Spark Machine Learning

Some theory ☺

DataFrames vs RDD

dept	age	name
Bio	48	H Smith
CS	54	A Turing
Bio	43	B Jones
Chem	61	M Kennedy

Data grouped into named columns

RDD API

```
pdata.map(lambda x: (x.dept, [x.age, 1])) \
  .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
  .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
  .collect()
```

DataFrame API

```
data.groupBy("dept").avg("age")
```

They can be compared for a task, but there is not a general answer!
They have different purposes!

What is ML for you?

What is Machine Learning?

“Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.”

Paraphrased from Samuel Arthur, "Some Studies in Machine Learning Using the Game of Checkers"

What is Machine Learning?

A computer program is said to learn from experience 'E',
with respect to some task 'T' and
performance measure ‘P’,
if its performance on 'T', as measured by 'P', improves with experience 'E'

Tom Mitchell (1998)

How Human Learn?

- We repeat the same task (T) over and over again to gain experience (E)
- This doing over and over again the same task (T) is known as practice (P)
- With practice (P) and experience (E), we get better at that task (T)
- Once we achieve some level, we have learnt
- Everything in live is learnt

Learning example - Music Instrument

- We practice every day the instrument with the same song, again and again
- We will be pretty bad to start with, but with practice we improve
- Our teacher will measure our performance to understand if we have done some progress
- Once the teacher approves, you are a player

Learning is hard!

- Practicing over and over again the same task is often boring and frustrating
- Without the proper measurement, we may be stuck at the same level without making much progress
- Making progress initially is pretty easy, but it starts to get harder and harder
- Sometimes natural talent also dictates cap on how good we can be

How Machine Learn?

- You start with an assumption about solution for given problem
- Make machine go over data and verify the assumption
- Run the program over and over again on the same data updating the assumption on the way
- Measure improvement after each round, adjust accordingly assumption
- Once you have no more assumption changing, stop running

ML and Big Data

- Ability to learn on large corpus of data is a real boon for ML
- Even simplistic ML models shine when they are trained on huge amount of data (but be careful with overfitting)
- With big data toolsets, a wide variety of ML application have started to emerge other than academic specifics one
- Big data democratizing ML for general public

ML vs Big Data

Machine Learning	Big Data
Optimized for iterative computations	Optimized of single pass computations
Maintains state between stages	Stateless
CPU intensive	Data intensive
Vector/Matrix based or multiple rows/cols at a time	Single row/column at time

ML in Hadoop

- In Hadoop each iteration of ML translates into a single Map/Reduce job
- Each of these jobs need to store data in HDFS, which leads to an important overhead
- Keeping state across jobs is not directly available in M/R
- Constant fight between quality of results vs performance

ML in Spark

- Spark is first general purpose big data processing engine built for ML from day one
- The initial design in Spark was driven by ML optimization
  - Caching - for running on data multiple times
  - Accumulator - to keep state across multiple iterations in memory
  - Good support for CPU intensive tasks with laziness
- One of the examples in Spark first version was of ML

How Spark handles ML?

The Goal is to make ML:

- Practical
- Scalable
- Easy

What Spark provides

- ML Algorithm
- Featurization
  [feature extraction, transformation, dimensionality reduction]
- Pipelines
  [tools for constructing, evaluating, tuning pipelines]
- Persistence
  [saving and load algorithms, models, pipelines]
- Utilities
  [linear algebra, statistics, data handling, …]

ML Algorithms

- **Supervised learning**

  Training data contains both input vector and desired output. We also call it as labeled data.

  Ex: Linear Regression, Logistic Regression

- **Semi-supervised learning**

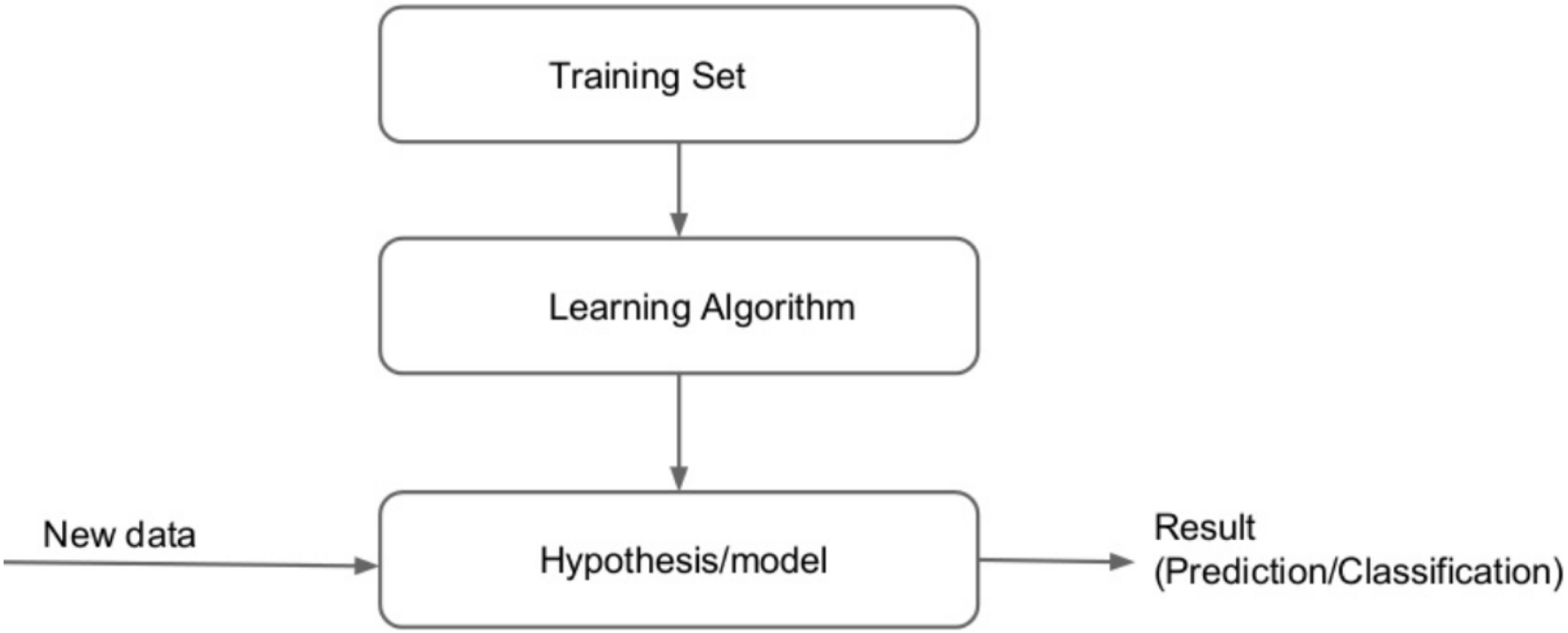
  Training data contains both labeled and unlabeled data.

- **Unsupervised learning**

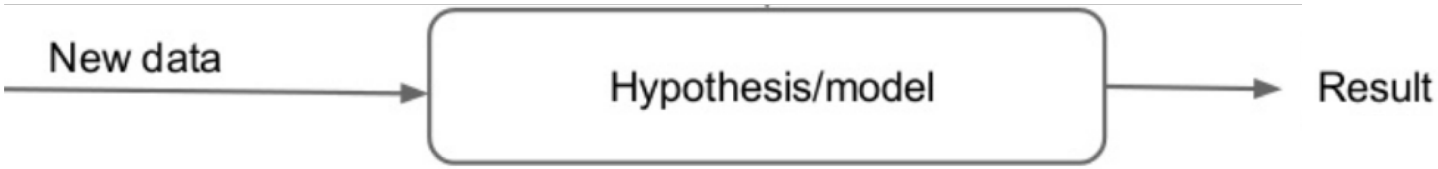
  Training data sets without labels.

  Ex: K-means clustering

Process of Supervised Learning



Process of Unsupervised Learning



ML Terminology

- **Training Set**

  Set of data used to teach the machine. In supervised learning both input vector and output will be available

- **Learning algorithm**

  Algorithm that consumes the training set to infer relation between input vectors that optimizes for known output labels

- **Model**

  Learnt function of input parameters

Algorithm Coverage

- Classification
  - Logistic regression
  - Naive Bayes
  - Streaming logistic regression
  - Linear SVMs
  - Decision trees
  - Random forests
  - Gradient-boosted trees
  - Multilayer perceptron
- Regression
  - Ordinary least squares
  - Ridge regression
  - Lasso
  - Isotonic regression
  - Decision trees
  - Random forests
  - Gradient-boosted trees
  - Streaming linear methods
  - **Generalized Linear Models**
- Frequent itemsets
  - FP-growth
  - PrefixSpan

Recommendation
- Alternating Least Squares

Feature extraction & selection
- Word2Vec
- Chi-Squared selection
- Hashing term frequency
- Inverse document frequency
- Normalizer
- Standard scaler
- Tokenizer
- One-Hot Encoder
- StringIndexer
- VectorIndexer
- VectorAssembler
- Binarizer
- Bucketizer
- ElementwiseProduct
- PolynomialExpansion
- **Quantile discretizer**
- **SQL transformer**

Model import/export

Pipelines

Clustering
- Gaussian mixture models
- K-Means
- Streaming K-Means
- Latent Dirichlet Allocation
- Power Iteration Clustering
- **Bisecting K-Means**

Statistics
- Pearson correlation
- Spearman correlation
- Online summarization
- Chi-squared test
- Kernel density estimation
- Kolmogorov–Smirnov test
- **Online hypothesis testing**
- **Survival analysis**

Linear algebra
- Local dense & sparse vectors & matrices
- **Normal equation for least squares**
- Distributed matrices
  - Block-partitioned matrix
  - Row matrix
  - Indexed row matrix
  - Coordinate matrix
- Matrix decompositions

databricks

List based on Spark 2.0

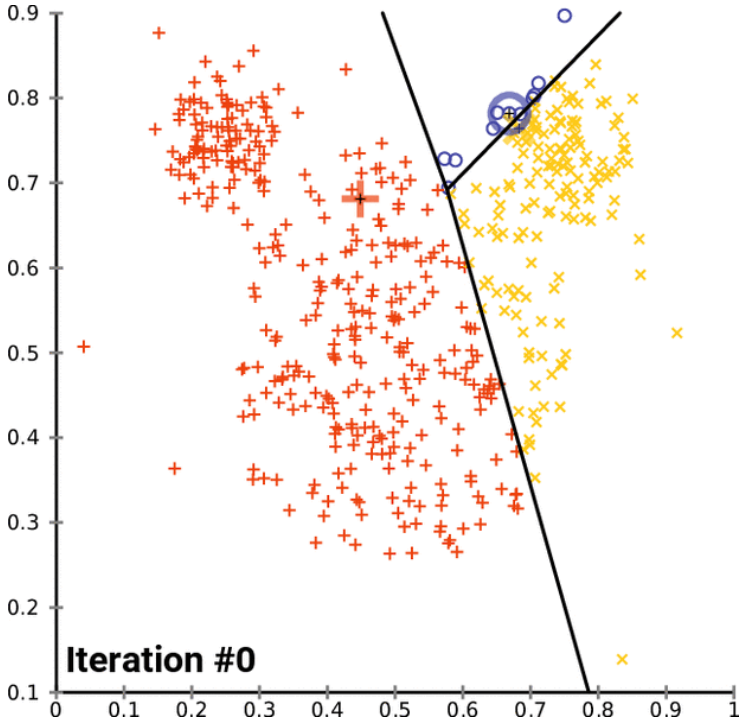
dbTrento

Clustering

- **K-Means**

Partitions *n* elements into *k* clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster

http://educlust.dbvis.de/

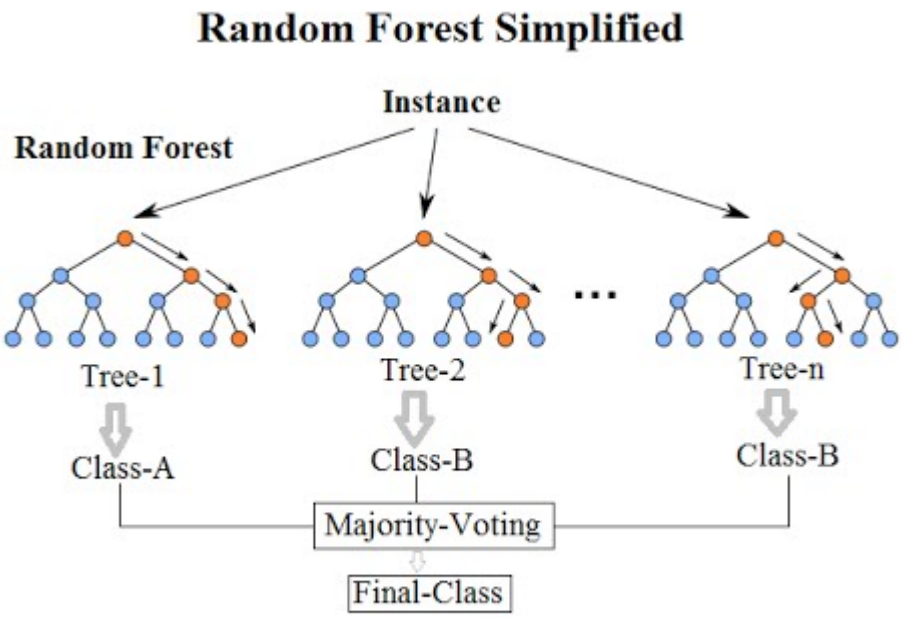


db Trento

Classification

- **Random Forest Classifier**

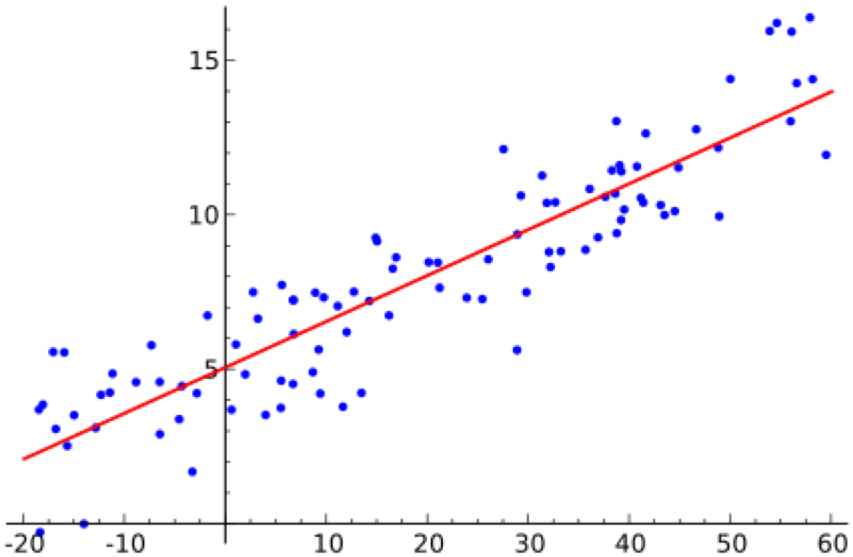
Operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes of the individual trees

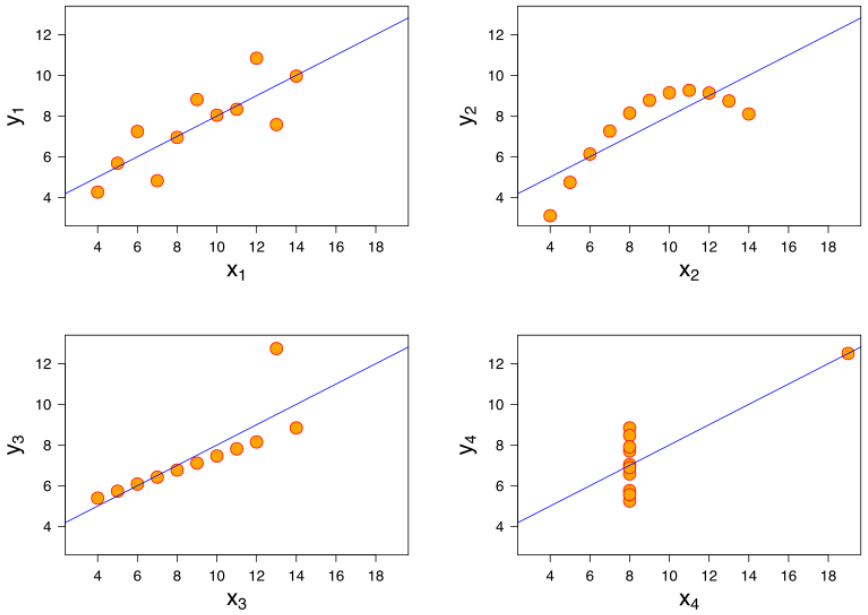


Regression

- **Linear Regression**

Linear approach for modeling the relationship between a scalar dependent variable *y* and one or more explanatory variables denoted *X*





dbTrento

Pattern Mining

Finds statistically relevant patterns between data examples

Consider *minSup*= 0.5 and *minConf*= 0.5:

ID	Sequences
seq1	$\{a, b\},\{c\},\{f\},\{g\},\{e\}$
seq2	$\{a, d\},\{c\},\{b\},\{a, b, e, f\}$
seq3	$\{a\},\{b\},\{f\},\{e\}$
seq4	$\{b\},\{f, g\}$

A sequence database

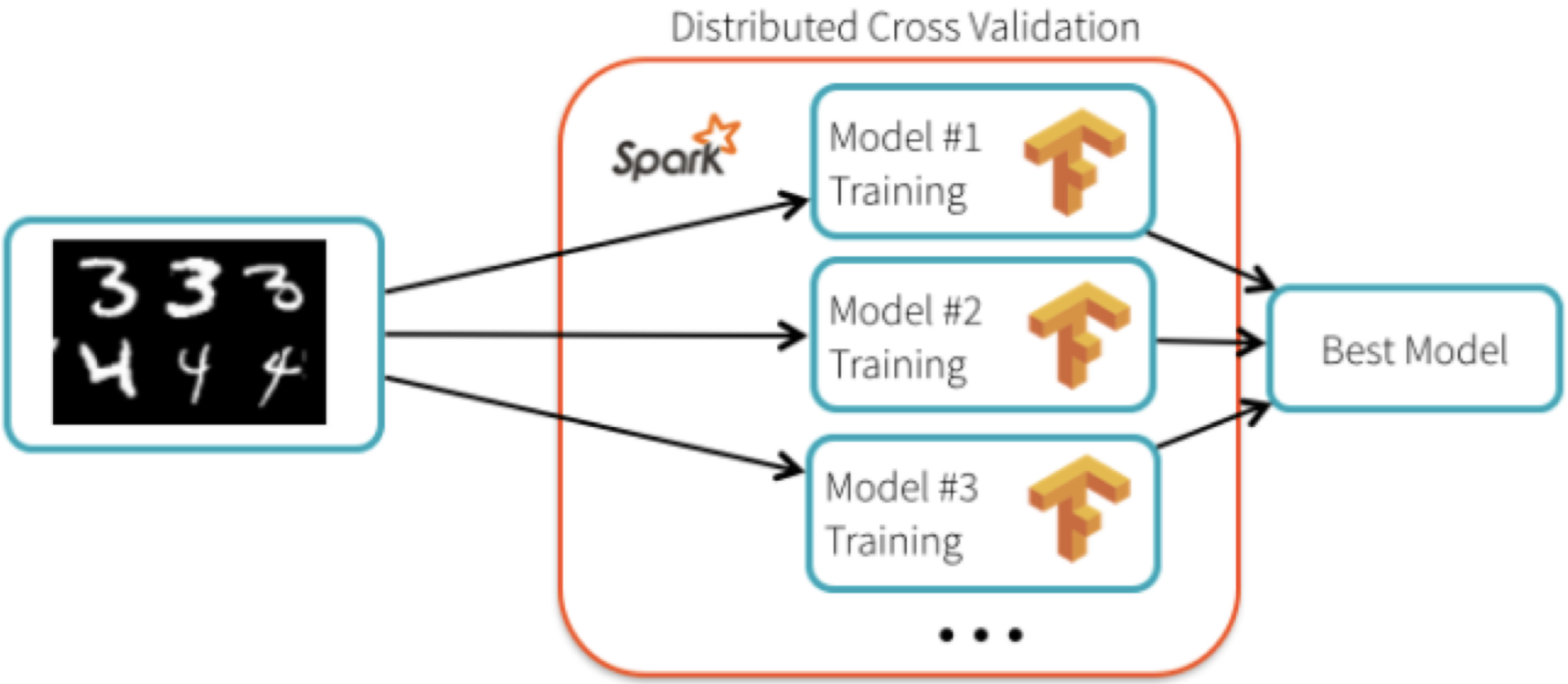
→

ID	Rule	Support	Confidence
r1	$\{a, b, c\} \Rightarrow \{e\}$	0.5	1.0
r2	$\{a\} \rightarrow \{c, e, f\}$	0.5	0.66
r3	$\{a, b\} \rightarrow \{e, f\}$	0.5	1.0
r4	$\{b\} \rightarrow \{e, f\}$	0.75	0.75
r5	$\{a\} \rightarrow \{e, f\}$	0.75	1.0
r6	$\{c\} \rightarrow \{f\}$	0.5	1.0
r7	$\{a\} \rightarrow \{b\}$	0.5	0.66
…	…	…	…

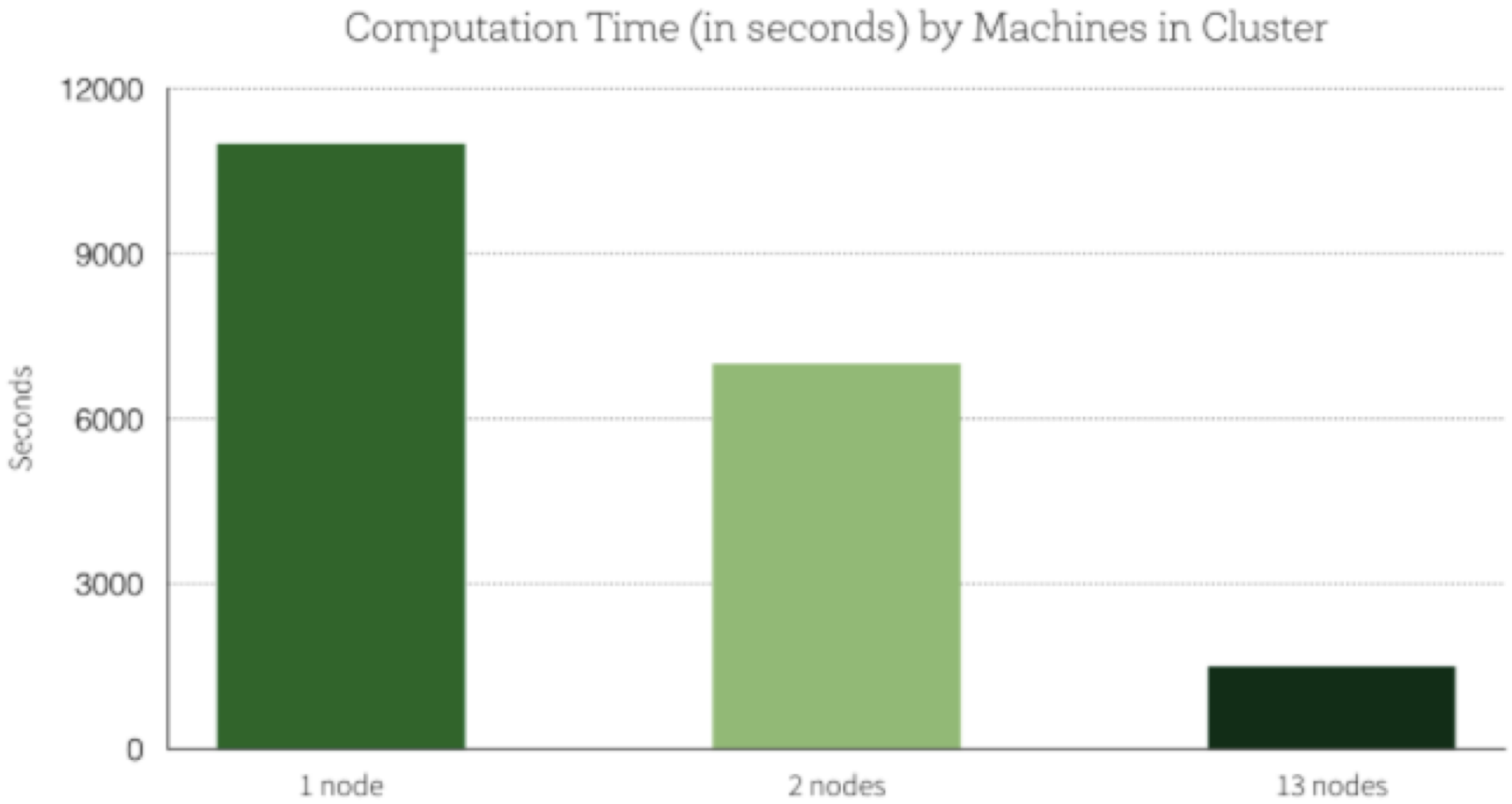
Some rules found

Deep Learning

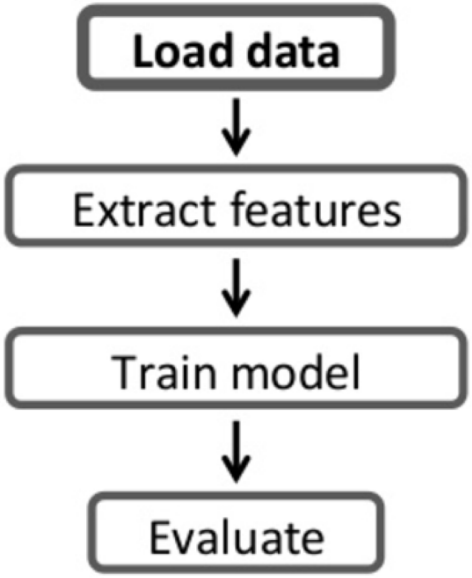
Deep Learning



Deep Learning



Data Loading



Data sources for DataFrames

built-in

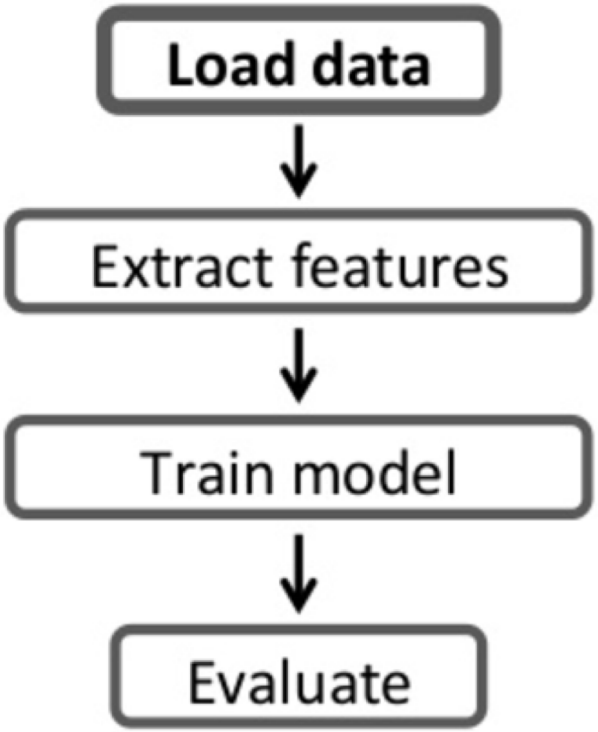
Parquet
JDBC
{ JSON }
PostgreSQL
HIVE
MySQL
HDFS
amazon S3
H2

external

AVRO
CSV
dBase
APACHE HBASE
elasticsearch.
cassandra
amazon Redshift
and more ...

dbTrento

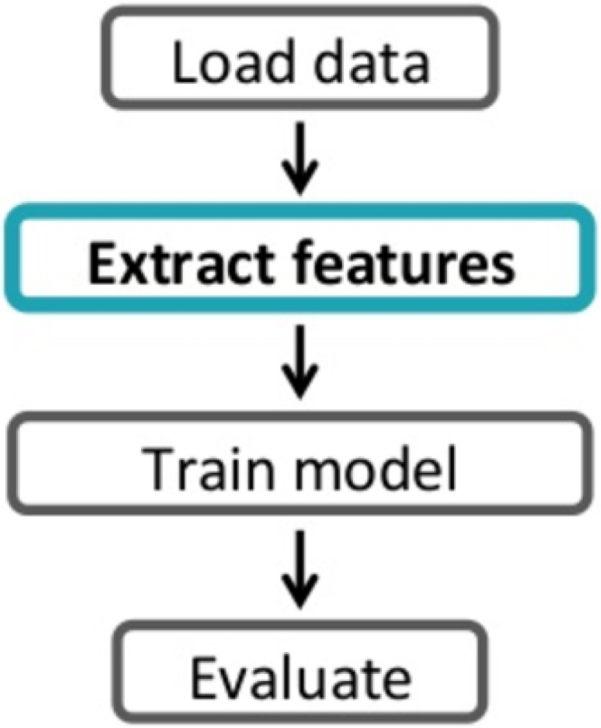
Data Loading



Data Schema:

- **label:** **Int**
- **text:** **String**

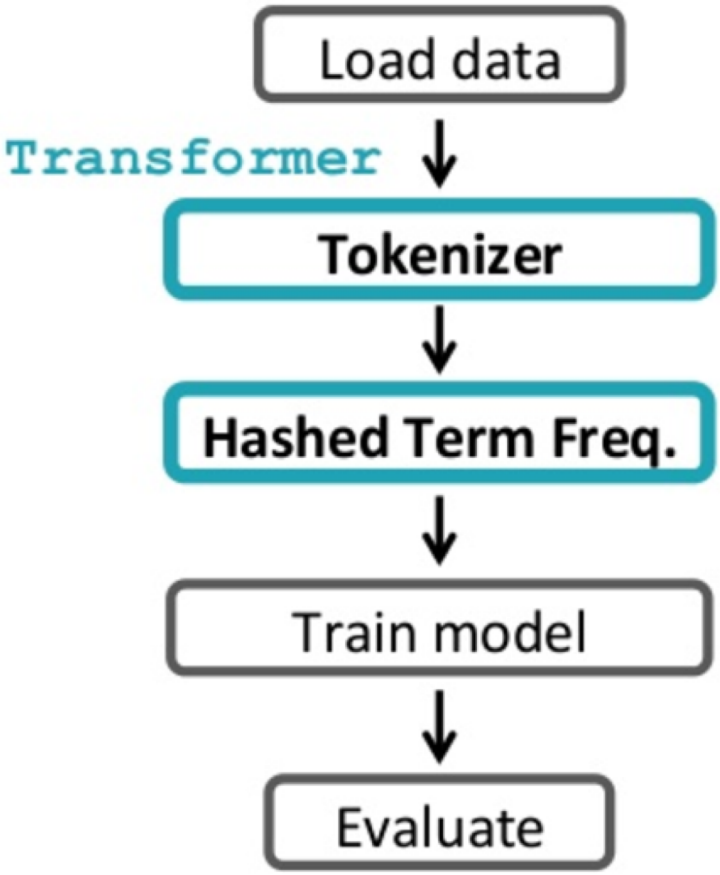
Feature Extraction



Data Schema:

- **label:** **Int**
- **text:** **String**

Feature Extraction

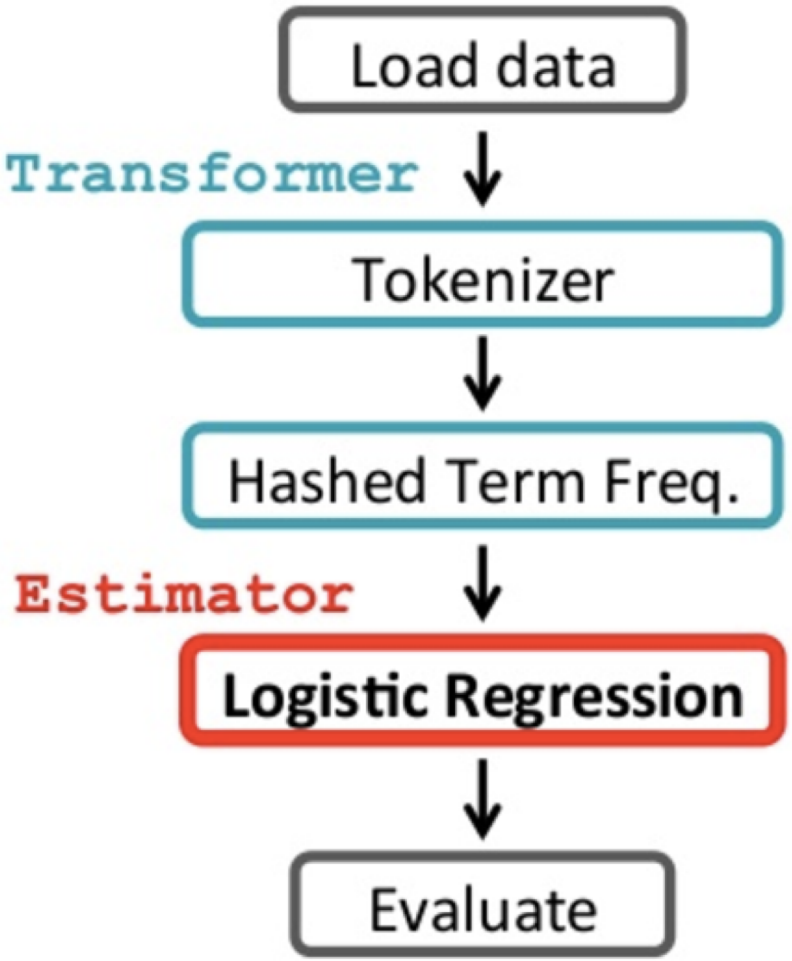


Data Schema:

- **label:** **Int**
- **text:** **String**

- **words:** **Seq[String]**
- **features:** **Vector**

Model Training



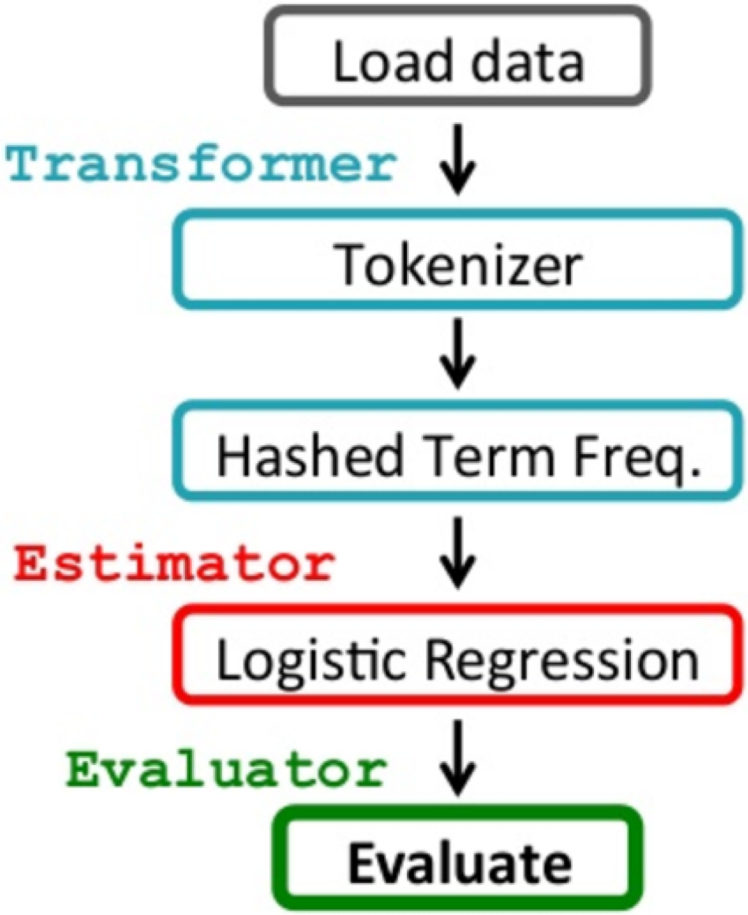
Data Schema:

- **label:** **Int**
- **text:** **String**

- **words:** **Seq[String]**
- **features:** **Vector**

- **prediction:** **Int**

Model Evaluation



Data Schema:

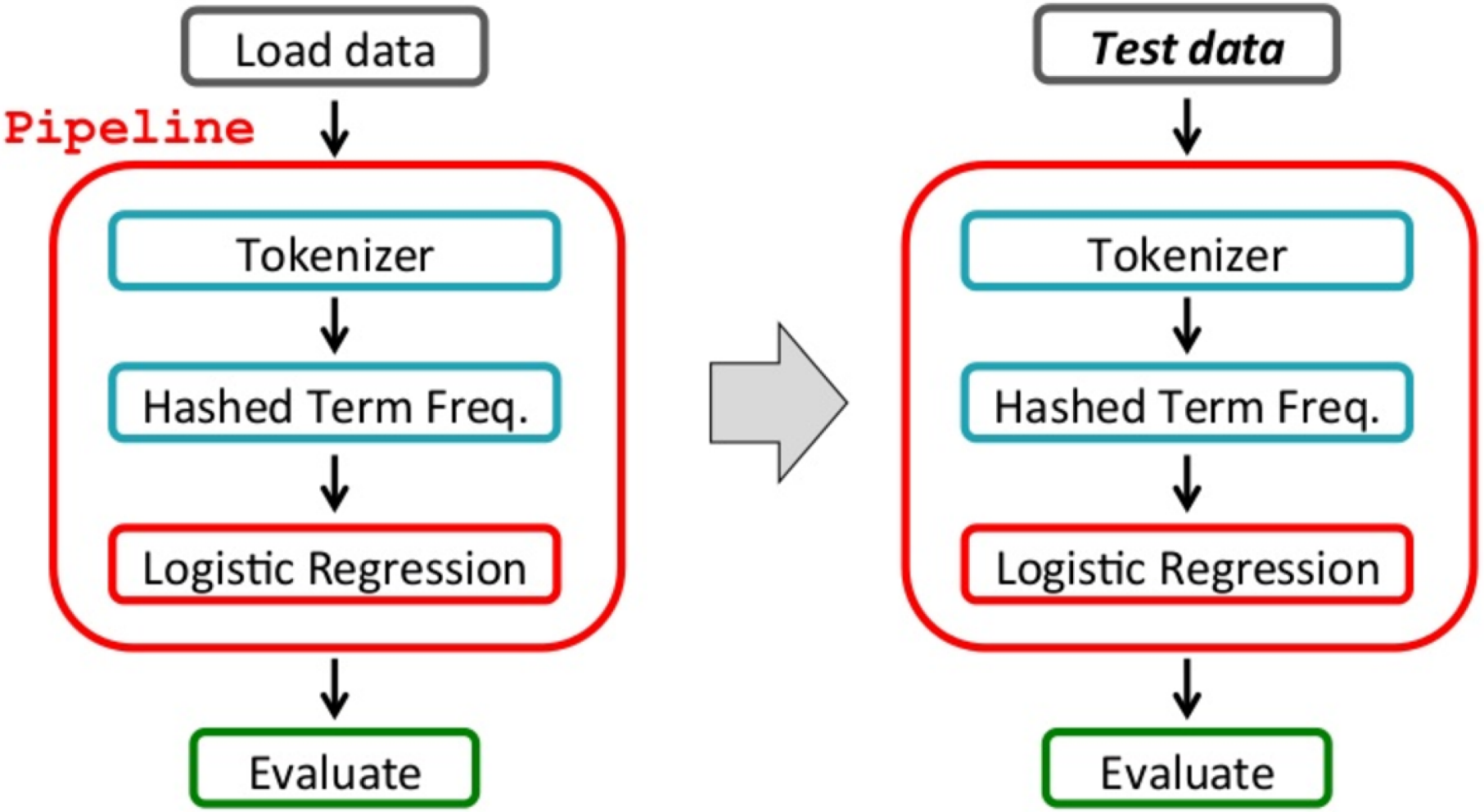
- **label:** **Int**
- **text:** **String**

- **words:** **Seq[String]**
- **features:** **Vector**

- **prediction:** **Int**

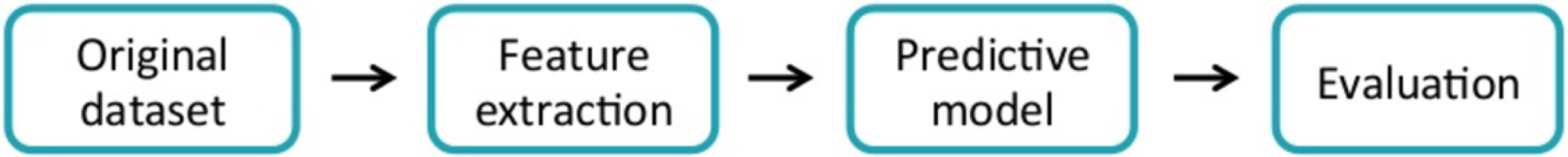
By default each step adds a column

ML Pipeline



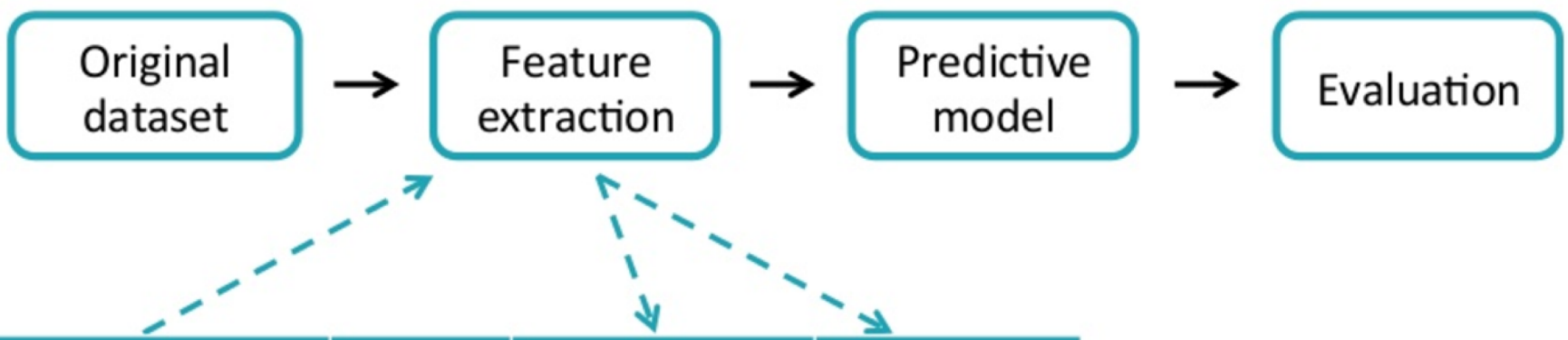
We use exactly the same pipeline
over different data (train and test set)

Data Loading



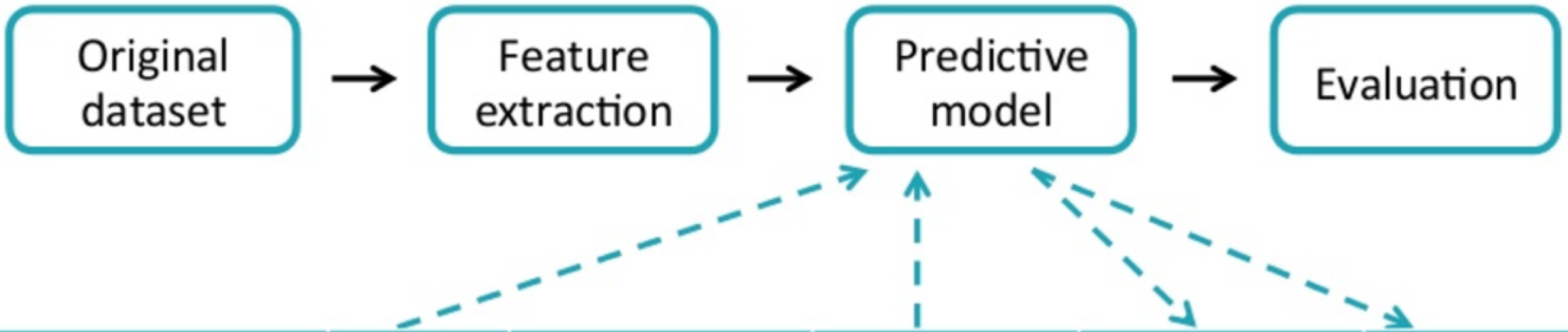
Text	Label
I bought the game...	4
Do NOT bother try...	1
this shirt is aweso...	5
never got it. Seller...	1
I ordered this to...	3

Feature Extraction



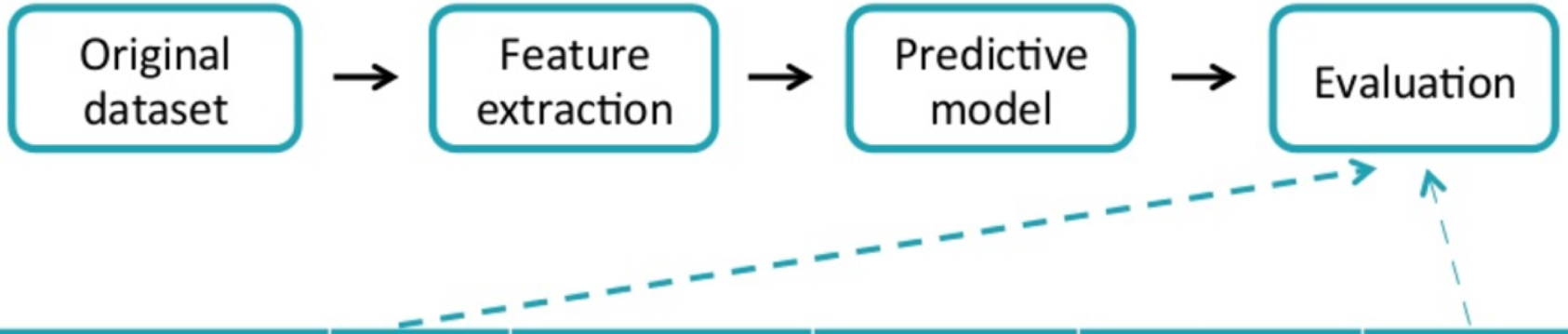
Text	Label	Words	Features
I bought the game...	4	“i", “bought”,...	[1, 0, 3, 9, ...]
Do NOT bother try...	1	“do”, “not”,...	[0, 0, 11, 0, ...]
this shirt is aweso...	5	“this”, “shirt”	[0, 2, 3, 1, ...]
never got it. Seller...	1	“never”, “got”	[1, 2, 0, 0, ...]
I ordered this to...	3	“i”, “ordered”	[1, 0, 0, 3, ...]

Model Fitting



Text	Label	Words	Features	Prediction	Probability
I bought the game...	4	“i", “bought”,...	[1, 0, 3, 9, ...]	4	0.8
Do NOT bother try...	1	“do”, “not”,...	[0, 0, 11, 0, ...]	2	0.6
this shirt is aweso...	5	“this”, “shirt”	[0, 2, 3, 1, ...]	5	0.9
never got it. Seller...	1	“never”, “got”	[1, 2, 0, 0, ...]	1	0.7
I ordered this to...	3	“i”, “ordered”	[1, 0, 0, 3, ...]	4	0.7

Model Evaluation



Text	Label	Words	Features	Prediction	Probability
I bought the game...	4	“i", “bought”,...	[1, 0, 3, 9, ...]	4	0.8
Do NOT bother try...	1	“do”, “not”,...	[0, 0, 11, 0, ...]	2	0.6
this shirt is aweso...	5	“this”, “shirt”	[0, 2, 3, 1, ...]	5	0.9
never got it. Seller...	1	“never”, “got”	[1, 2, 0, 0, ...]	1	0.7
I ordered this to...	3	“i”, “ordered”	[1, 0, 0, 3, ...]	4	0.7

Feature Extraction

Features

- **Extraction**

  Extracting features from “raw” data

- **Transformation**

  Scaling, converting, or modifying features

- **Selection**

  Selecting a subset from a larger set of features

- **Locality Sensitive Hashing (LSH)**

  Class of algorithms that combines aspects of feature transformation with other algorithms

Feature Extraction

- TF-IDF
- Word2Vec
- CountVectorizer

Feature Extraction: Tokenizer

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer

sentenceData = spark.createDataFrame([
    (0.0, "Hi I heard about Spark"),
    (0.0, "I wish Java could use case classes"),
    (1.0, "Logistic regression models are neat")
], ["label", "sentence"])

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
wordsData = tokenizer.transform(sentenceData)

hashingTF = HashingTF(inputCol="words", outputCol="rawFeatures", numFeatures=20)
featurizedData = hashingTF.transform(wordsData)
# alternatively, CountVectorizer can also be used to get term frequency vectors

idf = IDF(inputCol="rawFeatures", outputCol="features")
idfModel = idf.fit(featurizedData)
rescaledData = idfModel.transform(featurizedData)

rescaledData.select("label", "features").show()
```

Feature Extraction: Word2Vec

```
from pyspark.ml.feature import Word2Vec

# Input data: Each row is a bag of words from a sentence or document.
documentDF = spark.createDataFrame([
    ("Hi I heard about Spark".split(" "), ),
    ("I wish Java could use case classes".split(" "), ),
    ("Logistic regression models are neat".split(" "), )
], ["text"])

# Learn a mapping from words to Vectors.
word2Vec = Word2Vec(vectorSize=3, minCount=0, inputCol="text", outputCol="result")
model = word2Vec.fit(documentDF)

result = model.transform(documentDF)
for row in result.collect():
    text, vector = row
    print("Text: [%s] => \nVector: %s\n" % (", ".join(text), str(vector)))
```

Feature Transformation

- Tokenizer
- StopWordsRemover
- *n*-Gram
- PCA
- String Indexer
- OneHotEncoder
- And many others…

Feature Transformation: StopWordsRemover

```
from pyspark.ml.feature import StopWordsRemover

sentenceData = spark.createDataFrame([
    (0, ["I", "saw", "the", "red", "balloon"]),
    (1, ["Mary", "had", "a", "little", "lamb"])
], ["id", "raw"])

remover = StopWordsRemover(inputCol="raw", outputCol="filtered")
remover.transform(sentenceData).show(truncate=False)
```

Feature Transformation: NGram

```
from pyspark.ml.feature import NGram

wordDataFrame = spark.createDataFrame([
    (0, ["Hi", "I", "heard", "about", "Spark"]),
    (1, ["I", "wish", "Java", "could", "use", "case", "classes"]),
    (2, ["Logistic", "regression", "models", "are", "neat"])
], ["id", "words"])

ngram = NGram(n=2, inputCol="words", outputCol="ngrams")

ngramDataFrame = ngram.transform(wordDataFrame)
ngramDataFrame.select("ngrams").show(truncate=False)
```

Feature Transformation: OneHotEncoder

```
from pyspark.ml.feature import OneHotEncoderEstimator

df = spark.createDataFrame([
    (0.0, 1.0),
    (1.0, 0.0),
    (2.0, 1.0),
    (0.0, 2.0),
    (0.0, 1.0),
    (2.0, 0.0)
], ["categoryIndex1", "categoryIndex2"])

encoder = OneHotEncoderEstimator(inputCols=["categoryIndex1", "categoryIndex2"],
                                 outputCols=["categoryVec1", "categoryVec2"])
model = encoder.fit(df)
encoded = model.transform(df)
encoded.show()
```

Feature Selectors

- VectorSlicer
- Rformula
- ChiSqSelector

Feature Selectors

```
from pyspark.ml.feature import VectorSlicer
from pyspark.ml.linalg import Vectors
from pyspark.sql.types import Row

df = spark.createDataFrame([
    Row(userFeatures=Vectors.sparse(3, {0: -2.0, 1: 2.3})),
    Row(userFeatures=Vectors.dense([-2.0, 2.3, 0.0]))])

slicer = VectorSlicer(inputCol="userFeatures", outputCol="features", indices=[1])

output = slicer.transform(df)

output.select("userFeatures", "features").show()
```

Locality Sensitive Hashing (LSH)

- **LSH Operations**
  - Feature Transformation
  - Approximate Similarity Join
  - Approximate Nearest Neighbor Search

- **LSH Algorithm**
  - Bucketed Random Projection for Euclidean Distance
  - MinHash for Jaccard Distance

Pipeline

- **Transformers**

  Converts through a method ***transform()*** a DataFrame into another DataFrame, typically appending one or more columns

- **Estimators**

  Converts through a method ***fit()*** a DataFrame into a Model, which is a Transformer. By calling ***fit()***, it trains a model of the specified algorithm with the specified DataFrame

Contacts

For any problem, send a mail to

daniele.foroni@unitn.it