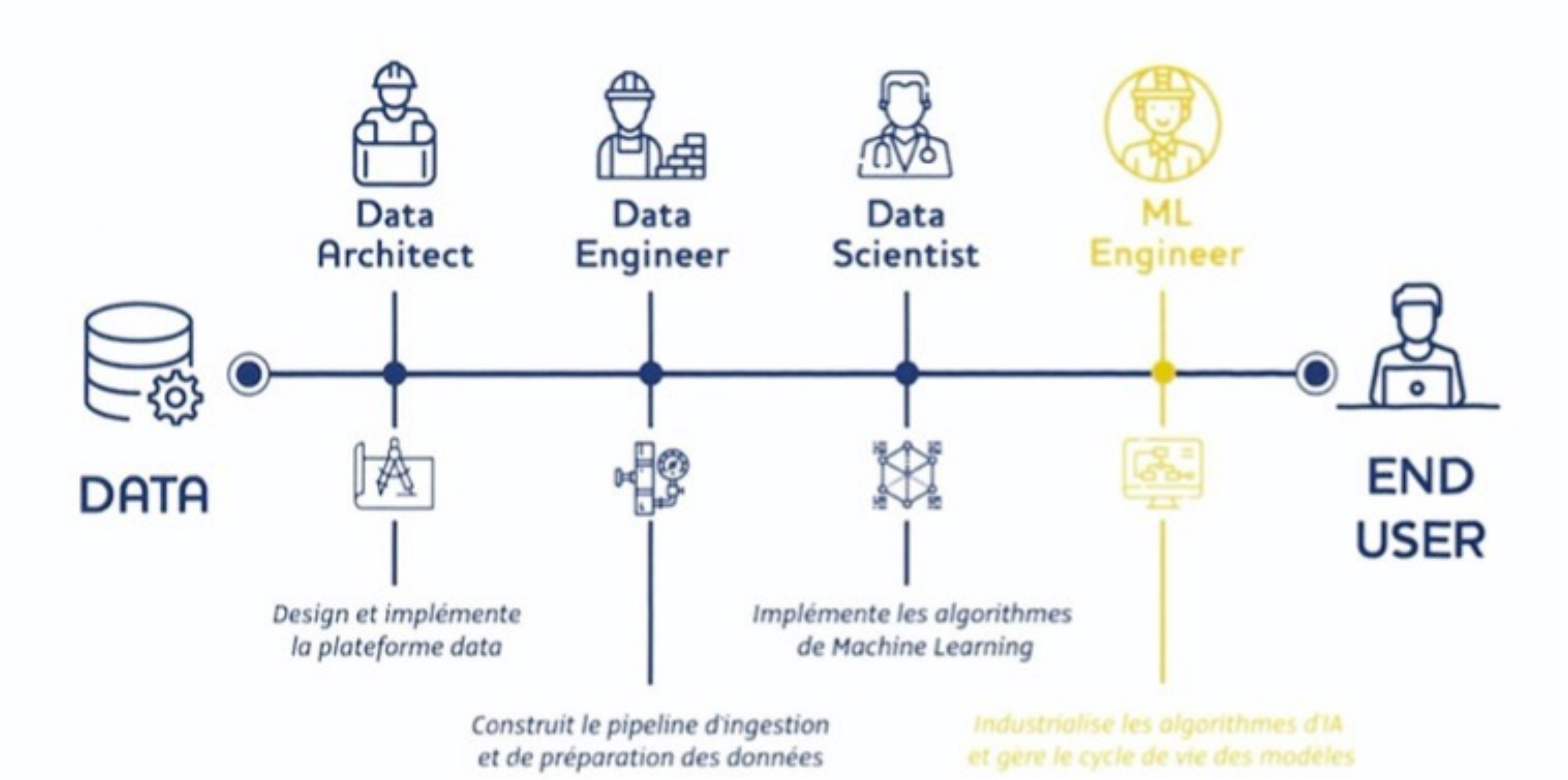
Préparation de données pour le ML en Spark

Master DAC – Bases de Données Large Echelle

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

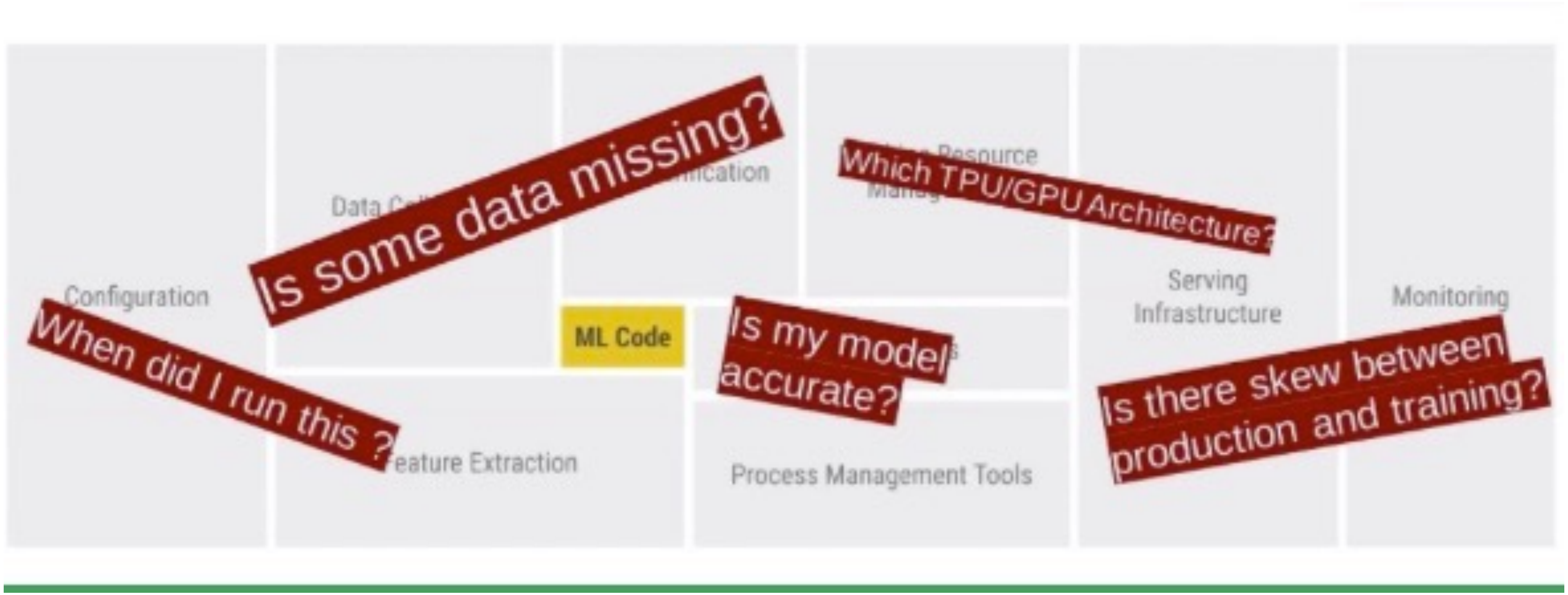
2021-2022

The data journey



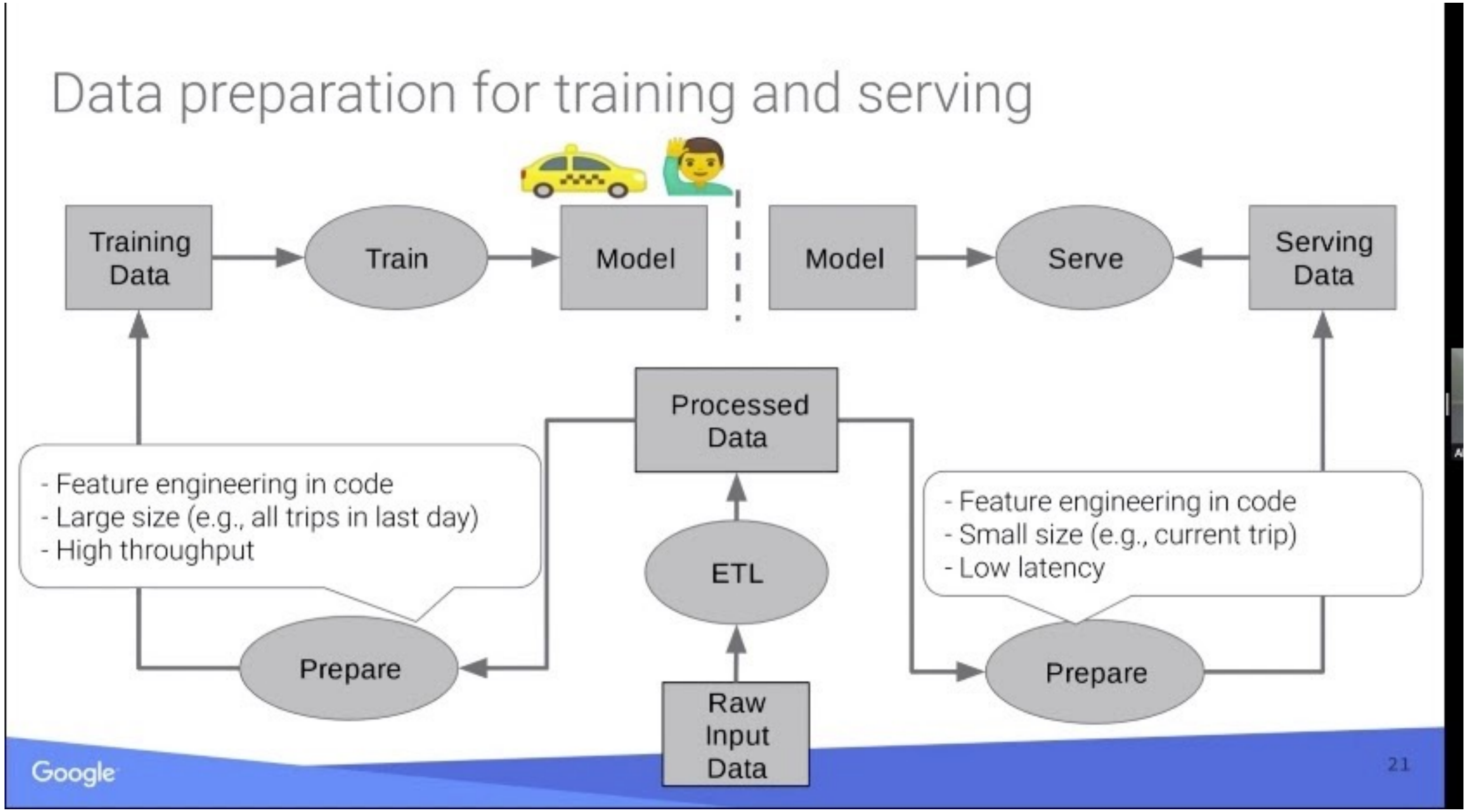
Plaquette Formation Quantmetry

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
  - A 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 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 data quality
  - Recall the garbage-in garbage-out principle

Spark ML main ingredients

- Transformer Transformer
    - Create features or perform prediction (using a trained model)
    - Invoke transform()
    - Ex. feature transformation:
        - Input : Dataframe with n columns of numbers -> a dataframe with one column of n-dimensional 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)
    - Invoke fit()

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 = either an estimator or a transformer
  - Usually Linear, DAG are also possible (specified using a topological order)
- 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!
  - Only interesting when the ratio of 0-values is very high
  - Tuple (s, I, V) indicating
    - s = the vector size
    - I = a sequence indicating the indices of non-0 values as per a dense vector
    - V = the sequence of non-0 values
  - E.g (4, [1,2], [1,3]) encodes [0,1,3,0]

Dense vs Sparse Vectors

```
from pyspark.ml.linalg import Vectors

vec1 = Vectors.dense(1.0, 1.0, 18.0)
vec2 = Vectors.dense(0.0, 2.0, 20.0)
vec3 = Vectors.sparse(3,[0.0,2.0],[1.0,18.0])
vec4 = Vectors.sparse(3,[0.0,1.2,2.0],[2.0,3.0,11.0])
vectors = spark.sparkContext.parallelize([vec1,vec2,vec3,vec4])
vectors.collect()
```

```
[DenseVector([1.0, 1.0, 18.0]),
DenseVector([0.0, 2.0, 20.0]),
SparseVector(3, {0: 1.0, 2: 18.0}),
SparseVector(3, {0: 2.0, 1: 3.0, 2: 11.0})]
```

Spark ML algorithms

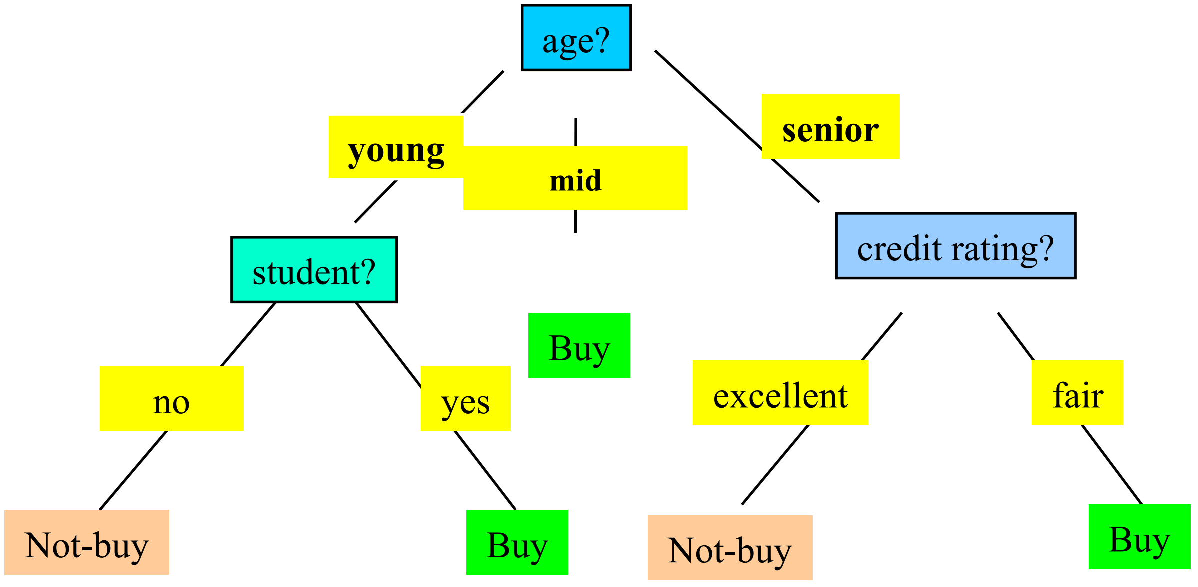
- Common algorithms for supervised and unsupervised learning
- Classification
  - Tree-based family: decision tree, random forest, gradient-boosted
  - Linear SVM, logistic regression, …
- Regression
  - Linear regression
  - Tree-based (same as above for regression)
- Clustering
  - K-means, LDA, ..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

```
from pyspark.ml.feature import StringIndexer

field = 'age'
age_indexer = StringIndexer(inputCol=field,\
outputCol='indexed_'+field)

df_age_idx = age_indexer.fit(data).transform(data)
```

```
+------+------+-------+-------------+-----+-----------+
|   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|
+------+------+-------+-------------+-----+-----------+
```

train an estimator based on the frequencies

```
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 explaining the inferred models

```
from pyspark.ml.feature import IndexToString

age_rev_indexer = IndexToString(inputCol=age_indexer.getOutputCol(),
outputCol='original_age')

df_orig_age =age_rev_indexer.transform(df_age_idx)
```

No training, simply back-transformation

age
young
young
middle
senior
senior
senior
middle
young
young
senior
young
middle
middle
senior

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

```
from pyspark.ml.feature import OneHotEncoder

age_onehotenc =
OneHotEncoder(inputCol=age_indexer.getOutputCol(),\
              outputCol='cat_age')
age_onehotenc.setDropLast(False)
df_age_onehot = age_onehotenc.fit(df_age_idx).transform(df_age_idx)
```

```
+-----------+-------------+
|indexed_age|      cat_age|
+-----------+-------------+
|        1.0|(3,[1],[1.0])|
|        1.0|(3,[1],[1.0])|
|        2.0|(3,[2],[1.0])|
|        0.0|(3,[0],[1.0])|
|        0.0|(3,[0],[1.0])|
|        0.0|(3,[0],[1.0])|
|        2.0|(3,[2],[1.0])|
|        1.0|(3,[1],[1.0])|
|        1.0|(3,[1],[1.0])|
|        0.0|(3,[0],[1.0])|
|        1.0|(3,[1],[1.0])|
|        2.0|(3,[2],[1.0])|
|        2.0|(3,[2],[1.0])|
|        0.0|(3,[0],[1.0])|
+-----------+-------------+
```

Vector assembler/slicer

- Assembler
  - Combines a list of columns C1,…, Cn into a single column of vectors obtained by concatenating values/vectors in C_i
- Slicer
  - Restricts to a set of columns, indicated by their coordinates

Vector assembler

```
-----------+--------------+
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|
-----------+--------------+
```

```
from pyspark.ml.feature import VectorAssembler

cols = ['indexed_age','indexed_income']
vec_assembler = VectorAssembler(inputCols= cols, \
outputCol= 'ageIncomeVec')

df_age_income_vec = vec_assembler.\
            transform(df_age_income_idx)
```

```
+------------+
|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

- Discriminate categorical from continuous features in a vector
- Index categorical features using 0-based indexes
- Input: col: Vector, maxCategories: int
- Set the maxCategories parameter
- If # d-values( ) <= maxCategories
  - 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]|
+--------------+
```

```
from pyspark.ml.feature import VectorIndexer

vecIndexer = VectorIndexer(inputCol='ageIncomeVec',\
             outputCol='indexed_ageIncomeVec',\
             maxCategories=3)

df_age_income_vec_idx = vecIndexer.fit(df_age_income_vec).\
transform(df_age_income_vec)
```

```
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]|
+--------------+-------------+
```

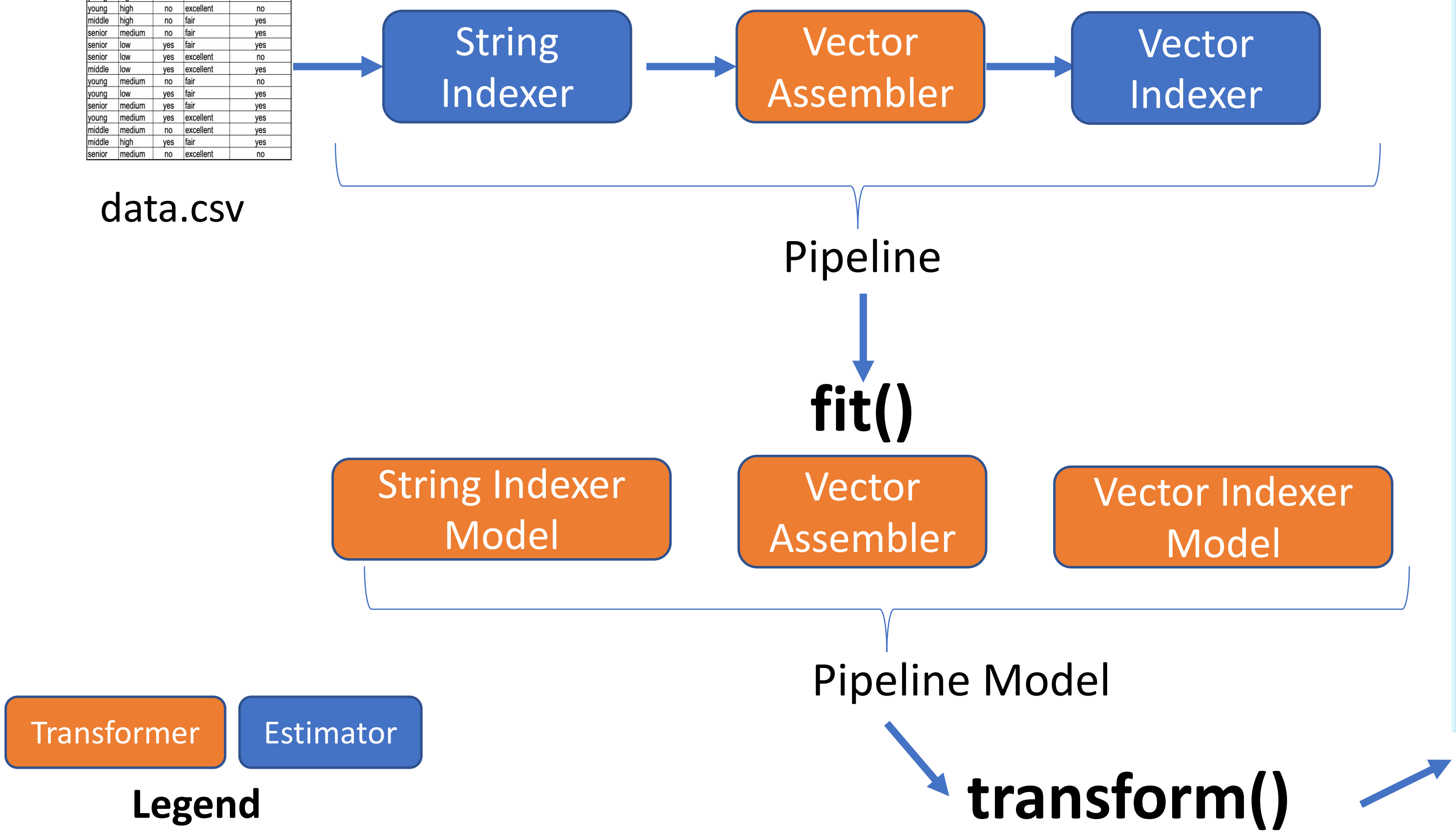
continuous feature

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

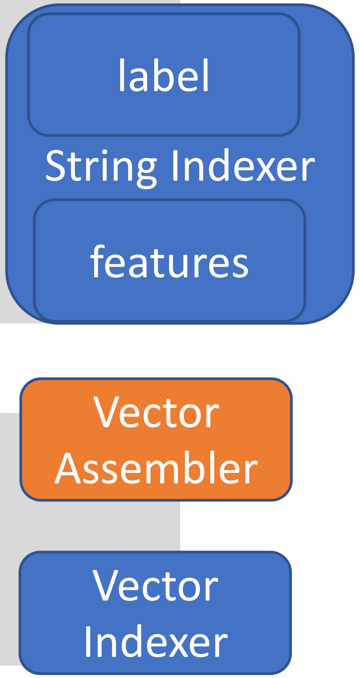



```
 |-- features: vector (nullable = true)
 |-- indexed_label: double (nullable = false)

+-----------------+-------------+
|         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

```
label = 'label'
features_col = data.columns
features_col.remove(label)

prefix = 'indexed_'


label_string_indexer = StringIndexer(inputCol=label, outputCol=prefix+label)

features_str_col = list(map(lambda c:prefix+c, features_col))
features_string_indexer = StringIndexer(inputCols=features_col,outputCols=features_str_col)
```

```
vec_assembler = VectorAssembler(inputCols= features_string_indexer.getOutputCols(),\
                outputCol= 'vector')

vec_indexer = VectorIndexer(inputCol='vector',outputCol='features’, maxCategories=3)
```



Pipelines illustrated

```
stages = [label_string_indexer,features_string_indexer,vec_assembler,vec_indexer]

from pyspark.ml import Pipeline

pipeline = Pipeline(stages = stages)
train_data = pipeline.fit(data).transform(data)
train_data.select("features","indexed_label").show()
```

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

```
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

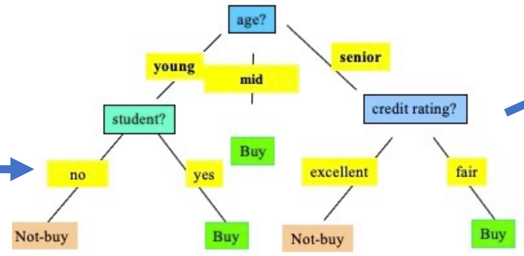
Decision Tree inference illustrated

```
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       |
+-----------------+----------+
```

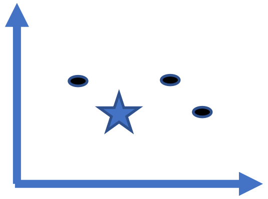
Decision Tree inference illustrated

```
from pyspark.ml.classification import DecisionTreeClassificationModel, DecisionTreeClassifier

dt = DecisionTreeClassifier(featuresCol="features", labelCol= "indexed_label")
dtModel = dt.fit(train_data)
```

```
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
```

```
DecisionTreeClassificationModel:
uid=DecisionTreeClassifier_5c99afcc20f4,
depth=4, numNodes=13, numClasses=2,
numFeatures=4
```

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

```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

dt_paramGrid = ParamGridBuilder()\
            .addGrid(dt.maxBins, [40,42])\
                        .addGrid(dt.minInstancesPerNode, [10,100]) \
                        .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.
cv = CrossValidator(estimator=dt, \
estimatorParamMaps=dt_paramGrid, \
evaluator=evaluatorPR, \
numFolds=5, \
parallelism=2)
```

Metric evaluator Class

Size of the split

How many cores are used

test

Train

The Grid contains 2 X 2 = 4 configuration to run

There are 5 folds

```
cvModel = cv.fit(train_data)
```

20 DT are inferred

```
bestModel = cvModel.bestModel
```

Closing remarks

- Pros
  - Efficiency thanks to the distributed evaluation
  - Static typing facilitates examining and reusing the pipeline
  - Metadata collection
- Cons
  - No fine-grained control on how to define categorical features
  - Impute of missing values limited to number (not possible for textual values)
- Possible extensions
  - Impute text values by using advanced NLP techniques (word2vec,…)
  - Parallel exploration of the search space to identify sub-set of relevant features
  - AutoML: automatic feature extraction, model selection and hyper-parameter search

Devoir maison : description

- 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
  - Comparer 2 stratégies en terme de précision des modèles obtenus
  - Dataset à choisir parmi une liste fournie, ou
  - libre, taille ~ 10 MB (sampling possible, utiliser Kaggle)
- Modalités
  - Rendre un notebook annoté avec explications
  - Date de remise : 11-11-2021 au soir

Devoir maison : réalisation

- Collecte de statistiques descriptive sur les données + taux de valeurs manquantes,
- Identification des attributs non pertinents
  - Ex. attributs avec nombre élevé de valeurs distinctes ou qui ne varient jamais
- Transformation des données lorsque possible
  - Ex. extraction composantes date depuis timestamp
- Imputation de valeurs manquantes
- 2 Itérations :
  - Iteration 1 : données d’origine encodées
  - Iteration 2 : données d’origine complétées, restreintes aux attributs pertinents, et avec attributs transformés (ex. dates) puis encodées
- Analyse comparative

Devoir maison : réalisation

- Chaque itération :
  - Cross validation avec 3 folds, grid search sur paramètres pertinents
  - Sélection du meilleur modèle
  - Analyses des métriques RMSE et MAE
- Analyse comparative
  - Evolution des valeurs des métriques
  - Evolution du vecteur des features importantes