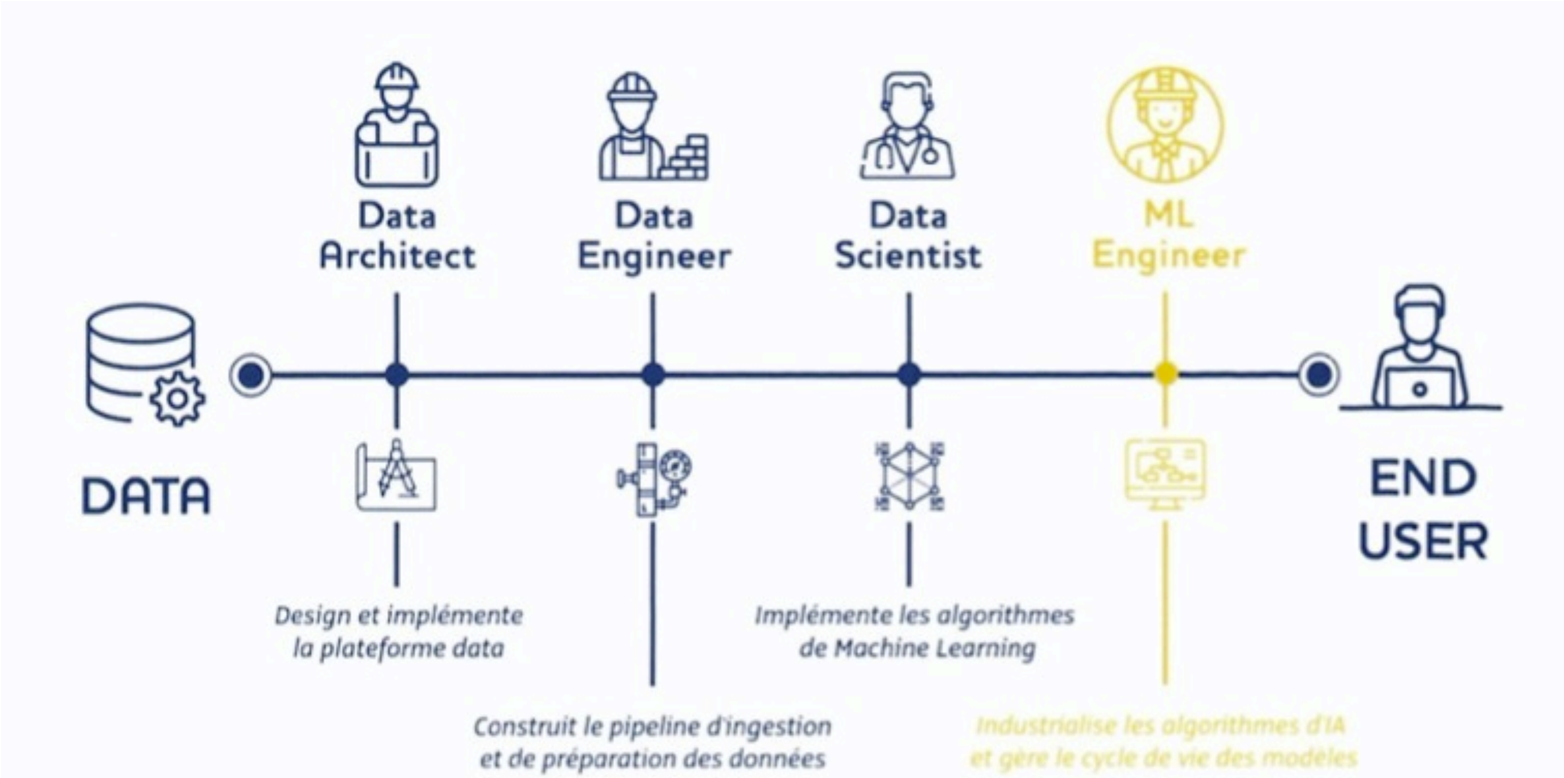
Préparation de données pour le ML en Spark

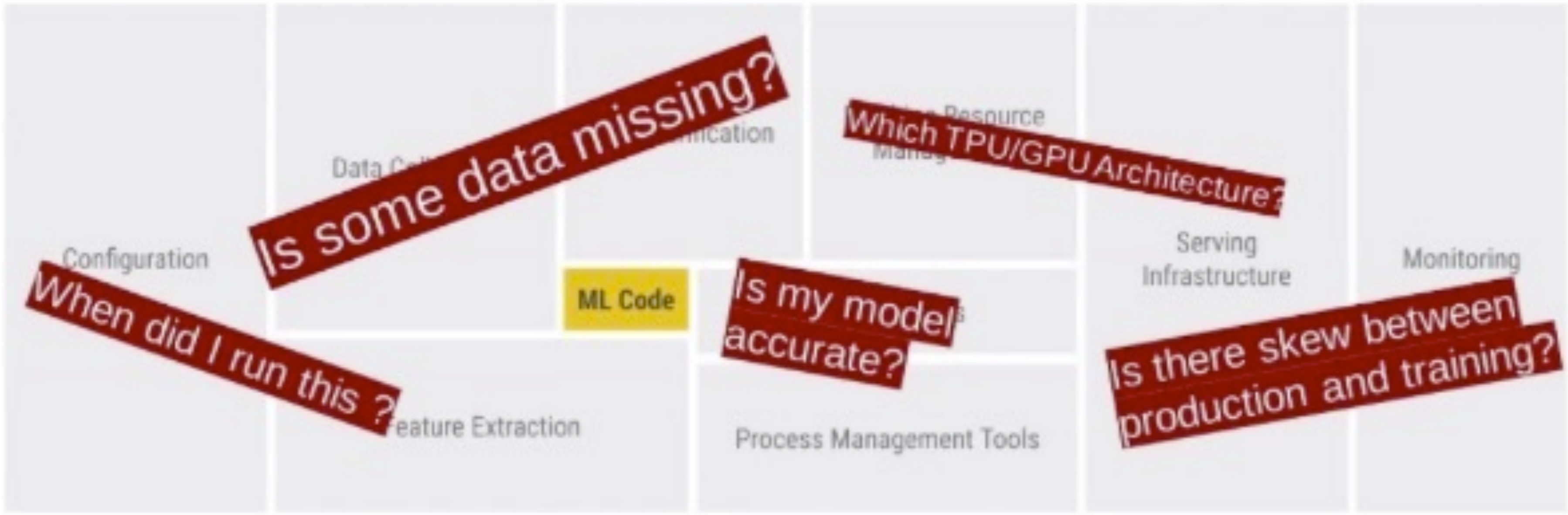
Mohamed-Amine Baazizi
mohamed-amine.baazizi@lip6.fr
December 2024

The data journey



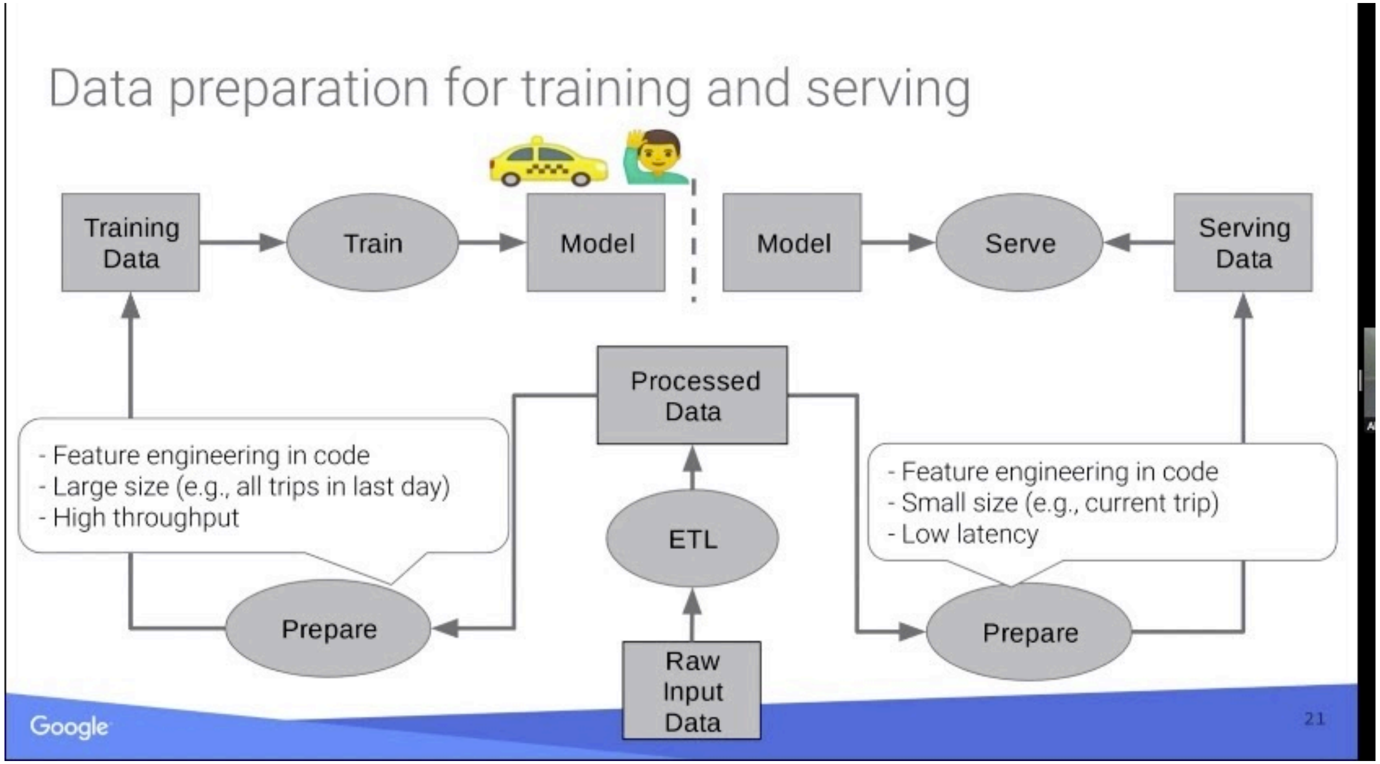
Plaquette Formation Quantmetry

Big data meets Machine learning



Sculley et al. Hidden. Technical Debt in Machine Learning Systems. NIPS 2015

A typical ML pipeline



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]

Spark ML algorithms

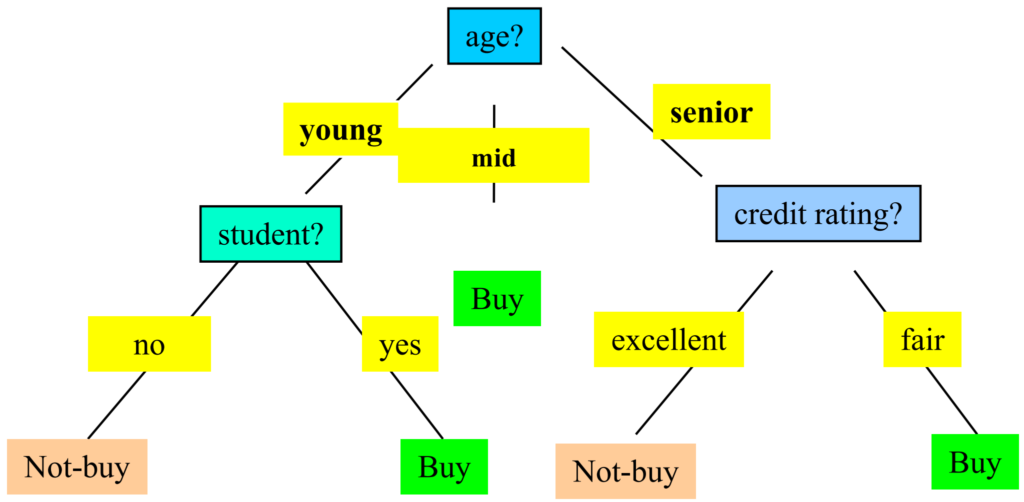
- Common algorithms for supervised and unsupervised learning
- Classification
  - Tree-based family: decision tree, random forest, gradient-boosted
  - Linear SVM, logistic regression, …
- Regression
  - Linear regression
  - Tree-based (same as above for regression)
- Clustering
  - K-means, LDA, ..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

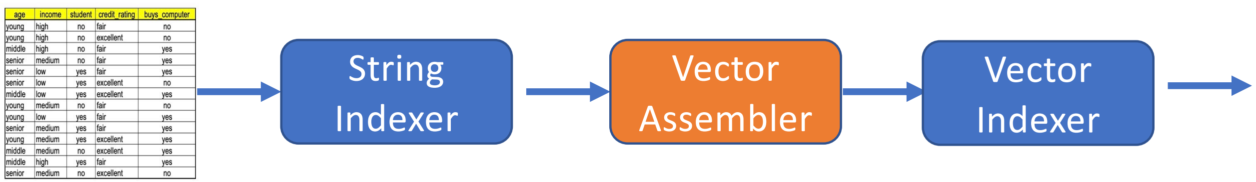
age	income	student	credit_rating	buys_computer
young	high	no	fair	no
young	high	no	excellent	no
middle	high	no	fair	yes
senior	medium	no	fair	yes
senior	low	yes	fair	yes
senior	low	yes	excellent	no
middle	low	yes	excellent	yes
young	medium	no	fair	no
young	low	yes	fair	yes
senior	medium	yes	fair	yes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



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

```
+------+------+-------+-------------+-----+-----------+
|   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
- No training, simply back-transformation

```
+------+-
|   age|-
+------+-
| young|
| young|
|middle|
|senior|r
|senior|
|senior|
|middle|
| young|r
| young|
|senior|r
| young|r
|middle|r
|middle|
|senior|r
+------+-
```

```
-+-----------+-----------+
l|indexed_age|originalAge|
-+-----------+-----------+
ɔ|        1.0|      young|
ɔ|        1.0|      young|
s|        2.0|     middle|
s|        0.0|     senior|
s|        0.0|     senior|
ɔ|        0.0|     senior|
s|        2.0|     middle|
ɔ|        1.0|      young|
s|        1.0|      young|
s|        0.0|     senior|
s|        1.0|      young|
s|        2.0|     middle|
s|        2.0|     middle|
ɔ|        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

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

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

categorical features: 0, 2

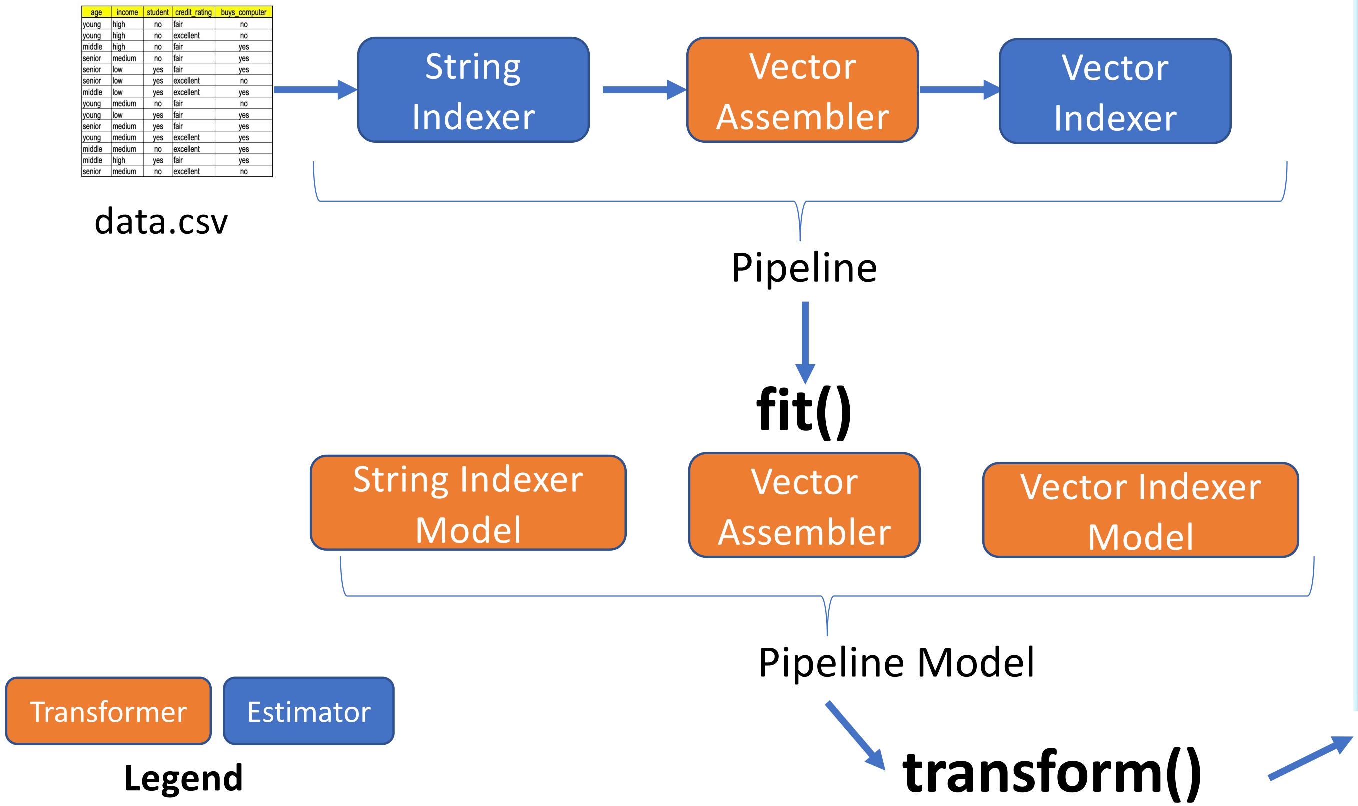
```
+--------------+-------------+
|     input_vec|   output_vec|
+--------------+-------------+
|[1.0,1.0,18.0]|[1.0,1.0,1.0]|
|[0.0,2.0,20.0]|[0.0,2.0,2.0]|
|[1.0,0.0,18.0]|[1.0,0.0,1.0]|
|[2.0,3.0,11.0]|[2.0,3.0,0.0]|
+--------------+-------------+
```

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




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

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

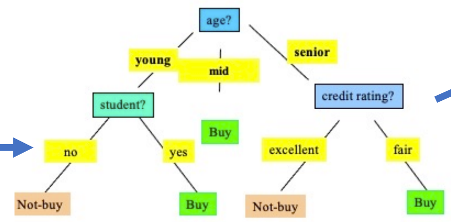
Decision Tree inference illustrated

```
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,[1],[])|          0.0|
```

Decision Tree Classifier

fit()



Decision Tree Model

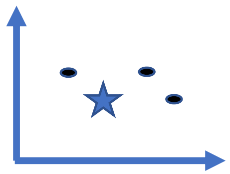
Unseen data

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

transform()

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

Model Selection and Tuning



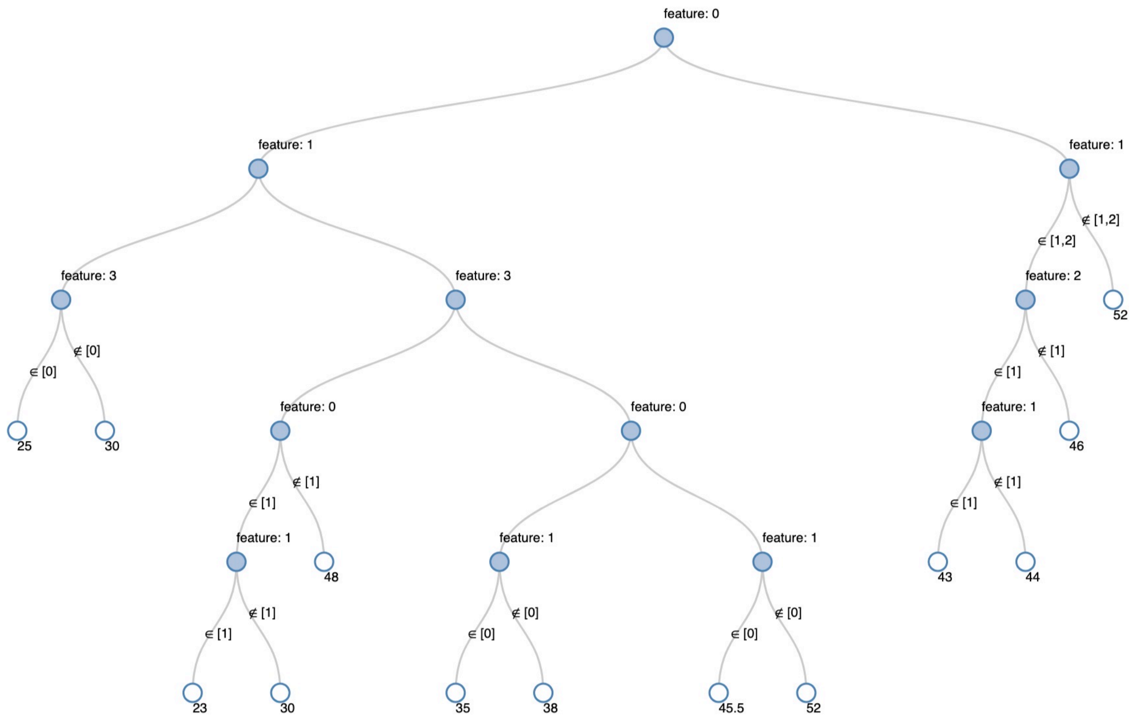
- To derive the best model:
  - experiment several hyper-parameters
  - split data in several manners
- Grid Search class
  - trying different combinations of pre-set parameters
- CrossValidator class
  - Build different (train, test) candidates
- Use default evaluation metrics (e.g. areaUnderROC for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

Planning

- Seance 1 : Decision Tree inference for classification on synthetic data
- Seance 2 : same as 1, real-data, model selection and tuning
- Seance 3 : Regression tree on synthetic and real data.

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



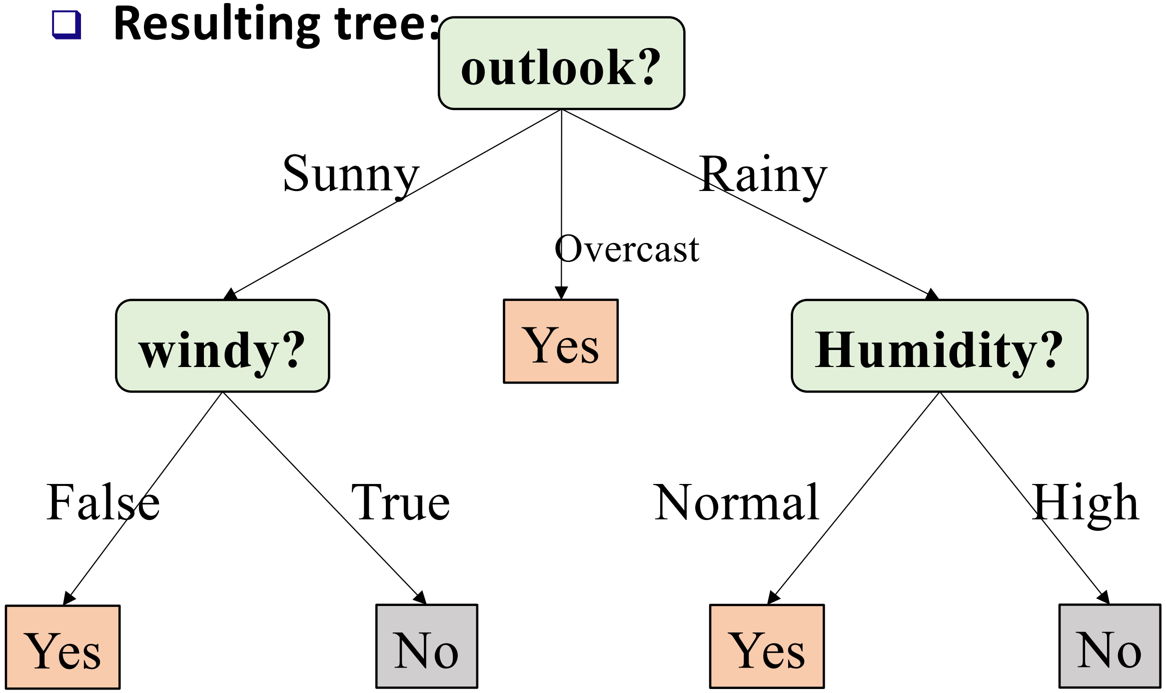
outlook	temp	humidity	windy	hours
rainy	hot	high	FALSE	25.0
rainy	hot	high	TRUE	30.0
overcast	hot	high	FALSE	46.0
sunny	mild	high	FALSE	45.0
sunny	cool	normal	FALSE	52.0
sunny	cool	normal	TRUE	23.0
overcast	cool	normal	TRUE	43.0
rainy	mild	high	FALSE	35.0
rainy	cool	normal	FALSE	38.0
sunny	mild	normal	FALSE	46.0
rainy	mild	normal	TRUE	48.0
overcast	mild	high	TRUE	52.0
overcast	hot	normal	FALSE	44.0
sunny	mild	high	TRUE	30.0

Outline

- Decision tree induction
  - Recall of the feature encoding
  - Computation of info gain
- Ensemble methods: random forest and boosting

Decision Tree Induction: An Example

- **Decision tree construction**:
  - A top-down, recursive, divide-and-conquer process
- **Resulting tree:**



Training data set: Play Golf?

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

https://www.saedsayad.com/decision_tree.htm

Decision Tree Induction: Algorithm

- Basic algorithm
  - Tree is constructed in a **top-down, recursive, divide-and-conquer manner**
  - At start, all the training examples are at the root
  - Examples are partitioned recursively based on selected attributes
  - On each node, attributes are selected based on the training examples on that node, and a heuristic or statistical measure (e.g., **information gain, Gini index**)

Decision Tree Induction: Algorithm

- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning
  - There are no samples left
- Prediction
  - **Majority voting** is employed for classifying the leaf

How to Handle Continuous-Valued Attributes?

- Method 1: Discretize continuous values and treat them as categorical values
  - E.g., age: < 20, 20..30, 30..40, 40..50, > 50
- Method 2: Determine the ***best split point*** for continuous-valued attribute A
  - Sort:, e.g. 15, 18, 21, 22, 24, 25, 29, 31, …
  - *Possible split point:* $(a_i+a_{i+1})/2$
    - e.g., (15+18)/2 = 16.5, 19.5, 21.5, 23, 24.5, 27, 30, …
  - The point with the *maximum information gain* for A is selected as the **split-point** for A
- Split: Based on split point P
  - The set of tuples in D satisfying A ≤ P vs. those with A > P

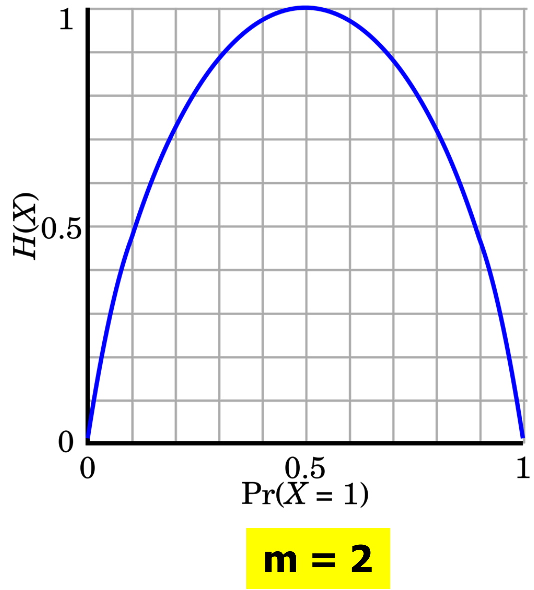
Splitting Measures: Information Gain

- Entropy (Information Theory)
  - A measure of uncertainty associated with a random number
  - Calculation: For a discrete random variable Y taking m distinct values $\{y_1, y_2, \ldots, y_m\}$

$$H(Y) = -\sum_{i=1}^{m} p_i \log(p_i) \quad where\ p_i = P(Y = y_i)$$

  - Interpretation
    - Higher entropy → higher uncertainty
    - Lower entropy → lower uncertainty
- Conditional entropy

$$H(Y|X) = \sum_{x} p(x) H(Y|X = x)$$



Information Gain: An Attribute Selection Measure

- Select the attribute with the highest information gain (used in typical decision tree induction algorithm: ID3/C4.5)
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Information needed (after using A to split D into v partitions) to classify D:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

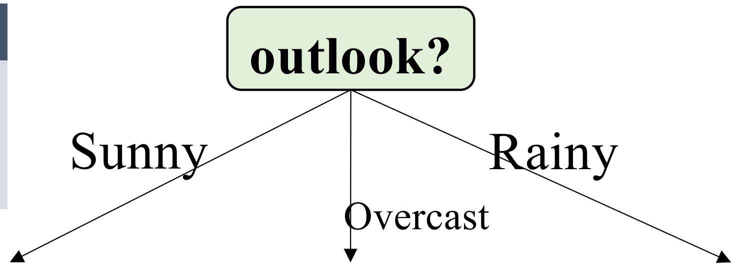
- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

Example: Attribute Selection with Information Gain

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

outlook	yes	no	I(yes, no)
rainy	2	3	0.971
overcast	4	0	0
sunny	3	2	0.971



$$Info(D) = I(9,5) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940$$

$$Info(D) = -\sum_{i=1}^{m} p$$

$$Info_{outlook}(D) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.694$$

$\frac{5}{14} I(2,3)$ means “outlook=rainy” has 5 out of 14 samples, with 2 yes’es and 3 no’s. Hence

$$Gain(outlook) = Info(D) - Info_{outlook}(D) = 0.246$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

Example: Attribute Selection with Information Gain

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Temp	Yes	No	I(Yes, No)
Hot	2	2	?
Mild	4	2	?
Cool	3	1	?

Windy	Yes	No	I(Yes, No)
True	?	?	?
False	?	?	?

Humidity	Yes	No	I(Yes, No)
Normal	6	1	?
High	3	4	?

Similarly, we can get

$$Gain(Temp) = 0.029,$$
$$Gain(humidity) = 0.151,$$
$$Gain(Windy) = 0.048$$

Gain(outlook) = 0.246

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

outlook?

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Rainy	Mild	Normal	True	Yes

windy?

Outlook	Temp	Humidity	Windy	Play Golf
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Sunny	Mild	Normal	False	Yes
Sunny	Mild	High	True	No

humidity?

Branche overcast

Gain Ratio: A Refined Measure for Attribute Selection

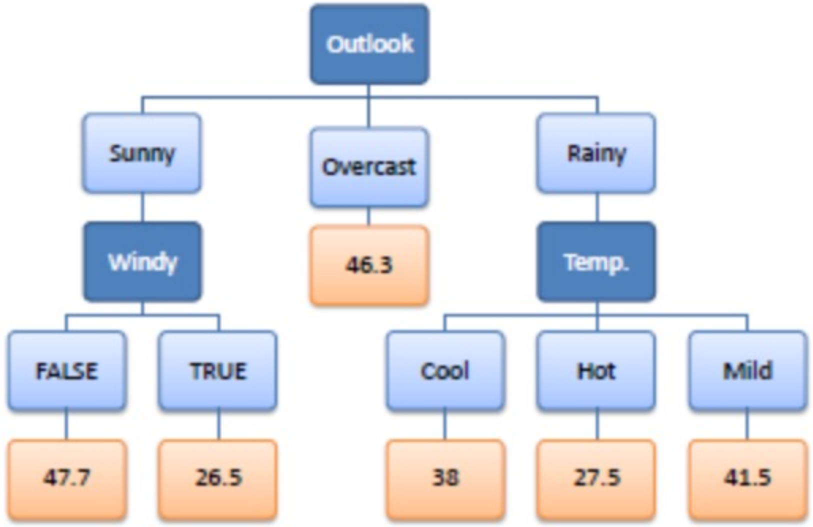
- Information gain measure is biased towards attributes with a large number of values (e.g. ID)
- Gain ratio: Overcomes the problem (as a normalization to information gain)

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

  - GainRatio(A) = Gain(A)/SplitInfo(A)
  - The attribute with the maximum gain ratio is selected as the splitting attribute
  - Gain ratio is used in a popular algorithm C4.5 (a successor of ID3) by R. Quinlan
- Example
  - $\mathrm{SplitInfo_{temp}(D)} = -\frac{4}{14}\log_2\frac{4}{14} - \frac{6}{14}\log_2\frac{6}{14} - \frac{4}{14}\log_2\frac{4}{14} = 1.557$
  - GainRatio(temp) = 0.029/1.557 = 0.019

Regression Trees

Outlook	Temp	Humidity	Windy	Hour s
Rainy	Hot	High	False	25
Rainy	Hot	High	True	30
Overcast	Hot	High	False	46
Sunny	Mild	High	False	45
Sunny	Cool	Normal	False	52
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	35
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30



Tree induction

Hours
25
30
46
45
52
23
43
35
38
48
48
52
44
30

Replace Information Gain with Standard Deviation (écart-type) Reduction

partition the data into subsets that contain instances with similar values (homogenous)
If the numerical sample is completely homogeneous its standard deviation is zero

Standard deviation for one attribute:

$$\boldsymbol{Count} = n = 14$$

$$\boldsymbol{Average} = \bar{x} = \frac{\sum x}{n} = 39.8$$

$$\boldsymbol{Standard\ Deviation} = S = \sqrt{\frac{\sum(x-\bar{x})^2}{n}} = 9.32$$

for tree building (branching)

$$\boldsymbol{Coeffeicient\ of\ Variation} = CV = \frac{S}{\bar{x}} * 100\% = 23\%$$

used to decide when to stop branching

Standard deviation for two attributes (target and predictor):

$$S(T,X) = \sum_{c \in X} P(c)S(c)$$

		Hours Played (StDev)	Count
Outlook	Overcast	3.49	4
	Rainy	7.78	5
	Sunny	10.87	5
			14

S(Hours, Outlook) = **P**(Sunny)***S**(Sunny) + **P**(Overcast)***S**(Overcast) + **P**(Rainy)***S**(Rainy)

= (4/14)*3.49 + (5/14)*7.78 + (5/14)*10.87

= 7.66

Standard Deviation Reduction

the decrease in standard deviation after a dataset is split on an attribute

Build the tree =

find attribute that returns the highest standard deviation reduction (i.e., the most homogeneous branches)

Step 1: The standard deviation of the target is calculated SD (Hours Played) = 9.32

Step 2: The dataset is then split on the different attributes.
The standard deviation for each branch is calculated.
The resulting standard deviation is subtracted from the standard deviation before the split.
The result is the standard deviation reduction.

		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		

		Hours Played (StDev)
Temp.	Cool	10.51
	Hot	8.95
	Mild	7.65
SDR= 0.48		

		Hours Played (StDev)
Humidity	High	9.36
	Normal	8.37
SDR=0.28		

		Hours Played (StDev)
Windy	False	7.87
	True	10.59
SDR=0.29		

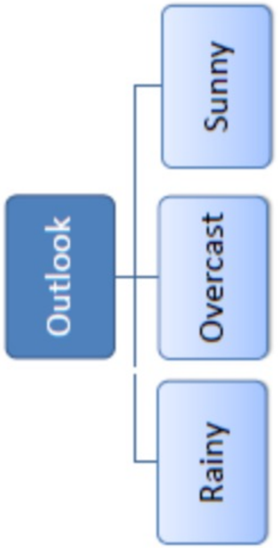
Step 3: The attribute with the largest standard deviation reduction is chosen for the decision node.

$$SDR(T,X) = S(T) - S(T,X)$$

SDR(Hours , Outlook) = **S**(Hours) – **S**(Hours, Outlook)

= 9.32 – 7.66 = 1.66

Step 4a: The dataset is divided based on the values of the selected attribute.
This process is run recursively on the non-leaf branches, until all data is processed.



Outlook	Temp	Humidity	Windy	Hours Played
Sunny	Mild	High	FALSE	45
Sunny	Cool	Normal	FALSE	52
Sunny	Cool	Normal	TRUE	23
Sunny	Mild	Normal	FALSE	46
Sunny	Mild	High	TRUE	30
Overcast	Hot	High	FALSE	46
Overcast	Cool	Normal	TRUE	43
Overcast	Mild	High	TRUE	52
Overcast	Hot	Normal	FALSE	44
Rainy	Hot	High	FALSE	25
Rainy	Hot	High	TRUE	30
Rainy	Mild	High	FALSE	35
Rainy	Cool	Normal	FALSE	38
Rainy	Mild	Normal	TRUE	48

termination criteria: when coefficient of deviation (CV) for a branch becomes smaller than a certain threshold (e.g., 10%) and/or when too few instances (n) remain in the branch (e.g., 3).

Pro’s and Con’s

- Pro’s
    - Easy to explain (even for non-expert)
    - Easy to implement (many software)
    - Efficient
    - Can tolerant missing data
    - White box
    - No need to normalize data
    - Non-parametric: No assumption on data distribution, no assumption on attribute independency
    - Can work on various attribute types

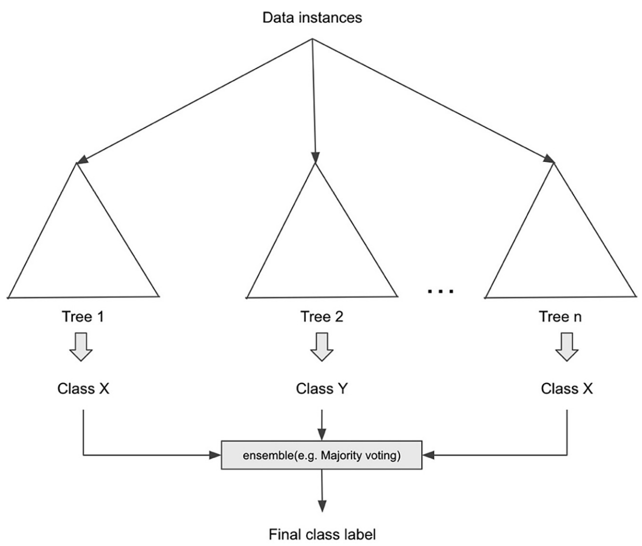
Con’s

- Con’s
    - Unstable. Sensitive to noise
    - Accuracy may be not good enough (depending on your data)
    - The optimal splitting is NP. Greedy algorithms are used
    - Overfitting

Random forests

combine the decisions of multiple trees can lead to improved overall performance

create multiple smaller subtrees, each subtree uses a random subset of all the features
the final decision is made by either majority voting (for classification) or averaging (for regression)

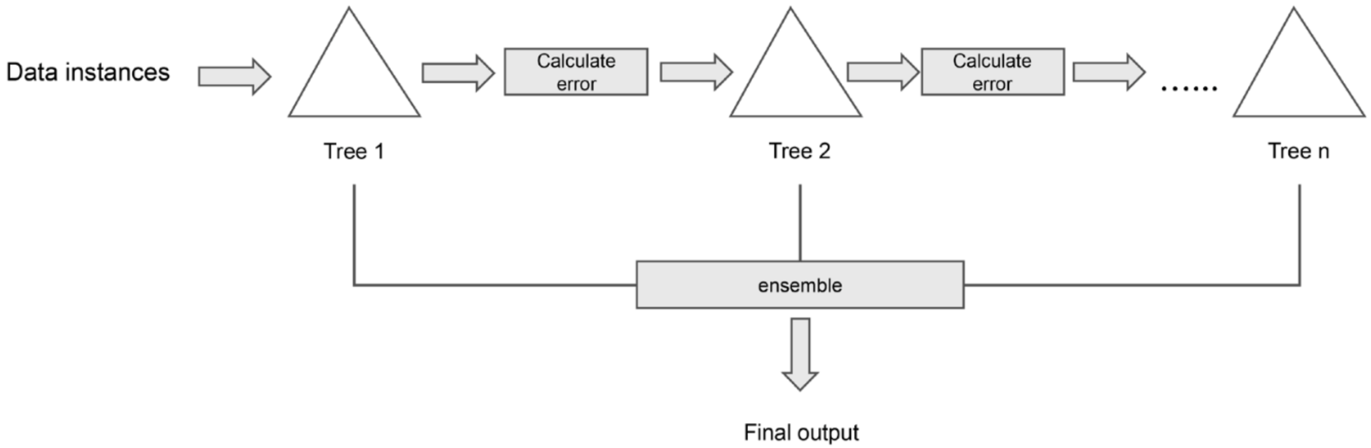


Pro’s
improved accuracy by combining predictions of multiple trees
reduce overfitting by introducing randomness
support parallel processing f

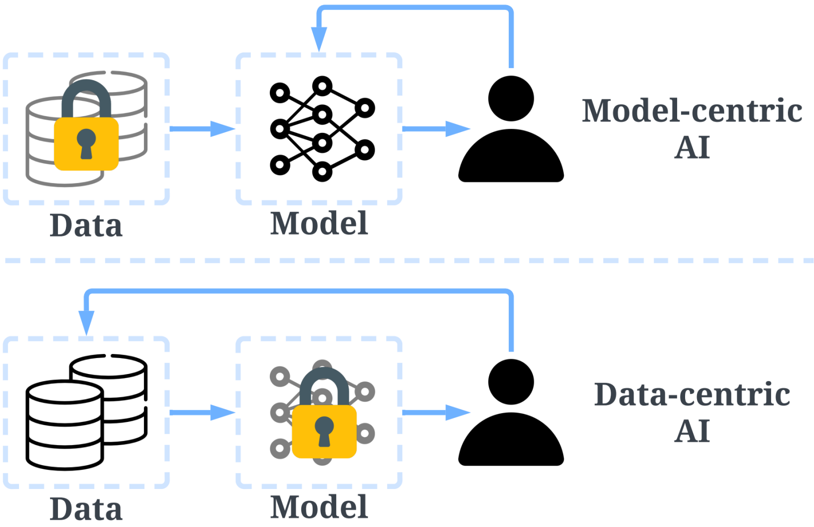
Con’s
reduced interpretability compared to decision trees
longer training and prediction times
need for hyperparameter tuning

Gradient boosting

sequentially aggregates results from different trees
each learner tree corrects the errors of the previous tree
offers more hyperparameters to fine-tune allows for custom loss functions
handles imbalanced datasets but no parallelization → slower in training
less interpretable
Many implementations: XGBoost, LightGBM



Data-centric AI vs. Model-centric AI



focus on model
data unchanged

focus on data
systematic engineering
of data to build AI

Zha, Daochen, et al. "Data-centric Artificial Intelligence: A Survey."

https://github.com/daochenzha/data-centric-AI

Which solutions

- Data profiling, quality assessment
  - Deequ (Amazon)[2], Delta Lake (Databricks), …
- Continuous data validation
  - Delta Lake, TFX validation [1]
- Manual approach
  - Error prone, costly, not always scalable
- Semantic type detection [3]
  - Use ML to classify column types
  - Extract richer information about types (date, location, name, … vs basic types like text/numbers/…)

[1] Breck et al. Data validation for machine learning. In Proceedings of SysML, 2019.
[2]Schelter et al. Automating large-scale data quality verification. VLDB 2018
[3] Shah et al. Towards benchmarking feature type inference for automl platforms. SIGMOD 2021

Data quality: overview

- Context
  - Data integrated from different sources
  - often produced w/o schema nor quality guarantees
- Improving data quality impacts all applications
  - Business intelligence: avoid wrong decision
  - Machine learning: improve model performance
  - Pipelines: prevent glitches (null values, type mis match…)
- Manually checking DQ may be complex
  - data volume and dynamicity
  - user-defined code: cumbersome and error prone
- Automation for quality check

Data quality dimensions

- Standard classification, extensive literature
- Dimensions
  - Completeness
    - degree to which an entity reflects the real-world
    - general: column with missing (null) values
    - contextualized: absence of value means not-applicable
  - Consistency
    - adherence to semantic rules: categorical column, referential constraints (foreign key)
  - Accuracy
    - syntactic: column value adheres to its type
    - semantic: domain-specific
- Several associated metrics

The Deequ approach

- Declarative specification and verification of DQ metrics
- Draws inspiration from software engineering: unit tests, cont. testing
- Features
  - A rich set of useful metrics
  - Suggested based on confidence score or verified upon user request
  - Use: enforce constraints upon ingestion, non valid data is quarantined
  - Low overhead: automatic optimization of the metric computation
  - Incremental maintenance: cope with data dynamicity
  - Deployed on Apache Spark and operational on AWS suite
  - Open source implementation (py)Deequ

Schelter et al. Automating large-scale data quality verification. VLDB 2018

Metrics computation

Completeness $|\{d \in D \mid d(col) \neq \mathtt{null}\}| \,/\, N$

Compliance $|\{d \in D \mid p(d)\}| \,/\, N$ *ratio of records sat. predicate p*

Uniqueness $|\{v \in V \mid c_v = 1\}| \,/\, |V|$ *ratio of unique values, c_v card of v*

Distinctness $|V| \,/\, N$ Entropy $-\sum_v \frac{c_v}{N} \log \frac{c_v}{N}$

Mutual Information $\sum_{v_1} \sum_{v_2} \frac{c_{v_1 v_2}}{N} \log \frac{c_{v_1 v_2}}{c_{v_1} c_{v_2}}$ v1 and v2 refer to different columns

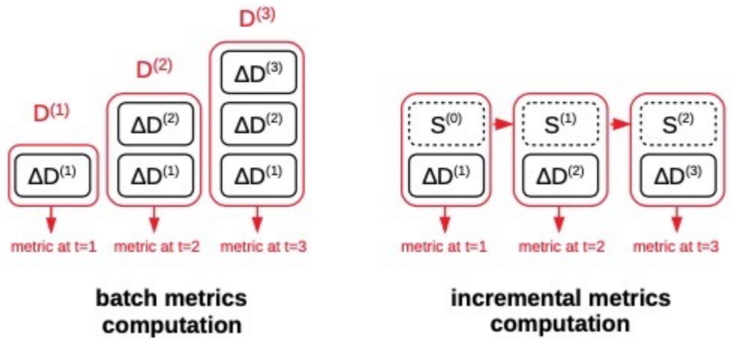
ApproxQuantile and ApproxCountDistinct: SOTA algorithms (refer to the paper)

Predictability: predict column from other columns - max. a posteriori decision rule

Incremental computation

Goal: avoid batch computation which may be costly!

Assumption: append-only updates



notation newly added records ΔD. ΔV denote all unique values

general formulae $S^{(t)} = f(\Delta D^{(t)}, S^{(t-1)})$

Compliance $$\frac{|\{d \in D \mid p(d)\}| + |\{d \in \Delta D \mid p(d)\}|}{N + \Delta N}$$

Uniqueness $$\frac{|\{v \in V \cup \Delta V \mid c_v + \Delta c_v = 1\}|}{|V \cup \Delta V|}$$

other formulas in the paper

Architecture and typical scenario

User

- runs analyses,
- requests profiles or suggestions
- enforces constraints