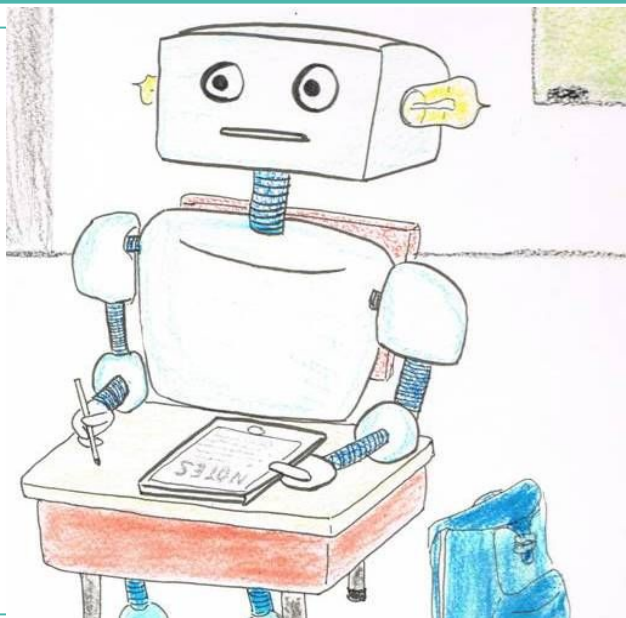




# FOR MACHINE LEARNING



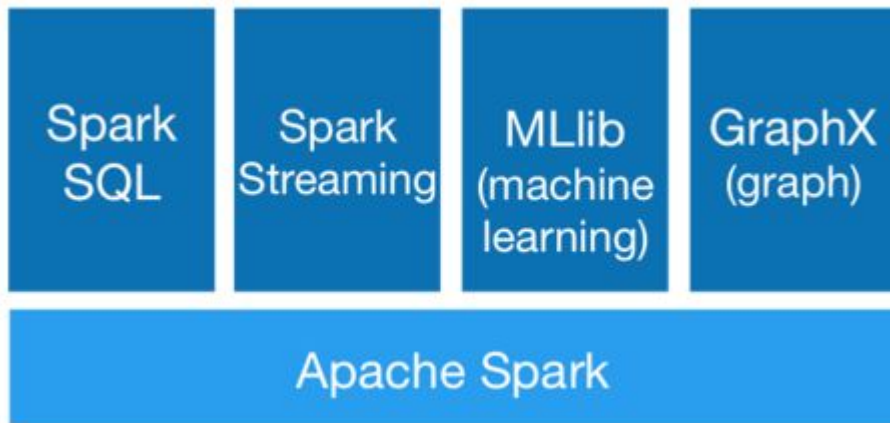
Anastasia Lieva @lievAnastazia

Julien Lafont @julien\_lafont



# Spark in a nutshell

- Traitement distribué de larges volumes de données
- 100x plus rapide qu'un job Hadoop (traitement en mémoire)
- APIs **Scala**, Java, Python & R



# Débutant sur Spark ?

- Notebooks en ligne : expérimentez facilement et gratuitement



... mais lentement !

- Ou tester en local : rien à installer, juste une dépendance maven/sbt

# Machine Learning : learn from data

**Supervisé :**

variable ciblée	variables explicatives		
Y	X1	X2	X3
OUI	1	56	65
NON	21	32	17
NON	19	24	2

**Non supervisé :**

variables explicatives		
X1	X2	X3
1	56	65
21	32	17
19	24	2

# Spark MLlib

Its goal is to make **practical** machine learning **scalable** and **easy**.

It consists of common learning algorithms and utilities, including **classification**, **regression**, **clustering**, collaborative **filtering**, **dimensionality reduction**

# MLlib & Machine Learning

Supervisé

Non supervisé

Classification et regression

Statistique

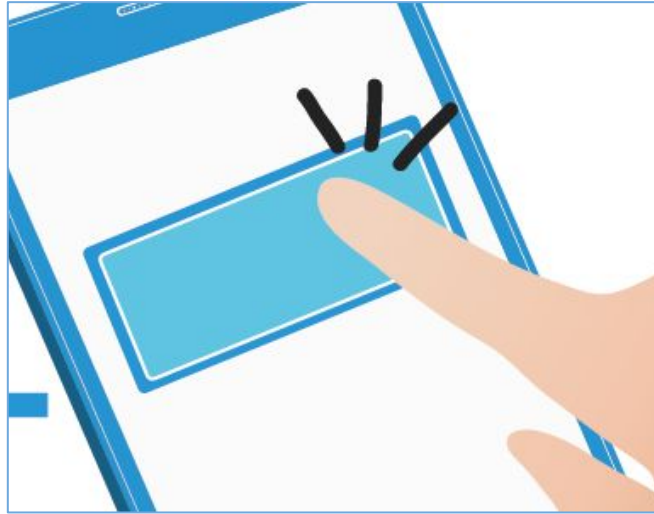
Clustering

Réduction des dimensions

Extraction des caractéristiques

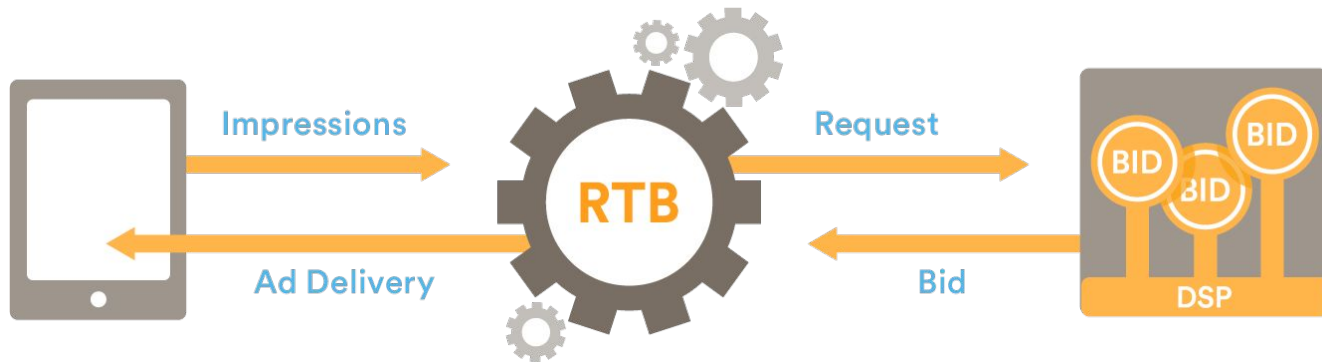
Métriques d'évaluation

# Prédictions de probabilité de clic d'une publicité



# Objectif ?

Optimiser le taux de clics sur les publicités diffusées



**Real Time Bidding : Beaucoup plus d'offre d'espace publicitaire que de pubs à diffuser.**  
On peut donc sélectionner les espaces publicitaires qui auront une plus forte probabilité de conversion



# Objectif ?

Optimiser le taux de clics sur les publicités diffusées

# Comment ?

En sélectionnant en priorité les offres d'espace publicitaire qui ont la plus forte probabilité de produire un clic

- Caractéristiques de l'espace publicitaire
- Informations sur le mobinaute
- Type de publicité attendu
- Informations extérieures

# Données

10 go  
20 000 000 observations

```
{  
  "id": "951cb9f5-2bab-46ce-b759-8245cffxxxxx",  
  "time": "2016-06-09T0:25:28Z",  
  "bidfloor": 2.88,  
  "appOrSite": "app",  
  "adType": "banner",  
  "categories": "IAB12,IAB1-3,IAB13-1,news,football",  
  "publisherId": "10d2e69ab36811e281c1123139xxxxx",  
  "carrier": "208-10",  
  "os": "iOS",  
  "connectionType": 3,  
  "coords": [48.929256439208984, 2.4255824089050293],  
  "adSize": [320, 50],  
  "exchange": "xxxxx",  
  [...],  
  "clicked": true  
}
```

# Données

## variables quantitatives

```
{  
  "id": "951cb9f5-2bab-46ce-b759-8245cffxxxxx",  
  "time": "2016-06-09T0:25:28Z",  
  "bidfloor": 2.88,  
  "appOrSite": "app",  
  "adType": "banner",  
  "categories": "IAB12,IAB1-3,IAB13-1,news,football",  
  "publisherId": "10d2e69ab36811e281c1123139xxxxx",  
  "carrier": "208-10",  
  "os": "iOS",  
  "connectionType": 3,  
  "coords": [48.929256439208984, 2.4255824089050293],  
  "adSize": [320, 50],  
  "exchange": "xxxxx",  
  [...],  
  "clicked": true  
}
```

# Données

## variables qualitatives

```
{  
  "id": "951cb9f5-2bab-46ce-b759-8245cffxxxxx",  
  "time": "2016-06-09T0:25:28Z",  
  "bidfloor": 2.88,  
  "appOrSite": "app",  
  "adType": "banner",  
  "categories": "IAB12,IAB1-3,IAB13-1,news,football",  
  "publisherId": "10d2e69ab36811e281c1123139xxxxx",  
  "carrier": "208-10",  
  "os": "iOS",  
  "connectionType": 3,  
  "coords": [48.929256439208984, 2.4255824089050293],  
  "adSize": [320, 50],  
  "exchange": "xxxxx",  
  [...],  
  "clicked": true  
}
```

# Variable ciblée      binaire (true/false)

```
{  
  "id": "951cb9f5-2bab-46ce-b759-8245cffxxxxx",  
  "time": "2016-06-09T0:25:28Z",  
  "bidfloor": 2.88,  
  "appOrSite": "app",  
  "adType": "banner",  
  "categories": "IAB12,IAB1-3,IAB13-1,news,football",  
  "publisherId": "10d2e69ab36811e281c1123139xxxxx",  
  "carrier": "208-10",  
  "os": "iOS",  
  "connectionType": 3,  
  "coords": [48.929256439208984, 2.4255824089050293],  
  "adSize": [320, 50],  
  "exchange": "xxxxx",  
  [...],  
  "clicked": true  
}
```

# Modèle de machine learning supervisé

Definir la structure mathématique pour prédire **Y** en fonction de **X**



$\text{Clic} = \text{Fonction}(\text{iOS} * \beta_1, \text{CreaSize} * \beta_2, \text{AppOrSite} * \beta_3)$

Trouver les paramètres inconnues de cette structure



$\beta_1, \beta_2, \beta_3$

Minimiser/maximiser la fonction objectif

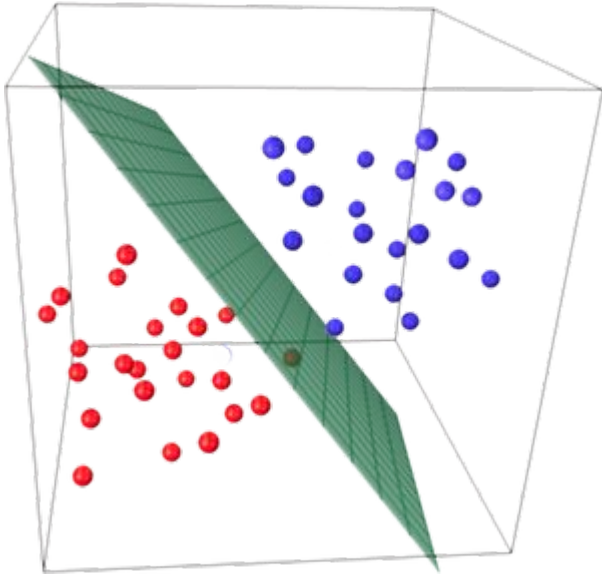
Erreur, logloss, probabilité

variable ciblée	variable explicatives		
Clic	os	creaSize	appOrSite
OUI	iOS	small	app
NON	Android	large	app
NON	iOS	large	site

# Algorithmes de classification binaire

Avantages	Régression logistique	Gradient Boosted Trees
Scalabilité	✓	
Parallelisme	✓	
Simplicité	✓	
Vitesse	✓	
Performance de prédiction		✓

# Régression logistique



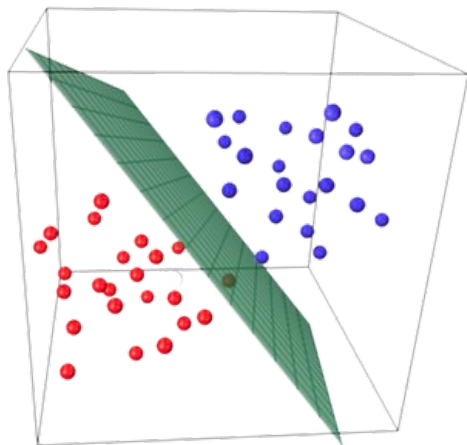
OUI, cet utilisateur va clicker



NON, cet utilisateur ne va pas clicker



# Régression logistique



$$t = (\text{app} \cdot \beta_1, \text{android} \cdot \beta_2, \text{wifi} \cdot \beta_3)$$



$$OR_+ = e^t$$



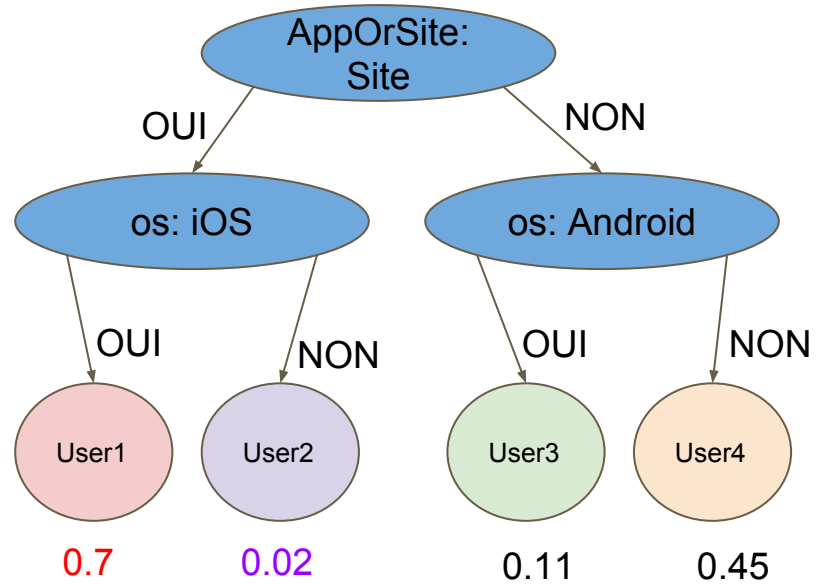
$$P_+ = \frac{OR_+}{1 + OR_+}$$

$P_+$  est probabilité que l'utilisateur va cliquer sur la publicité

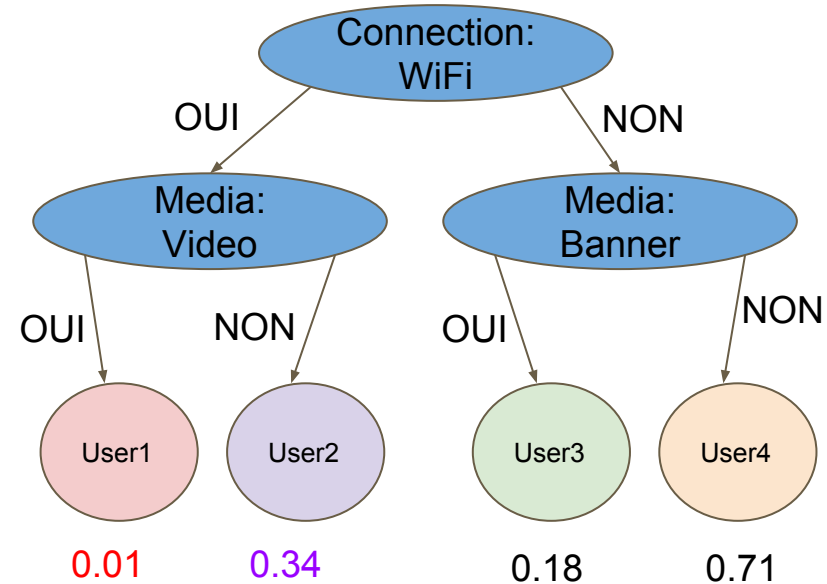
app	android	wifi
1	1	1
0	0	0
1	0	0
1	1	0
1	0	0
1	1	0
1	1	1
1	0	0
0	1	0
1	1	0

# Gradient Boosted Trees

**Arbre 1**



**Arbre 2**



$$F(\text{User1}) = 0.7 + 0.01 = 0.71$$

$$F(\text{User2}) = 0.02 + 0.34 = 0.36$$

# Algorithmes de classification binaire

	Régression logistique	Gradient Boosted Trees
Gestion des variables qualitatives		✓
Gestion de grande nombre des variables explicatives	✓	
Gestion des données éparses (sparsity)	✓	

# Pre-processing des données

Requête	appOrSite	os	carrier	connection
1	app	Android	Orange	WIFI
2	site	iOS	Free	3G
3	app	iOS	BT	4G
4	app	Android	BT	3G
5	app	Bada	Free	3G
6	app	Android	Orange	4G
7	app	Android	SFR	WIFI
8	app	iOS	SFR	3G
9	site	Android	Orange	3G
10	app	Android	Free	4G

# Pre-processing différent suivant l'algo

## Régression logistique

Création des variables  
numériques indépendantes  
(dummy variables)

app	site	android	ios	bada	orange	free	sfr	wifi	3g	4g
1	0	1	0	0	1	0	0	1	0	0
0	1	0	1	0	0	1	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0	1	0
1	0	1	0	0	1	0	0	0	0	1
1	0	1	0	0	0	0	1	1	0	0
1	0	0	1	0	0	0	1	0	1	0
0	1	1	0	0	1	0	0	0	1	0
1	0	1	0	0	0	1	0	0	0	1

VS

## Gradient boosted trees

Indexation des variables  
catégorielles

appOrSite	os	carrier	connection
0	0	0	0
1	1	1	1
0	1	2	2
0	0	2	1
0	2	1	1
0	0	0	2
0	0	3	0
0	1	3	1
1	0	0	1
0	0	1	2

# Pre-processing: String indexer

```
> val indexer = new StringIndexer()  
  .setInputCol("os")  
  .setOutputCol("osIdx")  
  .fit(df)  
  
val indexed: DataFrame = indexer.transform(df)
```

timestamp	os	osIdx
1465037789	iOS	1
1464983457	WindowsPhone	2
1465019529	Android	0
1464974567	iOS	1
1465018552	Android	0

Gradient boosted trees

## StringIndexer

Indexation des variables catégorielles (transformation des données texte en valeur numérique)

# Pre-processing: One Hot Encoder

Régression logistique

```
> val oneHotEncoder = new OneHotEncoder()  
    .setInputCol("osIdx")  
    .setOutputCol("osVec")  
  
val encoded = oneHotEncoder.transform(indexed)
```

## OneHotEncoder

Transformation des variables  
catégorielles en colonnes  
binaires

timestamp	os	osIdx	android	windows	ios	bada
1465037789	iOS	1	0	0	1	0
1464983457	Windows	2	0	1	0	0
1465019529	Android	0	1	0	0	0
1464974567	iOS	1	0	0	1	0
1465018552	Bada	3	0	0	0	1

# Pre-processing: Bucketizer

Regrouper des valeurs discrètes sur des intervalles donnés

Ex: classifier les requêtes en fonction du moment de la journée  
*0h-7h, 7h-12h, 12h-14h, 14h-18h, 18h-24h*

```
> val bucketizer = new Bucketizer()
  .setInputCol("hour")
  .setOutputCol("hourBucket")
  .setSplits(Array(0.0, 7.0, 12.0, 14.0, 18.0, 24.0))

val bucketized = bucketizer.transform(df)
```

Gradient boosted trees

Régression logistique

timestamp	hour	hourBucket
1465037789	4	0
1464983457	18	4
1465019529	7	1
1464974567	17	3
1465018552	21	4



# Hashing Term Frequency

Génère un vecteur à taille fixe à partir d'une suite de termes. Les termes ayant la plus forte fréquences sont gardées en priorité

```
> val tokenizer = new RegexTokenizer()  
  .setInputCol("cats")  
  .setOutputCol("catsToken")  
  .setPattern("""[-\\w]+""").setGaps(false)  
  
val tokenized = tokenizer.transform(df)  
  
val hasherTF = new HashingTF()  
  .setInputCol("catsToken")  
  .setOutputCol("catsVector")  
  .setNumFeatures(200)  
  
val hashed = hasherTF.transform(tokenized)
```

categories	categoriesToken	categoriesVec
IAB9,IAB9-30,games	["iab9","iab9-30","games"]	[0,200,[59,167,185],[1,1,1]]
IAB1,IAB3,entertainment,utilities	["iab1","iab3","entertainment","utilities"]	[0,200,[159,161,192,194],[1,1,1,1]]
IAB1,IAB17,entertainment,sports	["iab1","iab17","entertainment","sports"]	[0,200,[87,159,184,192],[1,1,1,1]]
IAB12,IAB13	["iab12","iab13"]	[0,200,[179,180],[1,1]]

# Extracting, transforming and selecting features



## Feature Extractors

HashingTF and IDF  
Word2Vec  
CountVectorizer

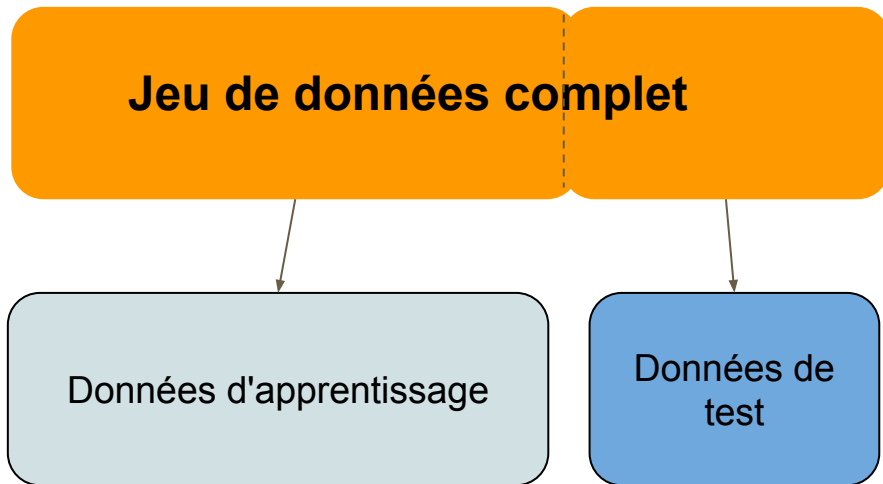
## Feature Transformers

Tokenizer  
StopWordsRemover  
Binarizer  
PCA  
PolynomialExpansion  
Discrete Cosine Transform  
VectorIndexer  
Normalizer  
StandardScaler  
MinMaxScaler  
ElementwiseProduct  
SQLTransformer  
VectorAssembler  
QuantileDiscretizer

## Feature Selectors

VectorSlicer  
RFormula  
ChiSqSelector

# Division du jeu des données



```
val splits = labeled.randomSplit(Array(0.8, 0.2))
val training = splits(0)
val test = splits(1)
training.persist
test.persist
```

# Entraînement du modèle de prédiction

Gradient boosted trees






## Tuning approfondi des paramètres

```
> val strategy = BoostingStrategy.defaultParams("Classification")
  strategy.numIterations = 100
  strategy.treeStrategy.numClasses = 2
  strategy.treeStrategy.maxDepth = 15
  strategy.treeStrategy.impurity = entropy
  strategy.treeStrategy.maxBins = maxCategoricalCardinality

val model = new GradientBoosterTrees(strategy)
  .run(trainingData)
```

# Evaluation du modèle sur les données de test

```
> val expectedWithPredictedLabels = testData.map { case LabeledPoint(label, features) =>
  val expectedResult = label
  val predictedResult = model.predict(features)
  (expectedResult, predictedResult)
}
```

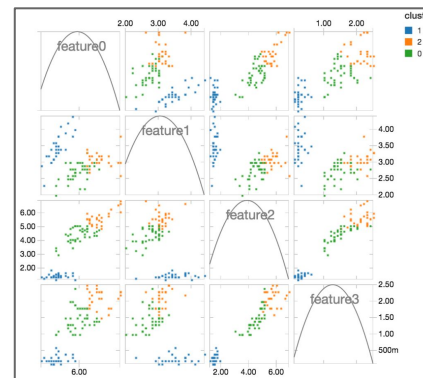
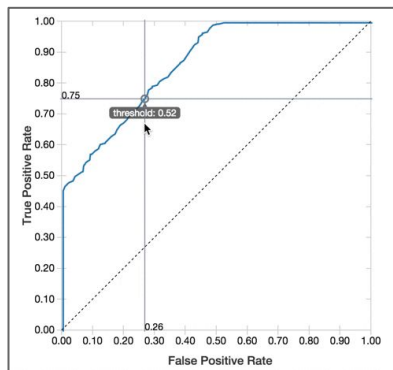
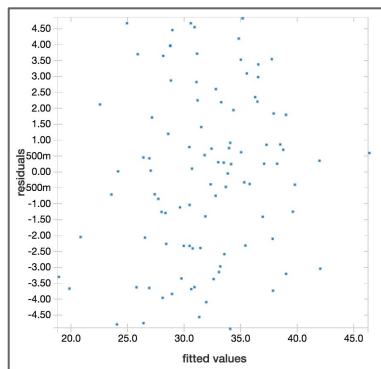
expected	predicted	
1.0	1.0	
1.0	0.0	
0.0	1.0	
0.0	1.0	
0.0	0.0	

# Métriques d'évaluation

Évaluation des modèles de classification :

- **Binary**ClassificationMetrics
- **Multiclass**Metrics
- **Multilabel**Metrics
- **Ranking**Metrics

> Precision, Recall, F1-measure, LogLoss, AUC, confusion Matrix...

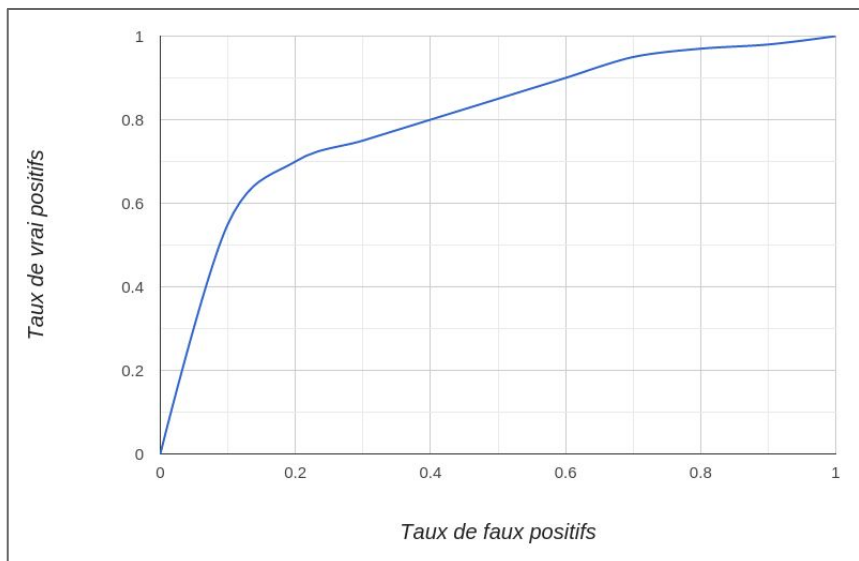


# Métriques d'évaluation

Métrique : **AUC** (Area Under the ROC)

```
val metrics = BinaryClassificationMetrics(predictions)
val auROC = metrics.areaUnderROC()
```

Courbe ROC (Receiver Operating Characteristic)



Taux de vrais positifs :

$$= \text{Vrai positif} / (\text{Vrai Positif} + \text{Faux Negatif})$$

Taux de faux positifs :

$$= \text{Faux Positif} / (\text{Faux Positif} + \text{Vrai Positif})$$

# Résultats expérimentaux

	Logistic Regression LBFGS	Logistic Regression SGD	Gradient Boosted Trees
Temps d'apprentissage	1 heure	1 heure	8 heures
% de clics prédits	40%	45%	60%

Hardware (Amazon EC2 c4.8xlarge)

- **60go** RAM
- **32** vCPUs (2.9 - 3.5 ghz)
- Prix : **2.0\$** / heure (réservé)  
ou **0.5\$** / heure (à la demande)

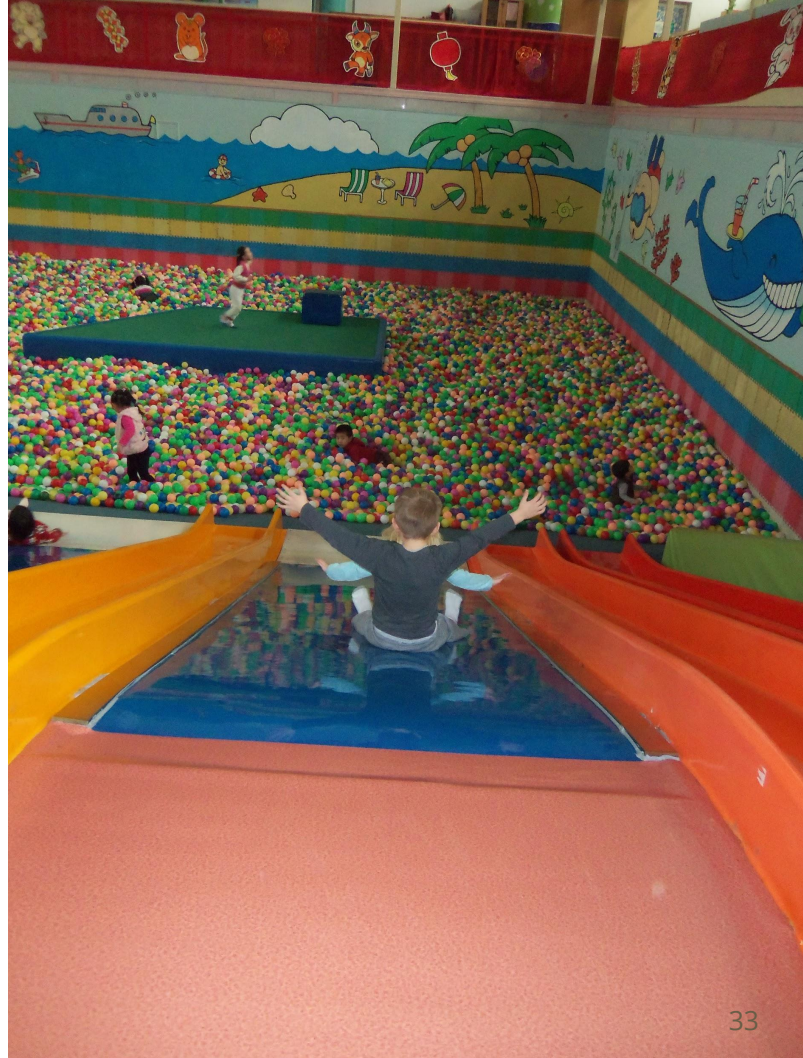


# Au fait ?

TabMo-Eng recrute sur Montpellier

- Data-engineers
- Dev Scala
- Dev JS
- Dev iOS
- .....

[jobs@tabmo.io](mailto:jobs@tabmo.io)



# TIPS: Production en Spark

- Si vous pensez avoir assez de RAM, doublez là !
- Partitionnement des tasks en fonction du nombre de coeurs CPU
- Optimisez l'empreinte mémoire de votre application
  - (Kryo Serialization, variable broadcasting, light data-structure)
- Tips: Amazon Spot Instances
  - 36 coeurs, 64go ram, SSD pour ~ 0.5\$ / heure