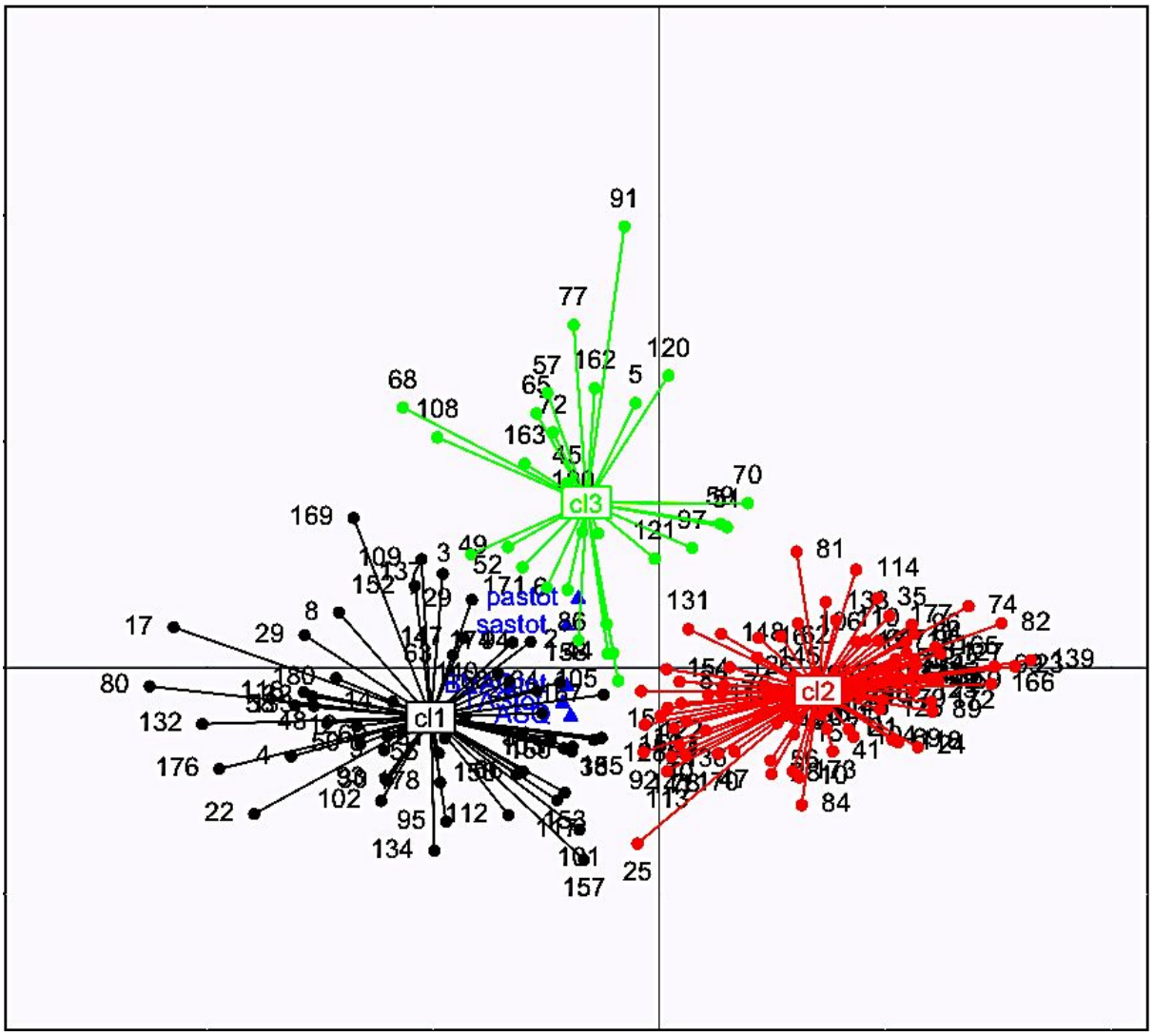
Projet Data Analytics

Aaron LELLOUCHE
Raphaël UZAN
Badr LAKHDARI



Université Paris Dauphine
Juin 2019

Table des matières :

Exercice 1: Implémentation de K-means avec PySpark

La classification en k-means est une méthode de quantification vectorielle, issue du traitement du signal, couramment utilisée pour l'analyse de grappes dans l'exploration de données. La classification k-means a pour but de partitionner n observations en k grappes dans lesquelles chaque observation appartient à la grappe dont la moyenne est la plus proche, servant de prototype à la grappe.

I) Présentation de l'implémentation

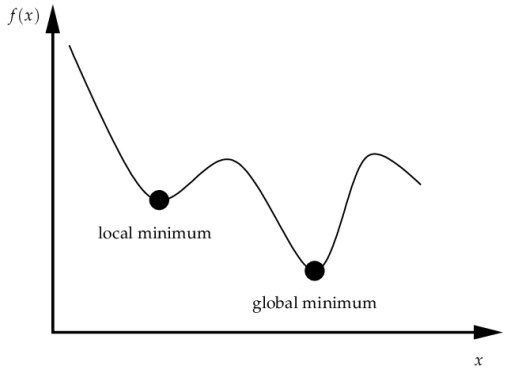
a/ Méthodes

- def loadData(path): Charge les données sous forme de rdd.
- def initCentroids_withsample(data, nbrClusters): Voir (I-b)
- def initCentroids_with_random_rdd(data): Voir (I-b)
- def distance_minimum(elem): Prend en entrée un point ainsi que sa distance à chacun des centroids et retourne l'identifiant du centroids le plus proche et sa distance.
- def assignToCluster(rdd, centroids): Affecte à chaque point l'identifiant du centroid le plus proche de lui
- def distance(x,y): Calcul la distance entre deux éléments.
- def calculate_barycentre(num_centroid, rdd_item): Calcul le barycentre pour un ensemble de vecteurs données.
- def computeCentroids(rdd_item):Calcul les coordonnées du centroid en agrégeant les données par identifiants de centroid et en calculant leurs barycentre.
- def computeIntraClusterDistance(rdd_item): Calcul la distance moyenne entre chaque point et son centroid

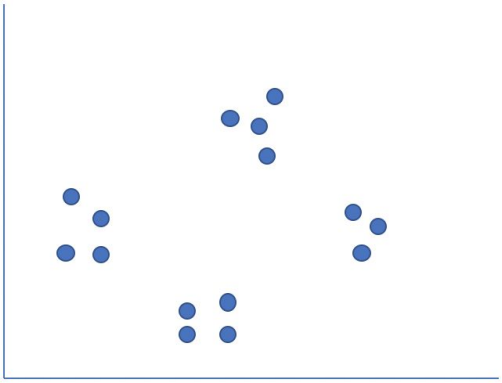
On peut notamment noter que l'exécution à chaque itération était de plus en plus longue, la solution pour éviter les ralentissement a été de rajouter un => parallelize() à chaque affectation des nouveaux centroïdes dans les centroïdes “courant” afin de distribuer proprement nos éléments.

b/ Choix de l’initialisation des centroïdes

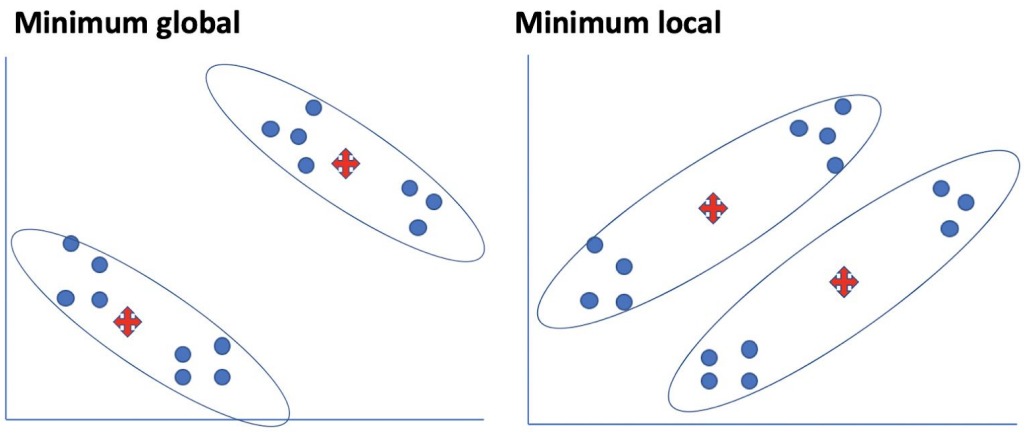
Le choix de l’initialisation des centroids est centrale dans Kmeans, cela est due au faite que Kmeans cherche à minimiser la distance intra-cluster par itération successive cependant comme on peut le voir dans l’image ci-dessous on peut se retrouver sur un minimum locale et avoir une convergence (Voir I-c) alors que le minimum global se trouve ailleurs.



Pour illustrer ce problème dans le cas de Kmeans on se donne les données suivantes que l’on souhaite classifier en 2 classes:



Voici ce que pourrait être une convergence n’ayant pas le le score minimum de distance intra-cluster et un autre qui l’aurait.



Pour limiter ce problème il faut choisir une initialisation qui n'induit pas de biais et limite ainsi les risques de converger vers un minimum local. De plus une fois mise en place une initialisation des centroids "aléatoire" on a de forte chance de trouver le minimum global si on lance un grand nombre de fois Kmeans.

Dans notre implémentation de l'initialisation des centroids nous avons implémenté deux techniques :

- La première utilise un échantillon de points choisis aléatoirement dans nos données
- La deuxième choisit des valeurs aléatoires dans des borne pré-calculé du max et min pour chaques coordonnées.

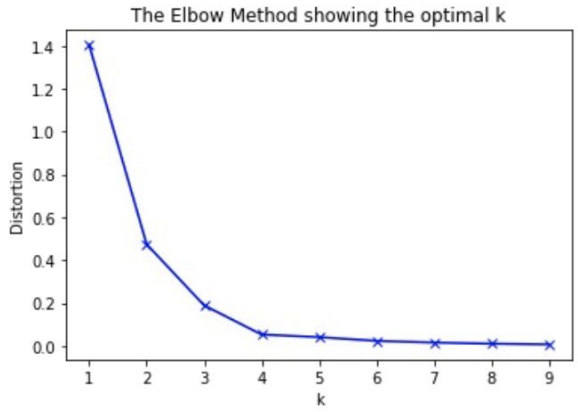
c/ Convergence

Nous avons choisis comme critère de convergence de compter le nombre d'éléments ayant changer de centroids entre deux itérations si ce nombre est égal à zéro alors on a une stabilité et on s'arrête.

d/ Choix du paramètre k

Kmeans laisse à l'utilisateur le choix du nombre k de clusters. Il n'existe pas de procédé automatique pour trouver la meilleur valeur de k a utiliser, cependant une technique couramment utilisée consiste a calculer la moyenne des distances intra-cluster pour différentes valeur de k comme on peut le voir ci-dessous. On utilisera alors la valeur de k a partir de laquelle l'ajout d'un cluster ne diminue plus beaucoup la moyenne des distances intra-cluster. (Ci-dessous[1] dans notre exemple on pourrait prendre 3 ou 4 par exemple).

[1] https://mrmint.fr/algorithme-k-means



II) Résultat avec 100 exécutions

```
********** resultat **********
moyenne : 1.9601267744729711
ecart-type : 0.05034285464202481
min : 1.9068020893187687
max : 2.0878002484639486
mediane :1.9470866444071784
********** iteration :  **********
moyenne : 7.4
ecart-type : 2.998316025688251
min : 3
max : 16
mediane :7.0
```

Nous voyons que nous avons une **moyenne de distance intra-cluster** de 1,960; l'écart-type est de 0,050, les données appartiennent à l'intervalle [1,90;2,09] enfin la médiane de la moyenne intra cluster est de 1,947.

Pour ce qui est maintenant du nombre d'itérations on en a en moyenne 7,4; écart-type de 2,998; le nombre d'itération est compris dans l'intervalle [3;16] et la médiane est 7.

On souhaite maintenant comparer les résultats de cette initialisation avec une initialisation choisissant des valeurs aléatoire dans les bornes [Min;Max] de la colonne pour 100 initialisations

```
********** resultat **********
moyenne : 1.7531489274069865
ecart-type : 0.1836790191396266
min : 1.5896564966694111
max : 2.1280566905484055
mediane :1.5896564966694111
********** iteration :  **********
moyenne : 4.5
ecart-type : 2.4638804903458835
min : 2
max : 14
mediane :4.0
```

Voici les résultats pour **la distance moyenne intra-cluster** (en vert les valeurs les plus basses) **:**

distance moyenne intra-cluster	Initialisation avec échantillon aléatoire	Initialisation avec des valeur aléatoire choisis dans les bornes [Min;Max]
Min	1.9068	1.590
Max	2.0878	2.128
Moyenne	1.960	1.753
ecart-type	0.050	0.184
Médiane	1.947	1.590

Voici les résultats pour le **nombre d'itérations :**

nombre d'itérations	Initialisation avec échantillon aléatoire	Initialisation avec des valeur aléatoire choisis dans les bornes [Min;Max]
Min	3	2
Max	16	14
Moyenne	7.4	4.5
ecart-type	2.998	2.464
Médiane	7	4

On voit donc grâce à ces résultats que la méthode d'initialisation par valeur aléatoire et n'utilisant pas d'échantillons a sur 100 exécutions une meilleur moyenne de distance intra-clusters (1,753 < 1,960) et vas donc plus rarement trouver un minimum local.
De plus le nombre d'itération est bien plus court (4,5 contre 7,4 avec les échantillon
La deuxième initialisation nous apparaît ainsi beaucoup plus efficace et sera à privilégier.

Une autre solution d'initialisation des centroïdes plus complexe à mettre en place est **K-means++**[2], proposé en 2007 par David Arthur et Sergei Vass qui consiste a choisir un premier centroïde de manière aléatoire uniforme parmis les points de données, ensuite chaque centroïde supplémentaire est choisi parmi les points de données restants avec une probabilité en proportion de sa distance au carré du centre de cluster existant le plus proche du point.
Par manque de temps nous n'avons pas pu implémenter cette solution cependant certains articles montrent que c'est une solution efficace[3].

[2] https://en.wikipedia.org/wiki/K-means%2B%2B
[3] https://math.unice.fr/~binard/kmeans_vs_kmeanspp.pdf

III) Comparaison avec Kmeans de MLlib

```
********** resultat **********
>>> stats_res(resultat)
moyenne : 1.9468653930545745
ecart-type : 0.02079912778914069
min : 1.9068017747045611
max : 2.0878000577290856
mediane :1.9421231267569627
```

Nous avons exécuté 100 fois Kmeans avec la bibliothèque MLlib.

Nous ne pouvons pas tirer de conclusions sur le nombre d'itération car l'information n'est pas retourné par MLlib.
Cependant nous avons pu récupérer les coordonnées des centroïdes (dans la variable centers) retourner par MLlib de la façon suivante :

```
kmeans = KMeans().setK(3).setSeed(randint(0,2000))
model = kmeans.fit(dataset)
centers = model.clusterCenters()
```

et ensuite appliquer la fonction **assignToCluster()** suivis de **computeIntraClusterDistance()** et on obtient alors la distance intra-cluster comme calculé précédemment, ce qui donne :

distance moyenne intra-cluster	Initialisation avec échantillon aléatoire	Initialisation avec des valeur aléatoire choisis dans les bornes [Min;Max]	Résultat avec ML lib
Min	1.9068	1.590	1,907
Max	2.0878	2.128	2,088
Moyenne	1.960	1.753	1,947
ecart-type	0.050	0.184	0,021
Médiane	1.947	1.590	1,942

On remarque alors que la moyenne est très proche de ce que l'on a pu trouver avec les échantillons aléatoires.

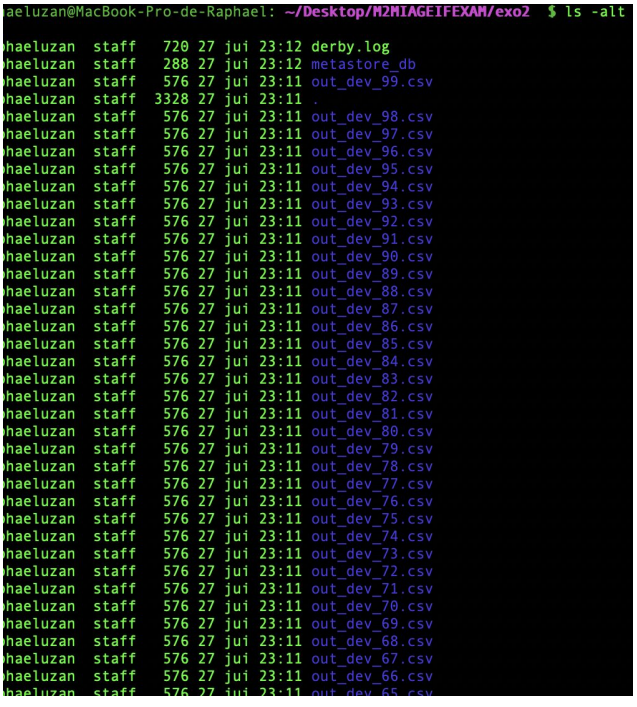
Exercice 2: Générateur de données & analyse de performance de Kmeans sur différentes distribution de données

I) Présentation de l'implémentation

II) Comparaison de Kmeans sur différentes distribution de données

a/ Comment varie la wsse quand l'écart-type augmente?

Afin de répondre à cette question nous avons généré[4] des données avec un écart-type allant de 1 à 100.
Voici un échantillon de nos 100 fichiers généré par notre generator.py



Voici un exemple de données généré

[4] Git : http://generator2_diff_ecart_type.py

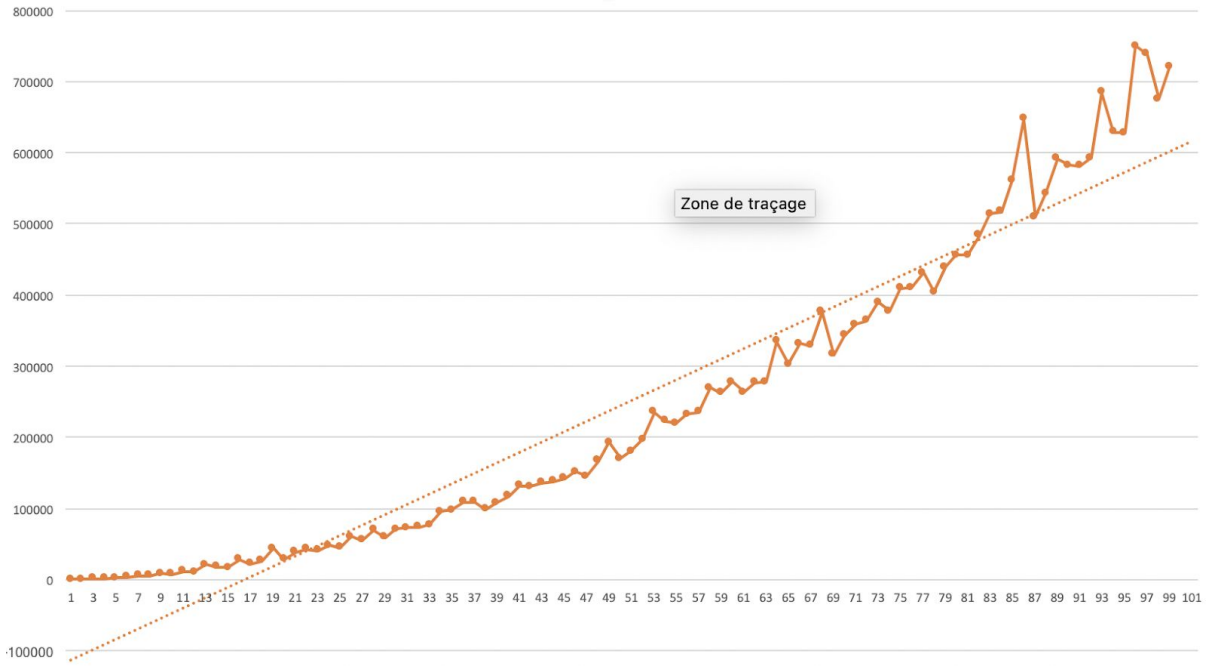
```
62 [23:23:11] raphaeluzan@MacBook-Pro-de-Raphael: ~/Desktop/M2MIAGEIFEXAM/exo2  $ cat out_dev_3.csv/part-0000*
2.89067462132202,61.06118043384157,41.78949973729257,35.738297464340l6,0
6.15114755125256,53.97647312842951,40.947673120971764,29.556537666432774,0
5.63844337798737,53.38018841942552,44.44845381510028,28.428382153753425,0
1.38938405746l,58.39367945916849,38.94594578724901,31.94587998389958,0
4.48381854729915,75.00996228291945,36.6068143277502026,9.53879297502777,1
0.88005190917707,74.53008432530l2,26.55716241924551,8.598832296643737,1
8.18713170271653,76.97235520226303,32.69365280651072,16.6222153761449,1
0.588346403228414,77.14213912493508,29.506955372529617,14.082881327424124,1
9.75233479908801,74.6245639175859,28.483106534740l2,14.562747474717732,1
3.12506434663163,74.79471689112799,35.53150420731699,15.494853871338062,1
9.21284411784019,78.75104931600715,33.782293888093086,11.176917052133588,1
2.62610420824097,80.17176131253315,31.827508467472104,18.752060498362297,1
1.377248593595,72.80501165037602,29.831655893682946,15.973043067111082,1
5.27305468855262,63.19701036799597,42.64987213921528,31.392025419547615,2
6.93036090400285,61.699879455101744,43.22326909299991,33.852563815025604,2
5.1168842719151,52.03653050244226,46.02531431131l46,29.703214617886278,2
6.47675980341361,60.72174311748504,39.42713956734092,31.191277912855718,2
6.09444012667745,50.06467334630517,43.95789747826662,32.545124546004416,2
7.84143594387515,56.86846936446851,35.93886423724022,30.84873765729655,2
6.57512494463339,65.25276137339785,40.96131474133791,32.938903799611436,2
1.594698681682594,62.356225983431784,45.93372220033738,31.18474270865898,2
4.71299256339l6,59.83063901857089,41.971527253555l4,31.18740688738422,2
6.86390829889253,55.501213655029844,40.98358923837321,31.252656644866285,2
6.87971629575843,59.16559467042762,40.967679088810286,34.9109487073046,2
7.01366423281955,55.43683857114228,39.68880112900576,28.513869730726395,2
6.09373241236565,57.56631474298348,40.2255595708725,36.534312825798,2
9.377075557723366,59.497029033171096,41.098335359448065,32.22004618256138,2
0.69161587094707,59.511632099880l1,37.22928661948849,28.35026922965166,2
8.08856735453985,68.7671859616944,43.141255249328395,29.582933320691662,2
0.23304661762684,62.220931308364506,43.69575241913057,27.50200467819257,2
63 [23:23:22] raphaeluzan@MacBook-Pro-de-Raphael: ~/Desktop/M2MIAGEIFEXAM/exo2  $
```

La WSSE est calculé de la manière suivante[5] :

$$\sum_{k=1}^{K} \sum_{i \in S_k} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{kj})^2$$

where S_k is the set of observations in the kth cluster and $\bar{x}_{kj}$ is the jth variable of the cluster center for the kth cluster.

Puis nous avons fait une analyse de la wsse retourné par Kmeans pour ces différents jeux de données générés. Enfin nous avons affiché dans un graphe avec une régression linéaire le résultat :



On voit ainsi que l'augmentation de l'écart type donne un WSSE augmentant de manière exponentielle car assez rapidement les points s'entre-croise.

5

https://discuss.analyticsvidhya.com/t/what-is-within-cluster-sum-of-squares-by-cluster-in-k-means/2706/2

Voici les autres données (fixé) utilisé pour générer nos données :

```
pts = int(30) # number of pts to be generated
k = int(3) # number of rdd
dim = int(4) # dim of the data
```

b/ Écart-type fort sur un centroïde et faible sur l'autre

Nous avons à nouveau calculé la WSSE entre deux cluster généré par generator.py, le premier à un écart-type de 1 et le deuxième a un écart-type de 3.

```
>>>
>>> print("WSSE avec deux centroide d'écart-type 1")
WSSE avec deux centroide d'écart-type 1
>>> print("Within Set Sum of Squared Errors = " + str(wssse1))
Within Set Sum of Squared Errors = 39.5368979345
>>>
>>>
>>> print("WSSE avec un centroide d'écart-type 1 et l'autre d'écart-type 4")
WSSE avec un centroide d'écart-type 1 et l'autre d'écart-type 4
>>> print("Within Set Sum of Squared Errors = " + str(wssse2))
Within Set Sum of Squared Errors = 103.9467530928
>>>
```

On voit donc que Kmeans à beaucoup de difficulté à classifier des données dont les écarts-type autour des centroïdes des différentes classe est variable.

Exercice 3 : Facteurs de réussite aux examens chez les étudiants

I) Contexte

a/ Dataset

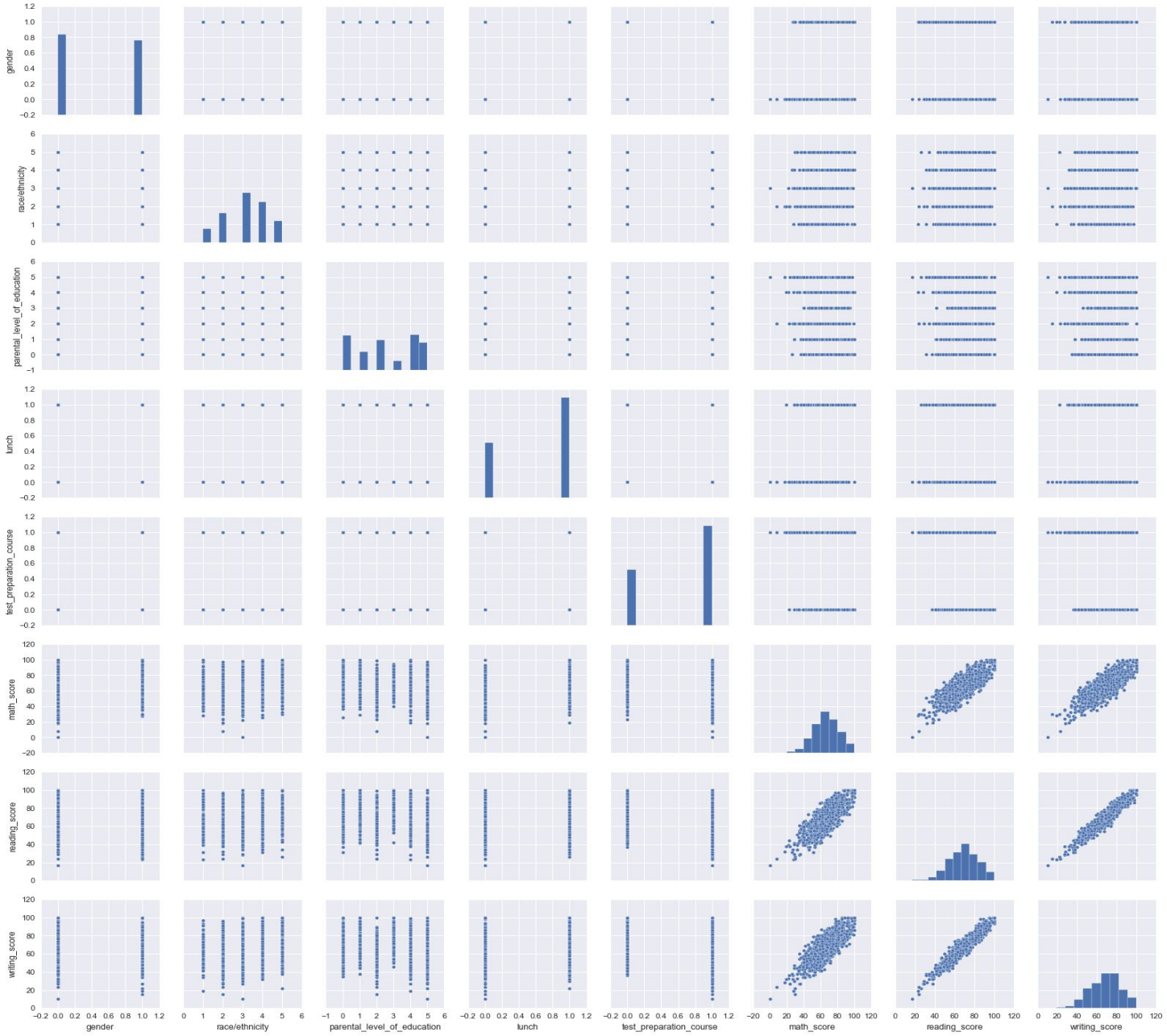
Nous avons téléchargé ce jeu de données sur Kaggle[6]. Celui-ci comprend les notes obtenues par des étudiants dans diverses matières avec des informations sur le profil des différents étudiants.

- ➔ gender : Homme ou Femme.
- ➔ race/ethnicity : Nous avons 4 groupes distincts contenant 4 ethnies différentes.
- ➔ parental_level_of_education : Il s'agit du niveau d'étude des parents.
- ➔ lunch : Permet de savoir si l'étudiant est boursier à la cantine ou non.
- ➔ test_preparation_course : La note sur 100 obtenue par l'étudiant à un examen préparatif.
- ➔ math_score : La note sur 100 obtenue par cet étudiant en Mathématique.

[6] https://www.kaggle.com/spscientist/students-performance-in-exams

- ➔ reading_score : La note sur 100 obtenue par cet étudiant dans une matière ici nommé de “reading”.
- ➔ writing_score : La note sur 100 obtenue par cet étudiant dans une matière ici nommé de “writing”.

b/ Matrice de corrélation



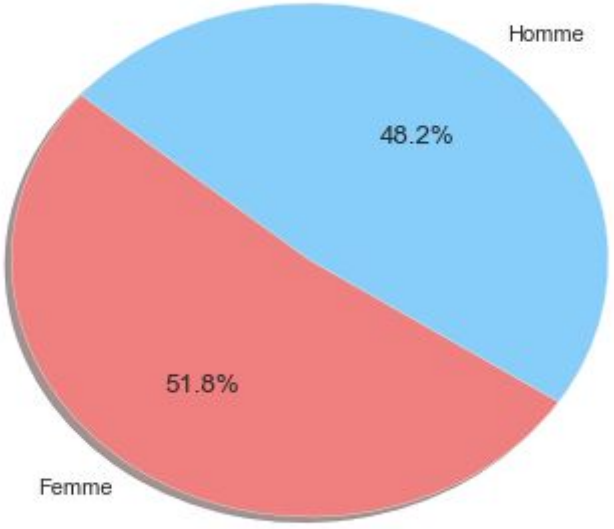
On remarque une bonne corrélation entre les différentes notes : Cela nous montre qu’il existe bien des “classes” d’étudiant qui réussissent et une classe d’étudiant ayant plus de difficultés.

c/ Data-visualisations

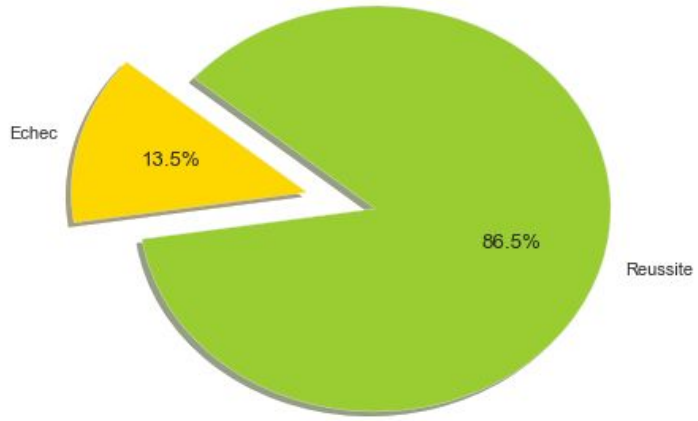
Distribution de valeurs entières :

	math_score	reading_score	writing_score
count	1000.00000	1000.000000	1000.000000
mean	66.08900	69.169000	68.054000
std	15.16308	14.600192	15.195657
min	0.00000	17.000000	10.000000
25%	57.00000	59.000000	57.750000
50%	66.00000	70.000000	69.000000
75%	77.00000	79.000000	79.000000
max	100.00000	100.000000	100.000000

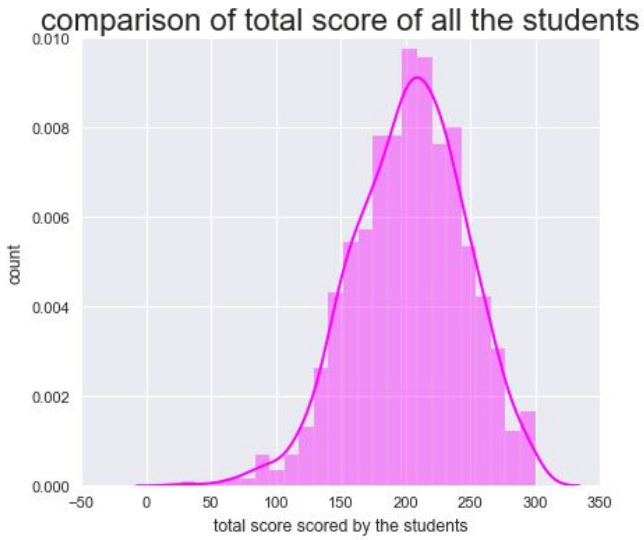
<u>Proportion d'Hommes et de femmes dans notre dataset :</u>



Nombre d'échecs et de réussites au test de math (réussite si note > 50) :



Visualisation de la dispersion des notes (Somme des 3 notes) :



d/ Préparation des données

Initialement un échantillon de données ressemble à ça :

Index	gender	race/ethnicity	ntal_level_of_educ	lunch	t_preparation_cou	math_score	reading_score	writing_score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44

Malheureusement les données tel quel ne peuvent pas être analysé par un algorithme de machine learning. En effet il est nécessaire de n'avoir que des données de type numérique.

Pour préparer les données nous avons fait le choix d'utiliser Pandas et Numpy avant de remettre sous format csv. Ensuite nous pourrons appliquer nos modèles de Machine learning avec Spark MLlib en chargeant ce fichier csv

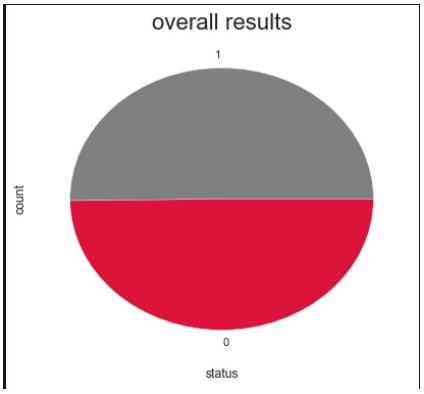
Une fois nos transformations appliqué voici le format de sortie de nos données :

Pour le problème de Classification :

Index	gender	race/ethnicity	ıtal_level_of_educ	lunch	t_preparation_cou	status
0	0	2	1	1	1	0
1	0	3	4	1	0	0
2	0	2	3	1	1	0
3	1	1	0	0	1	1
4	1	3	4	1	1	0

Status étant la variable cible généré de la manière suivante :
0 if x['total_score'] >205 else 1
Autrement dit si la somme des 3 notes de l'étudiant dépasse 205 alors mettre la valeur 0 sinon mettre la valeur 1.

Pourquoi le choix de cette valeur 205 car c'est la médiane et cela nous permet de couper notre dataset en deux parties équivalentes comme on peut le voir ci-dessous :



Pour le problème de Régression :

Ici nous allons essayer de prédire la note de l'étudiant en math a partir des autres informations disponible. Voici un exemple de donnée de sortie ci-dessous :

Index	gender	race/ethnicity	ıtal_level_of_educ	lunch	test_preparation_course	pass_reading	pass_writing	math_scoree
0	0	2	1	1	1	1	1	72
1	0	3	4	1	0	1	1	69
2	0	2	3	1	1	1	1	90
3	1	1	0	0	1	1	0	47

III) Tentative de prédiction de la réussite des étudiants

a/Classification

On va essayer à l'aides de modèle de Machine learning de classifier[7]si un étudiant est dans la classe réussite ou dans la classe échec. Pour cela nous avons utilisé la bibliothèque MLlib [8] de Spark en Python

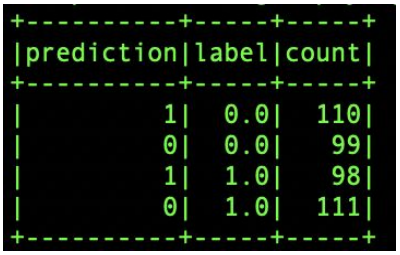
Voici un échantillon de code qui montre comment faire apprendre le modèle avec nos données et récupérer la matrice de confusion et l'accuracy :

```
print("###############")
print("*** Kmeans ***")
print("###############")
classifier = KMeans().setK(2).setSeed(1)
model=classifier.fit(train)
predictions = model.transform(test)
predictions.groupBy("prediction","label").count().show()
predictions2 = predictions.withColumn("prediction2", predictions["prediction"].cast(DoubleType()))

evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction2", metricName="accuracy")
accuracy = evaluator.evaluate(predictions2)
print("Test Error = %g " % (1.0 - accuracy))
comparaison["kmeans"] = accuracy

print("###############")
print("*** RegressionLogistique ***")
print("###############")
classifier= LogisticRegression()
model=classifier.fit(train)
predictions = model.transform(test)
predictions.groupBy("prediction","label").count().show()
evaluator = MulticlassClassificationEvaluator(
    labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g " % (1.0 - accuracy))
comparaison["regression_logistique"] = accuracy
```

Voici les matrices de confusions pour chacun de nos modèles nous permettant de comparer pour chaque classe le nombre de classe bien prédit et mal prédit :

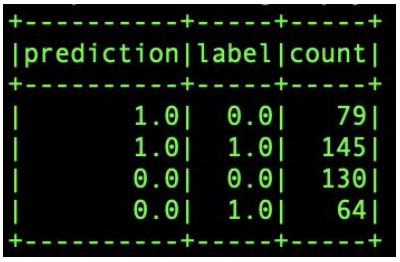
Kmeans (avec K=2)

[7] Git : http://Exercice3_predict_regression_classification.py
[8] https://spark.apache.org/docs/latest/ml-guide.html



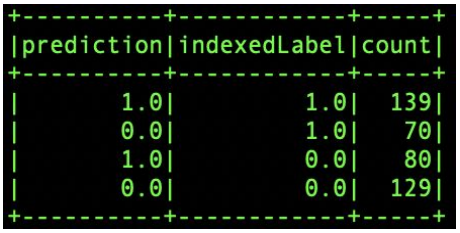
```
+----------+-----+-----+
|prediction|label|count|
+----------+-----+-----+
|         1|  0.0|  110|
|         0|  0.0|   99|
|         1|  1.0|   98|
|         0|  1.0|  111|
+----------+-----+-----+
```

RegressionLogistique



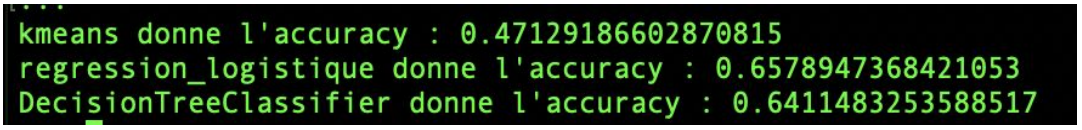
```
+----------+-----+-----+
|prediction|label|count|
+----------+-----+-----+
|       1.0|  0.0|   79|
|       1.0|  1.0|  145|
|       0.0|  0.0|  130|
|       0.0|  1.0|   64|
+----------+-----+-----+
```

DecisionTreeClassifier



```
+----------+------------+-----+
|prediction|indexedLabel|count|
+----------+------------+-----+
|       1.0|         1.0|  139|
|       0.0|         1.0|   70|
|       1.0|         0.0|   80|
|       0.0|         0.0|  129|
+----------+------------+-----+
```

Voici les accuracy pour nos 3 modèles



```
kmeans donne l'accuracy : 0.47129186602870815
regression_logistique donne l'accuracy : 0.6578947368421053
DecisionTreeClassifier donne l'accuracy : 0.6411483253588517
```

L'accuracy mesure la précision d'un modèle de la manière suivante[9] :

$$\text{précision}_i = \frac{\text{nb de documents correctement attribués à la classe } i}{\text{nb de documents attribués à la classe } i}$$

Autrement dit c'est un métrique mesurant le nombre de document correctement prédit pour chaque classe divisé par le nombre de documents. Plus cette valeur est importante plus le modèle est précis dans sa prédiction.

Ici Kmeans ne prédit clairement pas bien. La régression logistique et l'arbre de décision font mieux et sont assez proche. Cependant en analysant les matrices de confusions on se rend bien compte que ces deux modèle n'ont pas réussi à prédire la réussite ou non d'un étudiant de manière efficace étant donnée les informations disponible.

[9] https://fr.wikipedia.org/wiki/Précision_et_rappel

On peut à ce stade se dire que le fait que la réussite à un examen n'est pas déterminé par des questions de genre, de réussite des parents, de ses moyens financiers est plutôt rassurant.

b/ Régression

On essaye ici de prédire[10] la note obtenue par chaque étudiant en mathématique, pour cela on a utilisé 3 modèles(OneVsRest[11], LinearRegression, NaiveBayes) différents qu'on a comparé avec la méthode RMSE :

Le RMSE[12] se calcul de la manière suivante :

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

C'est une règle de notation quadratique qui mesure la moyenne de l'erreur. C'est la racine carrée de la moyenne des différences au carré entre la prévision et l'observation réelle.

Ci-dessous la comparaison des RMSE pour nos 3 modèles :

```
>>> for i in comparaison.keys():
...     print(str(i) + "donne le RMSE :" +str(comparaison[i]))
...
OneVsRestdonne le RMSE :15.472783122110247
LinearRegressiondonne le RMSE :11.251218721543122
NaiveBayesdonne le RMSE :33.09534339428985
>>>
```

c/ Conclusion exercice 3

On remarque qu'il a été relativement difficile de prédire une réussite des élèves avec les éléments en notre possession.
Si la méritocratie est correctement mis en place dans les universités, alors les seuls informations permettant de prédire la réussite d'étudiants aux examens serait leurs motivations et le travail qu'ils ont effectué. Cependant on a pu remarqué qu'il a été possible de dépasser l'accuracy de 0,50 qui serait la prediction aléatoire moyenne de manière significative (0,65) ce qui montre tout de même que si les informations de culture richesse et

[10] Git : http://Exercice3_predict_regression_classification.py
[11] https://spark.apache.org/docs/2.2.0/ml-classification-regression.html
[12] https://en.wikipedia.org/wiki/Root-mean-square_deviation

genres, même s'il ne permettent pas de prédire de manière totale la réussite semble tout de même être des éléments constituant de la réussite chez les étudiants.

Cela semble aller dans le sens de de la thèse défendu par cet article de bloomingyou[13] tout en la modérant car les résultats obtenue bien que significatif ils sont tout de même imprécis.

Cependant on peut se demander pour aller plus loin si utiliser un réseaux de neurones permettrait d'augmenter la précision de nos prédictions.

[13] https://www.bloomingyou.fr/pourquoi-la-meritocratie-nexiste-pas/