]
[2.0,0.0]
[2.0,1.0]
(2,[],[])

Vector Indexer

- Crucial for models relying on categorical features Decision trees
- Discriminate categorical from continuous features in a vector
- Index categorical features using 0-based indexes
- Input: col: Vector, maxCategories: int
- If # d-values( ) <= maxCat
  - then the feature is categorical
  - Otherwise, the feature is continuous

Vector Indexer Illustrated

```
+--------------+
|     input_vec|
+--------------+
|[1.0,1.0,18.0]|
|[0.0,2.0,20.0]|
|[1.0,0.0,18.0]|
|[2.0,3.0,11.0]|
+--------------+
```

```
import org.apache.spark.ml.feature.VectorIndexer

val vecIndexer = new VectorIndexer()
  .setInputCol("input_vec")
  .setOutputCol("output_vec")
  .setMaxCategories(3)

val vecIndexerModel = vecIndexer.fit(sample)

val sample_indexed = vecIndexerModel.transform(sample)
sample_indexed.show()
```

```
categorical features: 0, 2

+--------------+-------------+
|     input_vec|   output_vec|
+--------------+-------------+
|[1.0,1.0,18.0]|[1.0,1.0,1.0]|
|[0.0,2.0,20.0]|[0.0,2.0,2.0]|
|[1.0,0.0,18.0]|[1.0,0.0,1.0]|
|[2.0,3.0,11.0]|[2.0,3.0,0.0]|
+--------------+-------------+
```

indexed

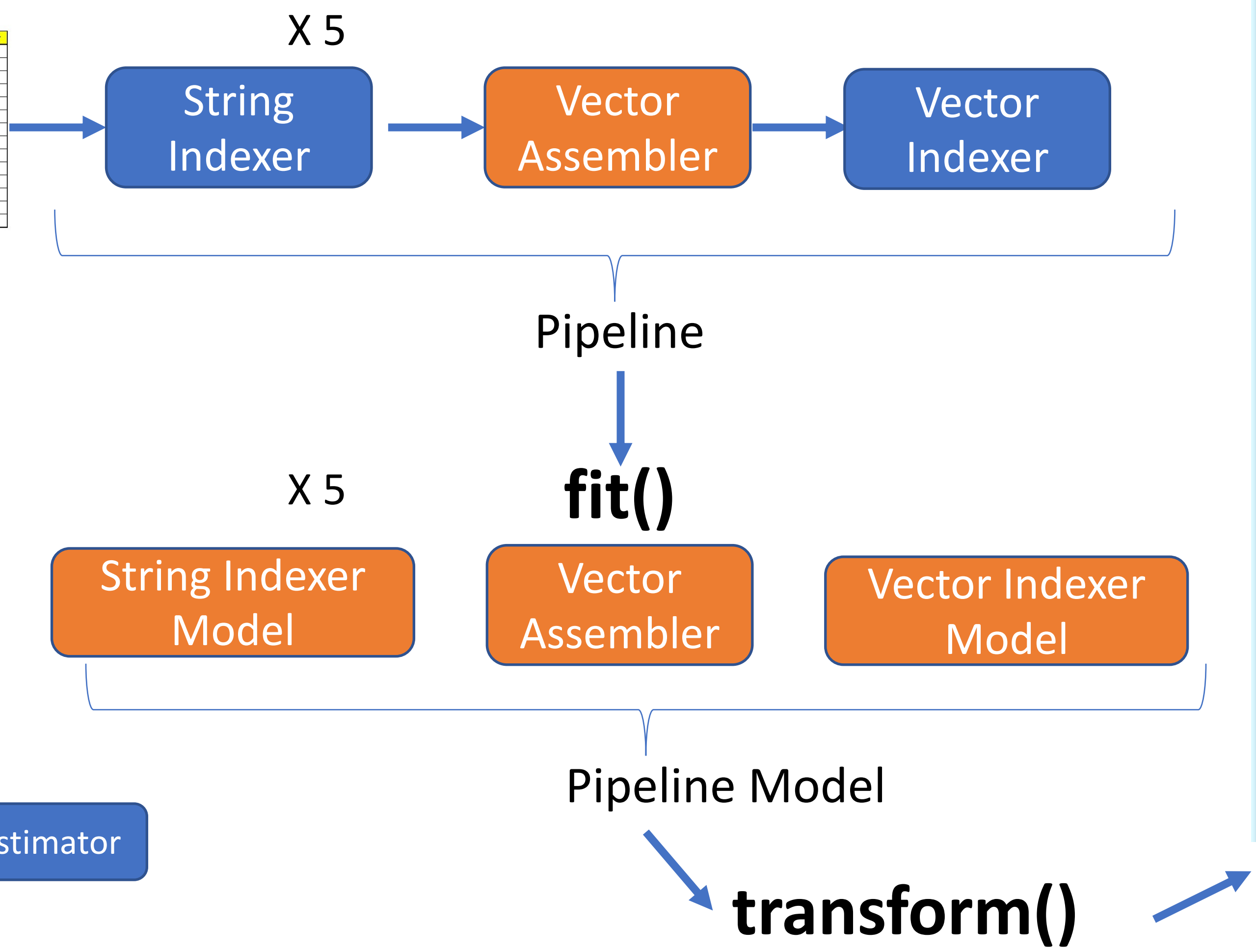
Pipelines

- Inspired by SickitLearn pipeline
- Used for combining several algorithms into one workflow
  - setStages(Array[<: PipelineStage])
- Each algorithm is either a transformer or an estimator
- P = op1, op2, …, op*n*
- Invoking fit() for P
  - Sequential processing of op*i s*
  - if op*i* is an estimator then invoke fit() for op*i*
  - Else // op*i* is a transformer
  - invoke transform()

Pipelines illustrated

age	income	student	credit_rating	buys_computer
young	high	no	fair	no
young	high	no	excellent	no
middle	high	no	fair	yes
senior	medium	no	fair	yes
senior	low	yes	fair	yes
senior	low	yes	excellent	no
middle	low	yes	excellent	yes
young	medium	no	fair	no
young	low	yes	fair	yes
senior	medium	yes	fair	yes
young	medium	yes	excellent	yes
middle	medium	no	excellent	yes
middle	high	yes	fair	yes
senior	medium	no	excellent	no

data.csv



Transformer | Estimator

Legend

```
|-- features: vector (nullable = true)
|-- indexed_label: double (nullable = false)

+-----------------+-------------+
|         features|indexed_label|
+-----------------+-------------+
|[1.0,1.0,0.0,0.0]|          1.0|
|[1.0,1.0,0.0,1.0]|          1.0|
|[2.0,1.0,0.0,0.0]|          0.0|
|        (4,[],[])|          0.0|
|[0.0,2.0,1.0,0.0]|          0.0|
|[0.0,2.0,1.0,1.0]|          1.0|
|[2.0,2.0,1.0,1.0]|          0.0|
|    (4,[0],[1.0])|          1.0|
|[1.0,2.0,1.0,0.0]|          0.0|
|    (4,[2],[1.0])|          0.0|
|[1.0,0.0,1.0,1.0]|          0.0|
|[2.0,0.0,0.0,1.0]|          0.0|
|[2.0,1.0,1.0,0.0]|          0.0|
|    (4,[3],[1.0])|          1.0|
+-----------------+-------------+
```

Pipelines illustrated

```
/*index the features attributes*/
val stringIndexerAllatts = data.columns.filterNot(_.contains(labName))
.map
{field =>
    new StringIndexer()
    .setInputCol(field)
    .setOutputCol("indexed_" + field)
}

stringIndexerAllatts: Array[org.apache.spark.ml.feature.StringIndexer] =
Array(strIdx_5ceff7f452bb, strIdx_ede8efa81cb2, strIdx_64c415600d11, strIdx_760f0
5df92f4)
```

Pipelines illustrated

```
import  org.apache.spark.ml.Pipeline
val pipeline = new Pipeline()
                .setStages(Array(stringIndexerLabel)++stringIndexerAllatts++
Array(vectorAssembler,vectorIndexer))
val train_data = pipeline.fit(data).transform(data).select("features","indexed_label")
```

age	income	student	credit_rating	buys_computer
young	high	no	fair	no
young	high	no	excellent	no
middle	high	no	fair	yes
senior	medium	no	fair	yes
senior	low	yes	fair	yes
senior	low	yes	excellent	no
middle	low	yes	excellent	yes
young	medium	no	fair	no
young	low	yes	fair	yes
senior	medium	yes	fair	yes
young	medium	yes	excellent	yes
middle	medium	no	excellent	yes
middle	high	yes	fair	yes
senior	medium	no	excellent	no

data.csv

```
root
 |-- features: vector (nullable = true)
 |-- indexed_label: double (nullable = false)

+-----------------+-------------+
|         features|indexed_label|
+-----------------+-------------+
|[1.0,1.0,0.0,0.0]|          1.0|
|[1.0,1.0,0.0,1.0]|          1.0|
|[2.0,1.0,0.0,0.0]|          0.0|
|       (4,[],[])|          0.0|
```

Decision Tree inference

- Expects a DF with
  - label column (target variable)
  - Features column (vector of indexed values)
- Exploits existing metadata :
  - maxCategories of the indexed vector to decide how to deal with features
  - Two kinds of conditions
    - Categorical features -> value equality
    - Continuous features -> interval comparison
- Multi-class/multi-label
- The inferred tree is binary, used for prediction

Decision Tree inference illustrated

```
import org.apache.spark.ml.classification.DecisionTreeClassificationModel
import org.apache.spark.ml.classification.DecisionTreeClassifier
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

val dt = new DecisionTreeClassifier()
  .setLabelCol("indexed_label")
  .setFeaturesCol("features")

val dtModel = dt.fit(train_data)
println(s"Learned classification tree model:\n ${dtModel.toDebugString}")
```

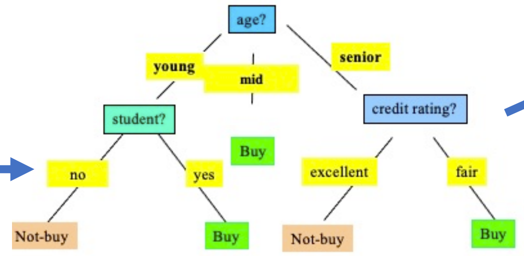
Decision Tree inference illustrated

```
root
 |-- features: vector (nullable = true)
 |-- indexed_label: double (nullable = false)

+-----------------+-------------+
|         features|indexed_label|
+-----------------+-------------+
|[1.0,1.0,0.0,0.0]|          1.0|
|[1.0,1.0,0.0,1.0]|          1.0|
|[2.0,1.0,0.0,0.0]|          0.0|
|        (4,[],[])|          0.0|
```

Decision Tree Classifier

fit()



Decision Tree Model

Unseen data

```
+-----------------+
|         features|
+-----------------+
|[1.0,0.0,0.0,1.0]|
|    (4,[2],[1.0])|
+-----------------+
```

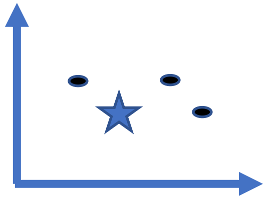
tranform()

```
+-----------------+----------+
|features         |prediction|
+-----------------+----------+
|[1.0,0.0,0.0,1.0]|1.0       |
|(4,[2],[1.0])    |0.0       |
+-----------------+----------+
```

Decision Tree inference illustrated

```
DecisionTreeClassificationModel: uid=dtc_cc9e8a41bf40, depth=4, numNodes=13, numClasses=2, numFeatures=4
 If (feature 0 in {2.0})
  Predict: 0.0
 Else (feature 0 not in {2.0})
  If (feature 2 in {1.0})
   If (feature 3 in {0.0})
    Predict: 0.0
   Else (feature 3 not in {0.0})
    If (feature 0 in {1.0})
     Predict: 0.0
    Else (feature 0 not in {1.0})
     Predict: 1.0
  Else (feature 2 not in {1.0})
   If (feature 0 in {0.0})
    If (feature 3 in {0.0})
     Predict: 0.0
    Else (feature 3 not in {0.0})
     Predict: 1.0
   Else (feature 0 not in {0.0})
    Predict: 1.0
```

Model Selection and Tuning



- To derive the best model:
    - experiment several hyper-parameters
    - split data in several manners
- Grid Search class
    - trying different combinations of pre-set parameters
- CrossValidator class
    - Build different (train, test) candidates
- Use default evaluation metrics (e.g. areaUnderROC for classif)
- Extract the best model w.r.t. the defined metrics

Model Selection and Tuning

A DT classifier

```
import org.apache.spark.ml.tuning.{CrossValidator, ParamGridBuilder}

val dt_paramGrid = new ParamGridBuilder()
        .addGrid(dt.maxBins, Array(40,42)) //try with two values
        .addGrid(dt.minInstancesPerNode, Array(10,100)) //try with two values
        .build()
```

Values to experiment

A parameter
of the model

The Grid contains 2 X 2 = 4 configuration to run

Model Selection and Tuning

Previous Slide

```
//create k folds with k=5.
val cv = new CrossValidator()
        .setEstimator(dt)
        .setEstimatorParamMaps(dt_paramGrid)
        .setEvaluator(new RegressionEvaluator)
        .setNumFolds(5)
        .setParallelism(2)
```

Metric evaluator Class

Size of the split

How many cores are use

The Grid contains 2 X 2 = 4 configuration to run

There are 5 folds

```
val cvModel = cv.fit(ftdata)
```

20 DT are inferred

```
val bestModel = cvModel.bestModel
            .asInstanceOf[DecisionTreeRegressionModel]
```

Mini-projet 2

- Réalisation d’un pipeline ML pour la ***régression*** à l’aide des arbres de décision
- Objectif principal : réaliser le pipeline de bout en bout sur des données réelles
  - Dataset libre, Taille ~ 10 MB (sampling possible)
  - Sources possibles : Kaggle
  - Traiter le problème des données manquantes (null), aberrantes (outliers),…
- Préparation :
  - Collecte de statistiques sur qualité des données : valeurs aberrantes, valeurs manquantes, …
  - Elimination des attributs non pertinents (bcp de valeurs distinctes)
  - Transformation des valeurs (timestamps vers nombre / extraction de l’année ou année-mois, …)
  - Encodage des features

Mini-projet 2

- Itération 1 :
  - Cross validation avec 3 folds, grid search sur paramètres pertinents
  - Sélection du meilleur modèle
  - Analyses des métriques RMSE et MAE si dispo
- Iteration 2 : tentative d'amélioration de la précision
  - Elimination des valeurs aberrantes (si elles existent)
  - Imputation des valeurs manquantes (utiliser fonctions Spark ML)
  - Relancer l'inference et constater les nouvelles valeurs des métriques

Mini-projet 2

- Analyse
  - Comparer les résultats des deux itérations
  - Tenter d’expliquer la différence
  - Avis sur la libraire ML : difficultés rencontrées, aspects appréciés
- Modalités
  - Rendre un notebook + petit compte-rendu (suivant trame)
  - Date de remise : 10-12-2020