

Modélisation Mathématique Machine Learning

Professeur : Rémi Boulle

Date : 25/01/2024

Étudiants : Hugo CASTELL - Egxon ZEJNULLAHI
BUT 3 - 3A

Sommaire

Présentation et objectif

1. Types d'apprentissage supervisé

[Dataset](#)

[Visualisation et model](#)

[Problème](#)

[Applicatif](#)

2. Types d'apprentissage non supervisé

[K-Means Clustering](#)

[Application - K-Means Clustering](#)

[Isolation Forest](#)

[Application - Isolation Forest](#)

3. Types d'apprentissage par renforcement.

[Exploration et exploitation à travers e-greedy \(epsilon-greedy\)](#)

[Coefficient d'apprentissage](#)

[Exemple d'apprentissage: Fonction de valeur SARSA](#)

[Autres exemples de fonctionnement](#)

[Application sur un cas réel \(python\)](#)

[Contexte](#)

[Mise en place](#)

[Résultats](#)

[Conclusion](#)

4. Régression linéaire

[Méthode des moindres carrés](#)

[Qualité de la régression](#)

[Fiabilité des prévisions](#)

[Application sur un cas réel \(python\)](#)

[Contexte](#)

[Raisonnement](#)

[Mise en place](#)

[Résultats](#)

[Conclusion](#)

5. Sources

6. Modules Python utilisés

Présentation et objectif

Ce document est une présentation des différents types d'apprentissage, supervisé, non supervisé et par renforcement, et les algorithmes associés dans le domaine du machine learning.

Ce document va aussi présenter la régression linéaire avec un exemple précis et contient des extraits de nos réalisations dont le code source est disponible dans ce [dépôt git](#).

Machine Learning : Donner à une machine la capacité d'apprendre sans la programmer de façon explicite.

1. Types d'apprentissage supervisé

Dataset

L'objectif de cette méthode est de montrer à la machine des exemples X et Y et lui demander de trouver l'association qui relie X à Y telle que $Y = F(X)$.

Ces exemples sont dans un dataset (Un jeu de données d'apprentissage entraîne un modèle de machine learning à effectuer une tâche ou à réaliser une prédiction). Dans les dataset en apprentissage supervisé, nous avons deux variables importantes.

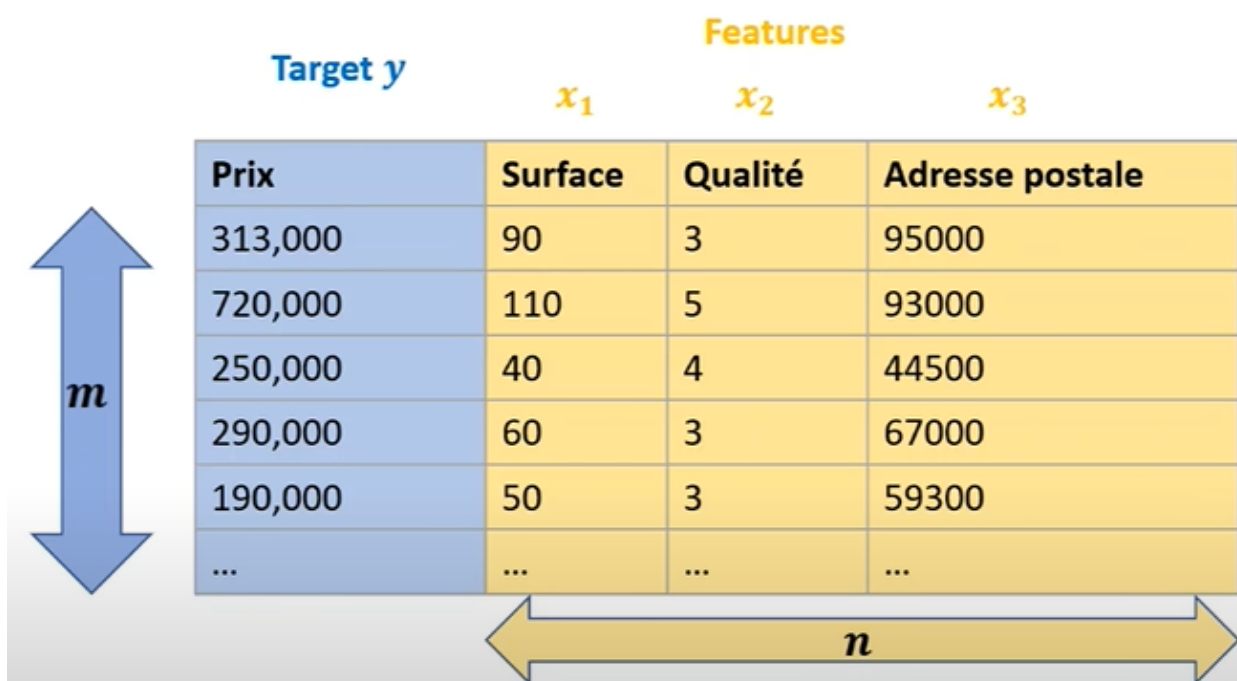
On va avoir la **Target Variable (y)** qui est donc la variable d'objectif, la variable que notre machine doit apprendre à prédire. Ça peut être par exemple : le prix d'un appartement, la courbe de la bourse, identifier si un mail est un spam ou non, etc. . .

Ensuite, il y a les **Features (x)**, qui sont donc les facteurs, ces variables vont influencer la valeur de Y (Target Variable). Y est une fonction de toutes les features.

Voici les notions par conventions pour les datasets :

m : nombre d'exemples

n : nombre de features



The diagram illustrates a dataset table. A vertical blue double-headed arrow on the left is labeled 'm', representing the number of examples (rows). A horizontal yellow double-headed arrow at the bottom is labeled 'n', representing the number of features (columns). The table itself has a blue header row for the target variable and yellow data rows for features.

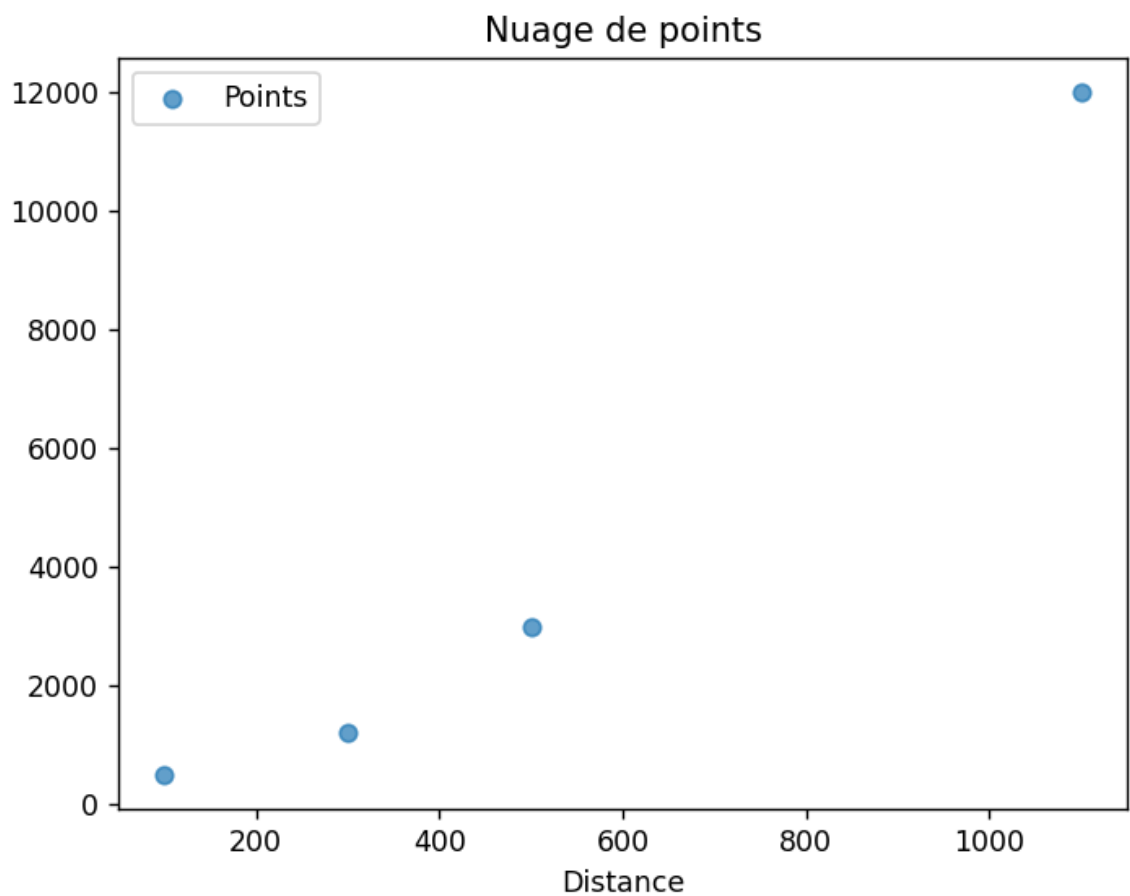
Target y	Features		
	x_1	x_2	x_3
Prix	Surface	Qualité	Adresse postale
313,000	90	3	95000
720,000	110	5	93000
250,000	40	4	44500
290,000	60	3	67000
190,000	50	3	59300
...

Une dataset est donc un Vecteur ***Target Variable*** de y élément et une Matrice **Feature** x $n * m$.

À partir du DataSet, on peut visualiser les données de différentes manières, par exemple avec un nuage de point.

Visualisation et model

y	x_1	x_2	x_3	...	x_n
$y^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$...	$x_n^{(1)}$
$y^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$...	$x_n^{(2)}$
$y^{(3)}$	$x_1^{(3)}$	$x_2^{(3)}$	$x_3^{(3)}$...	$x_n^{(3)}$
...
$y^{(m)}$	$x_1^{(m)}$	$x_2^{(m)}$	$x_3^{(m)}$...	$x_n^{(m)}$



Ensuite, on doit spécifier à la machine quel modèle il doit apprendre selon la visualisation, c'est-à-dire choisir le modèle le plus adapté.

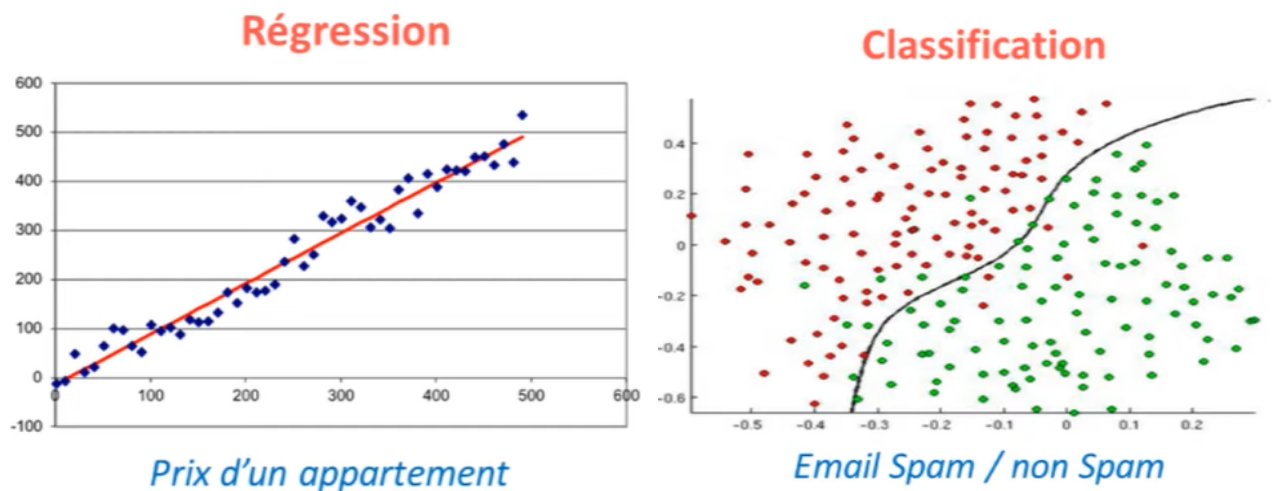
Il peut y avoir plusieurs choix :

- modèle linéaire
- modèle polynomial
- modèle de neurone
- Arbre de décision . . .

Une fois que le modèle de données est choisi, la machine va pouvoir entrer dans ce que l'on appelle la phase d'entraînement. Grâce à un algorithme d'optimisation, la machine va trouver les paramètres qui fournissent les meilleures performances sur la dataset, pour pouvoir donc prédire les potentiels futurs Target Variable les features (les facteurs).

Problème

Le méthode d'apprentissage supervisé nous permet de résoudre deux problèmes, de régression et de classification



Applicatif

Grâce au module sklearn sur python, on est capable de faire du machine learning facilement. Scikit-learn est une bibliothèque libre Python destinée à l'apprentissage automatique.

Voici les 4 étapes fondamentales pour mettre en application l'apprentissage supervisé sur python.

```
model = LinearRegression()  
model.fit(X,y)  
model.score(X,y)  
model.predict(X)
```

Plein de model et d'algorithmes ont déjà été implémentés par sklearn sur python et chacun de ces modèles dispose de leur propre classe orientée en objet :

1. Supervised learning

- ▶ 1.1. Generalized Linear Models
- ▶ 1.2. Linear and Quadratic Discriminant Analysis
- 1.3. Kernel ridge regression
- ▶ 1.4. Support Vector Machines
- ▶ 1.5. Stochastic Gradient Descent
- ▶ 1.6. Nearest Neighbors
- ▶ 1.7. Gaussian Processes
- 1.8. Cross decomposition
- ▶ 1.9. Naive Bayes
- ▶ 1.10. Decision Trees
- ▶ 1.11. Ensemble methods
- ▶ 1.12. Multiclass and multilabel algorithms
- ▶ 1.13. Feature selection

Dans un premier temps, on va devoir déclarer une variable qui va correspondre à un de ces modèles. Voici un exemple qui montre l'initialisation de notre variable selon les modèles disponibles

```
model = LinearRegression()  
model = SGDRegressor()  
model = RandomForestClassifier()  
model = SVR()
```

Une fois que notre modèle est choisi, on rentre dans la phase d'entraînement on va évaluer les paramètres avec la fonction fit qui a comme argument deux tableaux Numpy pour représenter les Variables de notre dataset (les features, X, et les targets variables ,y)

```
model.fit(X,y)
```

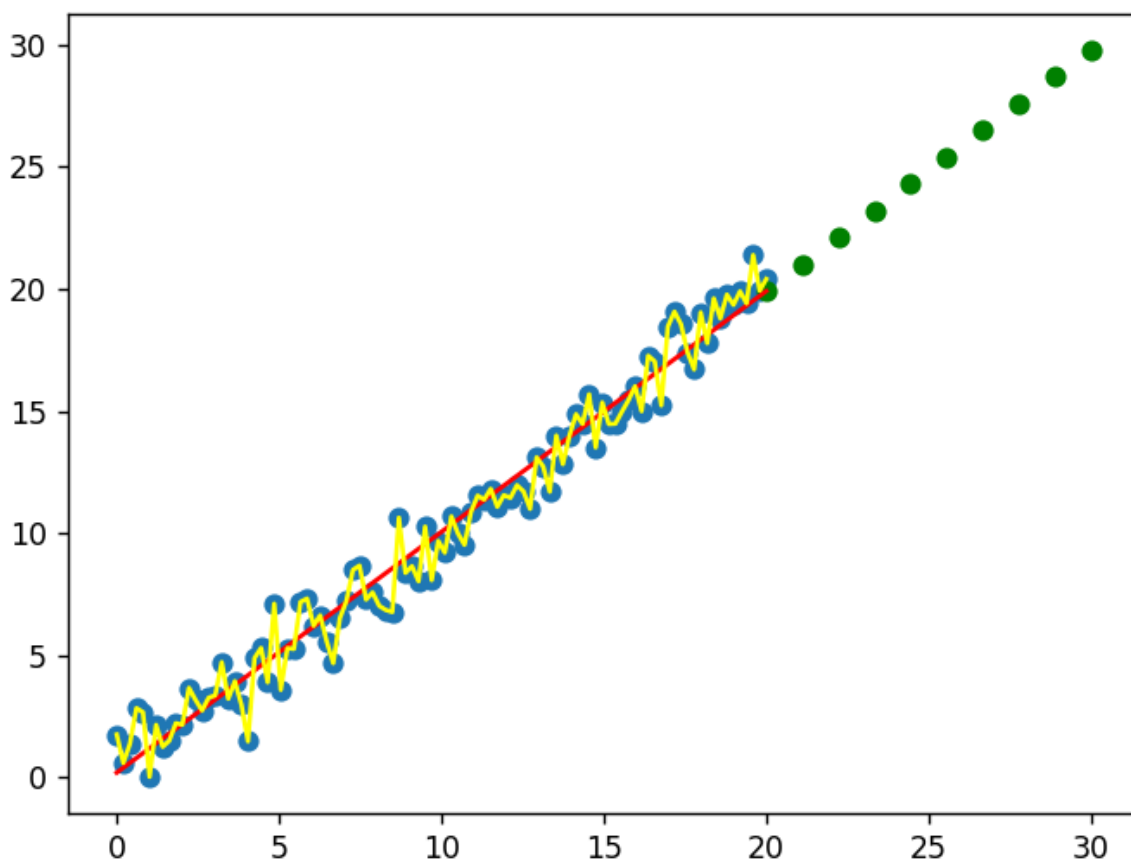
De plus, on peut utiliser la fonction score qui nous permet d'évaluer la performance de notre modèle. Cette méthode va de nouveau utiliser les variables X et y en utilisant les données X pour faire des prédictions et les comparer aux valeurs y. C'est le calcul de coefficient de détermination.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

```
model.score(X,y)
```

Et enfin, on va pouvoir utiliser la fonction prédictive qui va nous permettre d'utiliser notre méthode pour de nouvelles valeurs.

Voici un exemple d'application. Sur ce graphique, nous avons les points bleus qui représentent les données X et Y de notre dataset et nous avons la courbe de tendances en jaune. Mais le plus important, c'est le trait rouge, qui représente la prédiction des données avec le modèle d'entraînement avec la méthode prédictive, et on peut voir que le résultat est satisfaisant, car la régression linéaire est bien superposée aux données réelles. Et enfin, les points verts sont aussi une prédiction des target variables avec de nouvelles données X, plus grande que les données de base, et encore une fois là-dessus, on peut voir une continuité de la régression linéaires.



Lien pour tester le code :

https://github.com/Hugo-CASTELL/modelisation-mathematiques/blob/main/types_app/rentissage/supervis%C3%A9/supervise_linerTest.py

2. Types d'apprentissage non supervisé

Définition **d'apprentissage non supervisé** : La machine analyse la structure des données X pour apprendre elle-même à réaliser certaines tâches.

L'objectif principal de l'apprentissage non supervisé est de permettre à un modèle d'extraire des informations significatives, des structures ou des modèles à partir de données non étiquetées, contrairement à l'apprentissage supervisé.

L'apprentissage non supervisé consiste donc à permettre au modèle d'explorer et de découvrir des structures ou des modèles intrinsèques au sein des données sans aucune orientation préalable.

En d'autres termes, le modèle est laissé à lui-même pour identifier des relations, on va donner à la machine que des données X et ça sera le travail de la machine de reconnaître la structure de ces données pour apprendre elle-même à réaliser certaines tâches.

La machine peut apprendre à classer ces données en les regroupant selon leurs ressemblances, faire donc du clustering (de la classification non supervisée). Avec cette méthode on peut réaliser énormément de tâches comme :

- Classer des documents
- Classer des photos
- Classer des tweets/des posts
- Segmenter la clientèle d'une entreprise pour le marketing.

L'algorithme associé pour cette application est l'algorithme de K-Means Clustering

L'apprentissage non supervisé nous permet aussi de faire de la détection d'anomalie. C'est-à-dire qu'on va pouvoir trouver des échantillons dans nos données avec des caractéristiques qui sont très éloignées des autres échantillons. Cette méthode nous permet de développer :

- des systèmes de sécurité

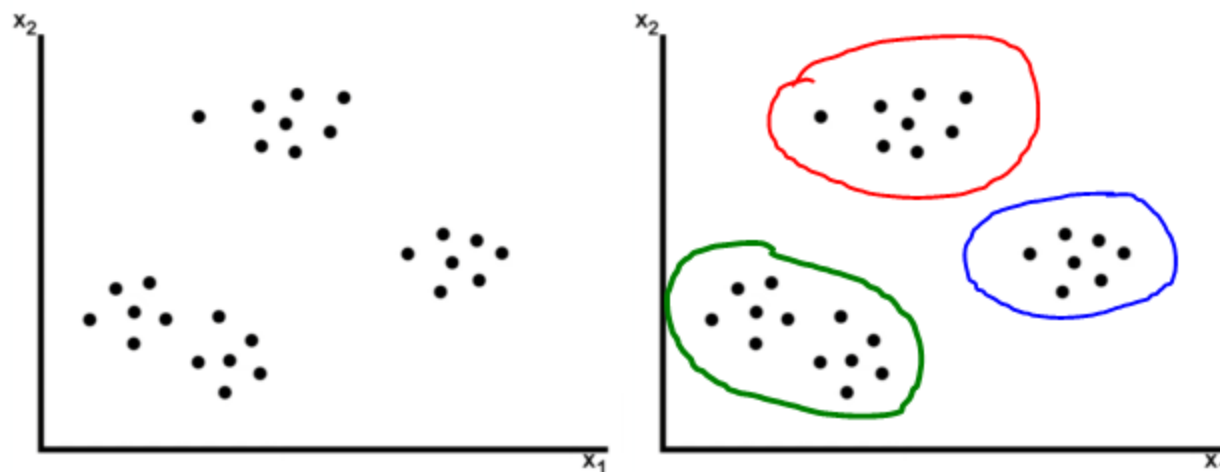
- des détections de fraude bancaire (du hacking en général)
- des détections de défaillances.

L'algorithme associé pour cette application est l'algorithme d'Isolation Forest

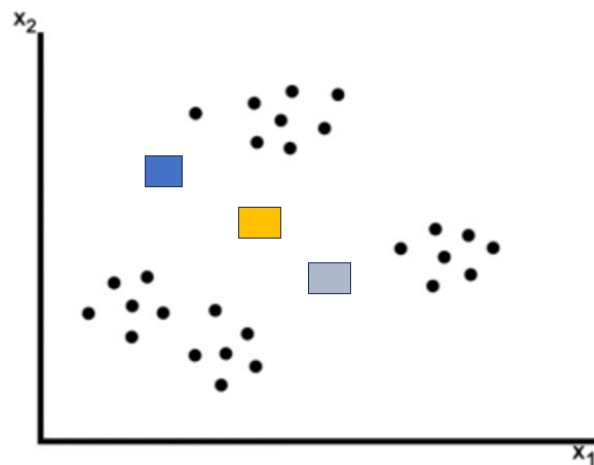
K-Means Clustering

Le but de l'algorithme K-Means Clustering va être de laisser la machine classer les données selon leurs ressemblances. Voici un exemple pour comprendre comment marche cet algorithme.

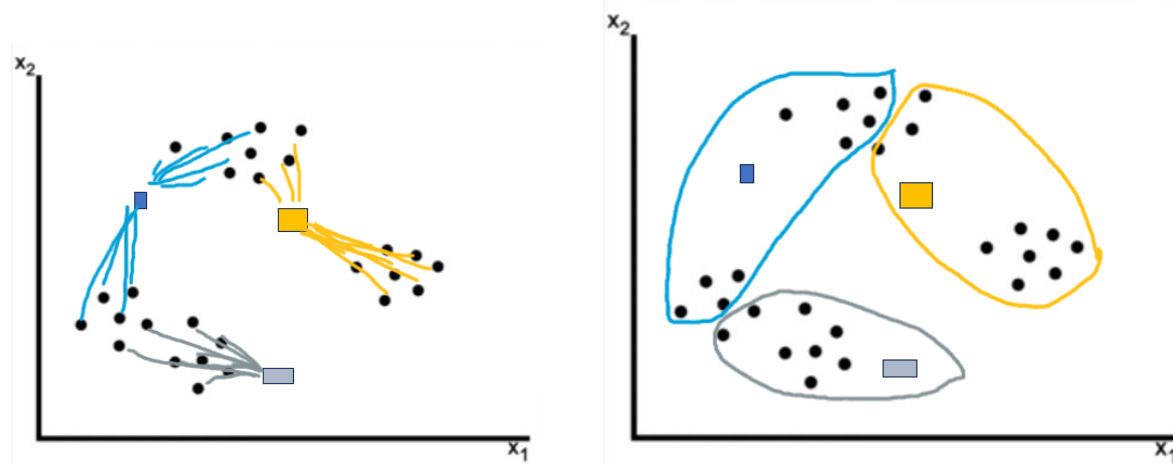
Voici un jeu de données simple avec des données sur un graphique cartésien. L'objectif va être de regrouper des groupes en 3 clusters .



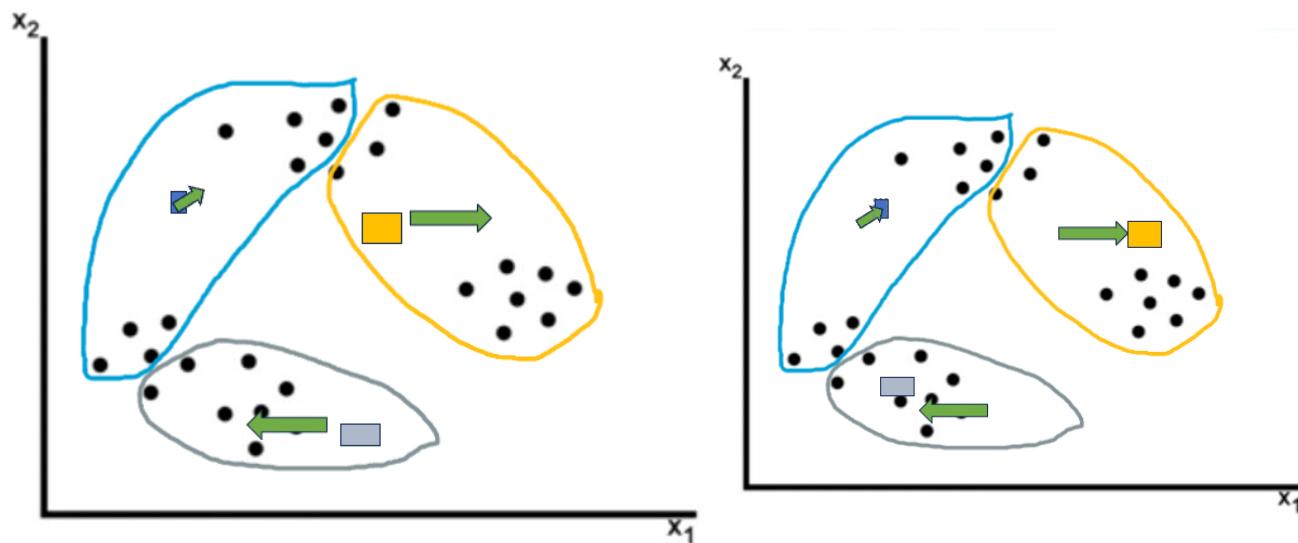
Pour réaliser cela, il faut placer des points qu'on nomme "Centroid" sur le graphique au hasard



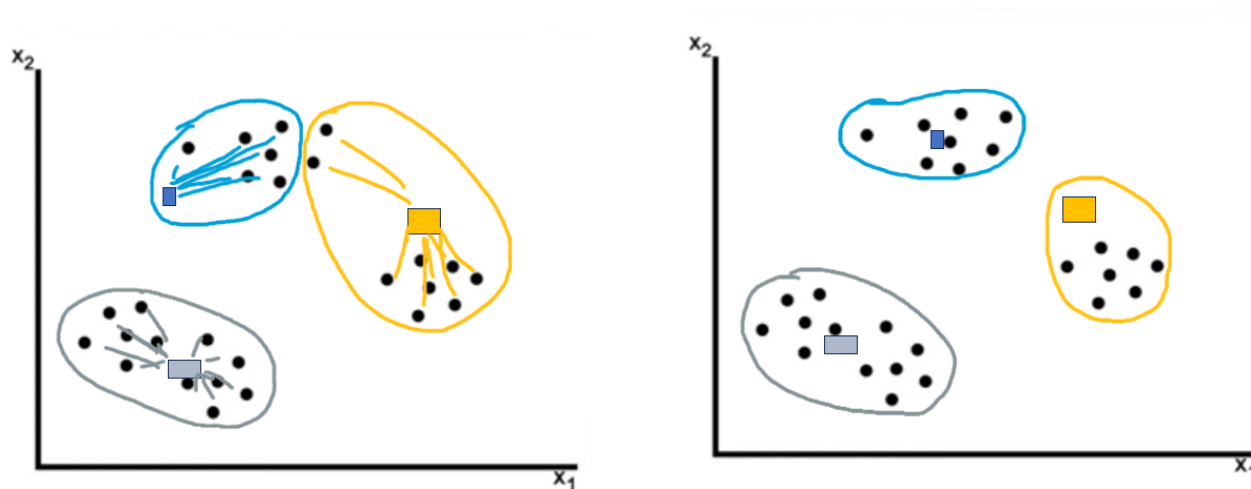
Ensuite, il faut relier ces centroid aux points les plus proches de notre Dataset, ce qui va créer nos 1er clusters.



Puis, il faut déplacer chaque centroid au milieu de son cluster (la ou se situe la moyenne des points)



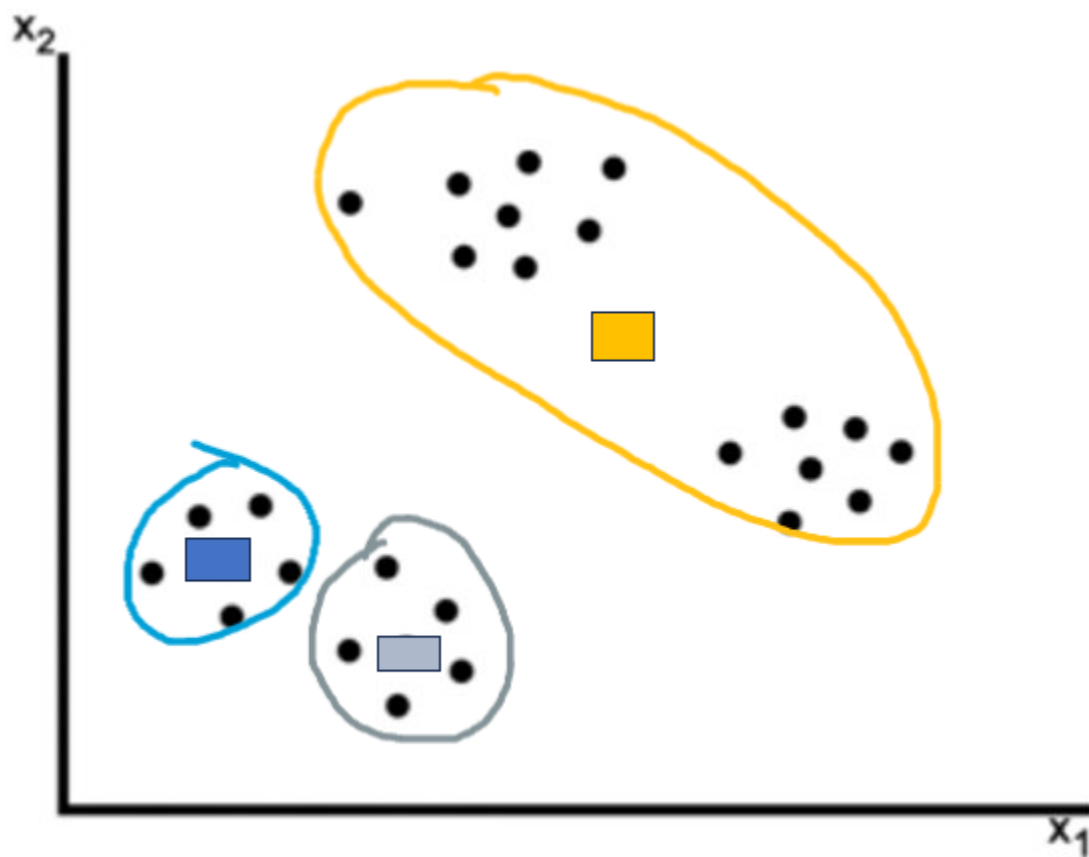
Et donc , il faut recommencer à affecter chaque point de notre dataset au centroid le plus proche. Il faut continuer ainsi, jusqu'à que les centroid converge vers une position d'équilibre.



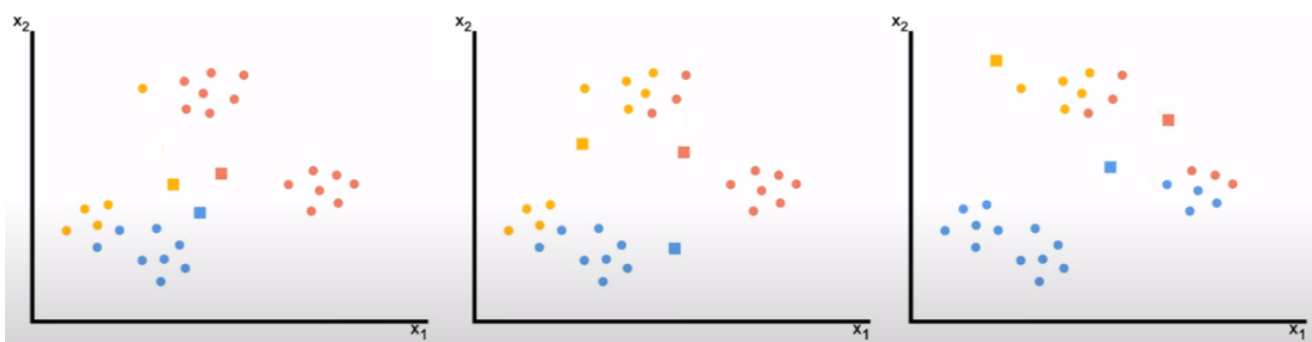
L'algorithme de K-Means Clustering est donc un algorithme itératif qui fonctionne en 2 étapes et qu'il faut répéter jusqu'à que les centroid converge vers une position d'équilibre .

1. Affection des centroids points au centre le plus proche dans le dataset
2. Déplacement du centre à la moyenne du cluster.

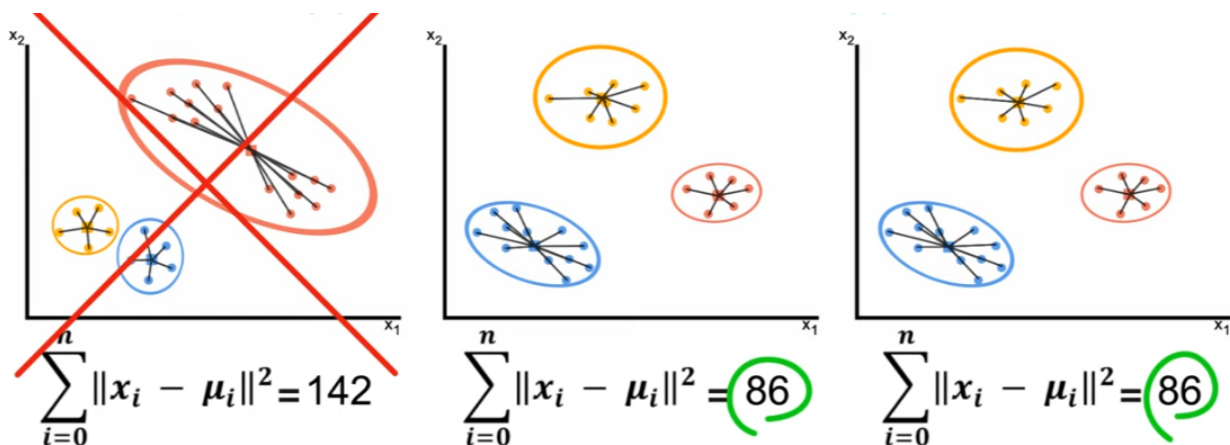
ATTENTION : SELON LA POSITION DES CENTROIDS, K-MEANS PEUT DONNER DE MAUVAIS CLUSTER



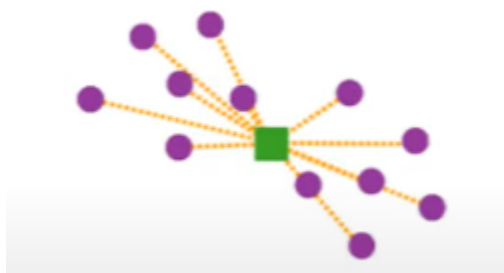
Pour éviter ce problème, on exécute l'algorithme K-Mean avec différente position de départ. La solution retenue est celle qui minimise la somme des distances entre les points (x) et son centre.



Pour chaque résultat donné, on mesure la distance entre chaque



En résumé : K-Mean cherche la position des centres qui minimise la distance entre les points d'un cluster et son centre



$$\sum_{i=0}^n \min_{\mu_j \in C} (\|x_i - \mu_j\|^2)$$

Cela équivaut donc à minimiser la variance des clusters.

Application - K-Means Clustering

Voici un exemple pour tester cette méthode sur python avec le module Scikit learn et matplotlib.

Dans un premier temps, il faut obtenir un jeu de données. Grâce à scikit learn on peut utiliser la méthode `make_blobs`. Cette méthode nous retourne une matrice X qui représente les échantillons sur le graphique cartésien, et une liste Y qui représente l'appartenance au cluster de chaque échantillon.

Description des paramètres:

`n_samples` : Nombre total de points généré

`centers` : Nombre de total de cluster

`cluster_std` : Le standard de déviation des clusters

`random_state` : Détermine la génération de nombres aléatoires pour la création d'un ensemble de données.

```
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=50, centers=4, cluster_std=0.2, random_state=0)
plt.scatter(X[:, 0], X[:, 1])
plt.show()
```

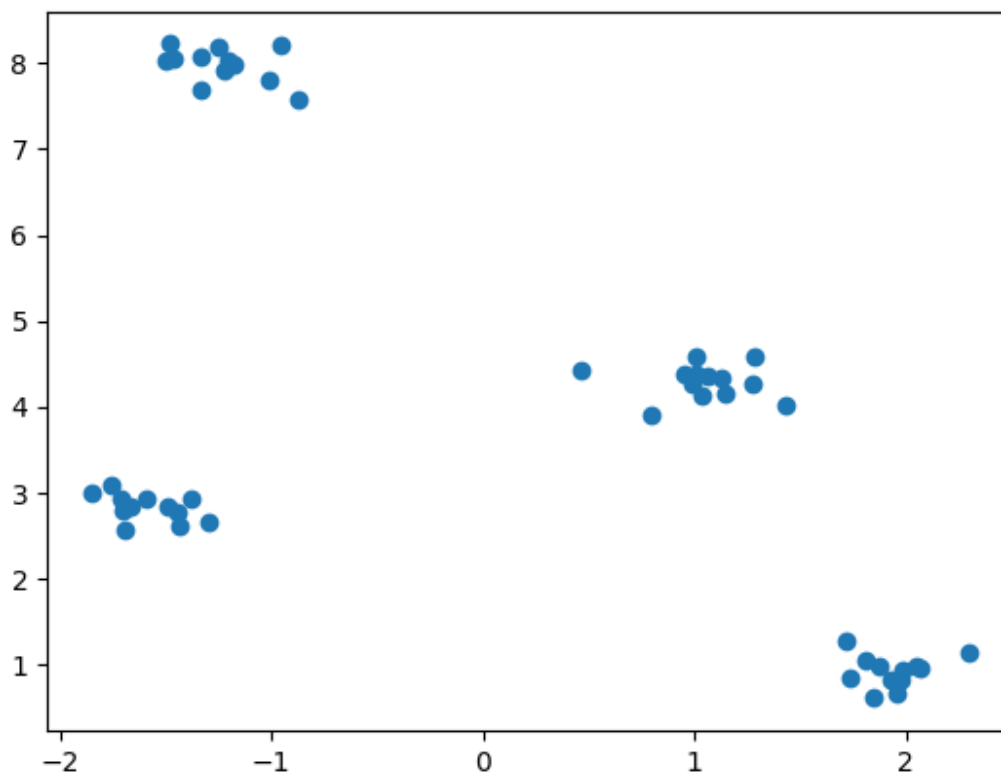
Valeur X :

```
[[ 0.95562631  4.38590703]
 [-1.6613961   2.84597163]
 [ 1.27508589  4.26275567]
 [ 0.46567212  4.43451105]
 [-1.1769825   7.97677465]
 [ 1.98568509  0.92893345]
```

Valeur y :

```
[0 2 0 0 3 1 3 3 2 2 3 1 1 2 1 0 0 3 3 1 2 1 1 0 3 1 2 2 2 0 1 0 0 2 1 0 1
 2 3 0 3 1 0 3 1 3 2 0 2 3]
```

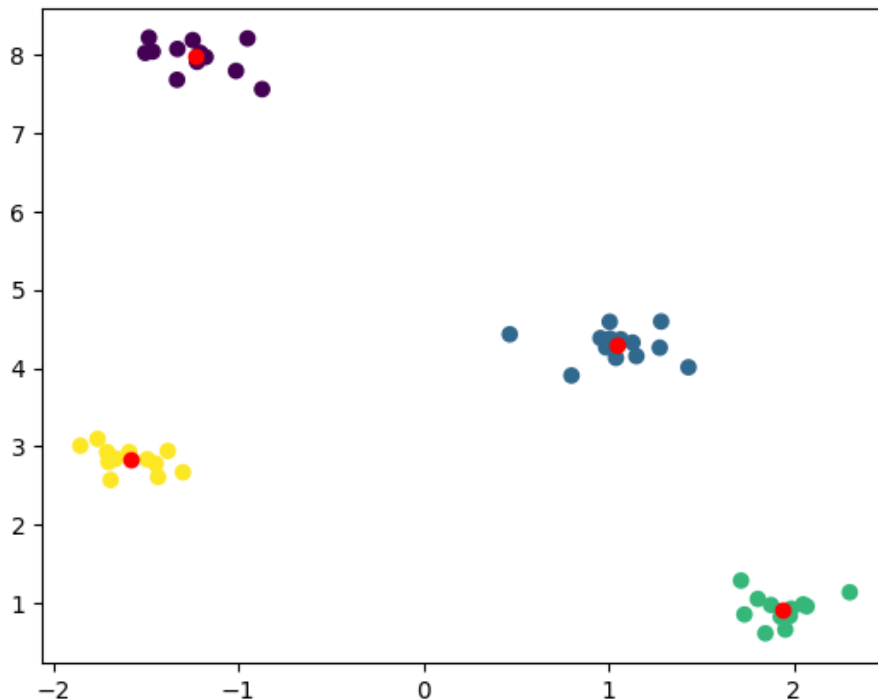
Graphique généré avec matplotlib :



Ensuite, il faut déclarer le modèle KMeans avec le paramètre ***n_cluster*** qui équivaut à 4 pour que ça soit logique avec le nombre de cluster créer par le parametre centers de la méthode make_blobs.

Une fois que le modèle est entraîné avec la méthode fit, on peut avec matplotlib afficher nos cluster en couleur, surtout, affiché nos centroids sur le graphique, grâce à l'attribut cluster centers. (le paramètre c désigne la couleur, dans l'exemple r est égal a "red" donc les centroid sont rouges)

```
model = KMeans(n_clusters=4)
model.fit(X)
plt.scatter(X[:,0], X[:,1], c=model.predict(X))
plt.scatter(model.cluster_centers_[0,0], model.cluster_centers_[0,1], c='r')
```



Lien pour tester le code :

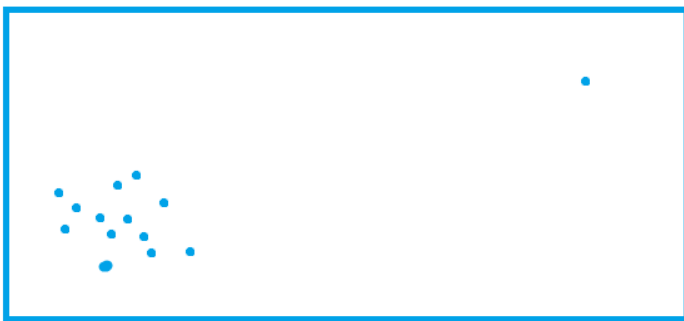
https://github.com/Hugo-CASTELL/modelisation-mathematiques/blob/main/types_app/rentissage/non_sup%C3%A9vis%C3%A9/NonSupervise_KMeans.py

Isolation Forest

Comme cela a été dit précédemment, l'isolation forest est un algorithme qui nous permet de faire de la détection d'anomalie.

Pour cette méthode, on effectue une série de split (de découpe) aléatoire sur notre graphique et on compte le nombre de splits qu'il faut effectuer pour pouvoir isoler nos échantillons.

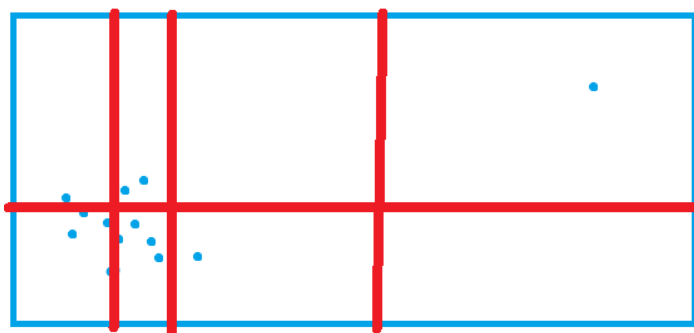
Nombre de split = 0



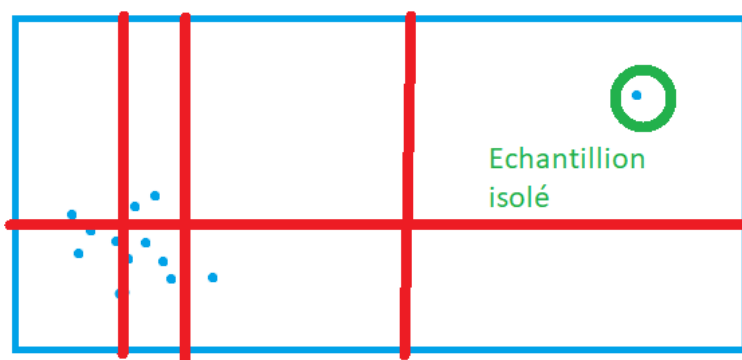
Nombre de split = 1



Nombre de split = 4



Nombre de split = 4

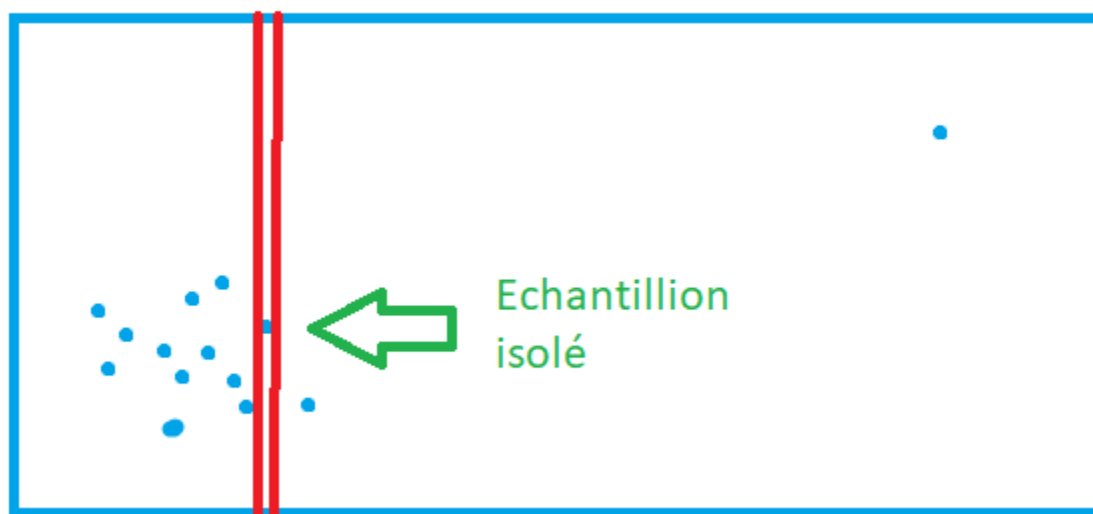


Plus le nombre de découpe est petit , plus il y a de chances que cet échantillon soit une anomalie.

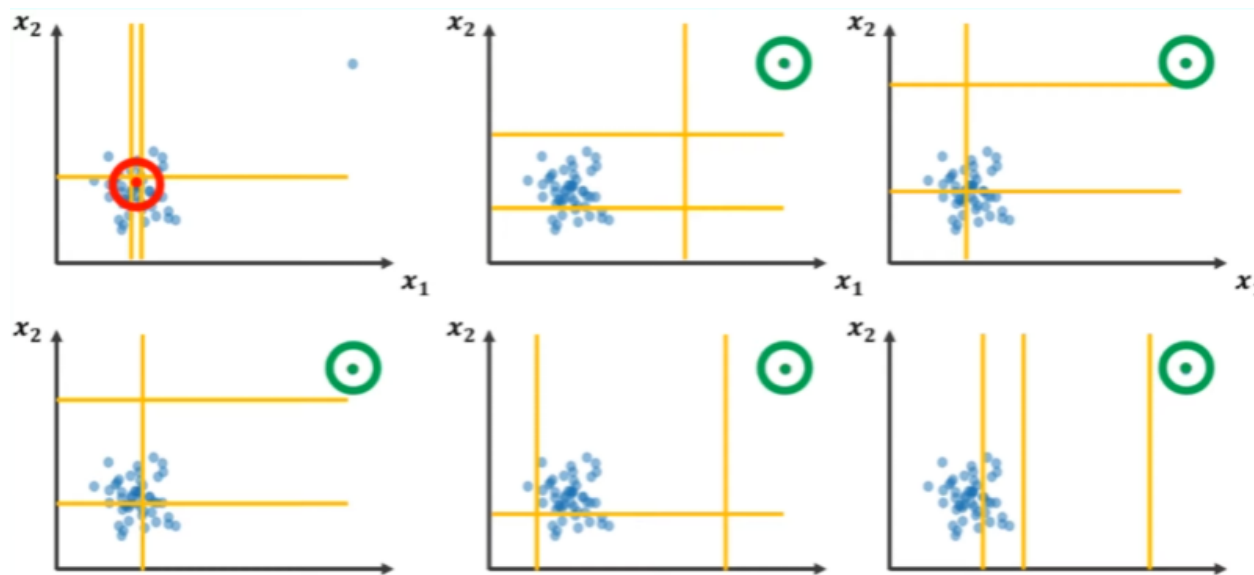
Donc : Faible nombre de **split** → **Anomalie**

Néanmoins, cette méthode est sûre de marcher à tous les coups. Vu que les split sont aléatoires, ce n'est pas impossible qu'on puisse isoler un échantillon dans la masse, malgré que la probabilité est très faible. Donc, même si la probabilité est faible, il est quand même possible d'isoler un échantillon dans la masse avant d'isoler une vraie anomalie.

Nombre de split = 2



Pour résoudre ce problème, il faut tout simplement réaliser une technique d'ensemble. Il faut entraîner plusieurs estimateurs pour ensuite considérer l'ensemble de leurs estimations. Le résultat final correspond à l'ensemble (la moyenne) des résultats. Donc il faudrait relancer plusieurs fois notre algorithme et garder comme résultat le point a le plus était détecté.

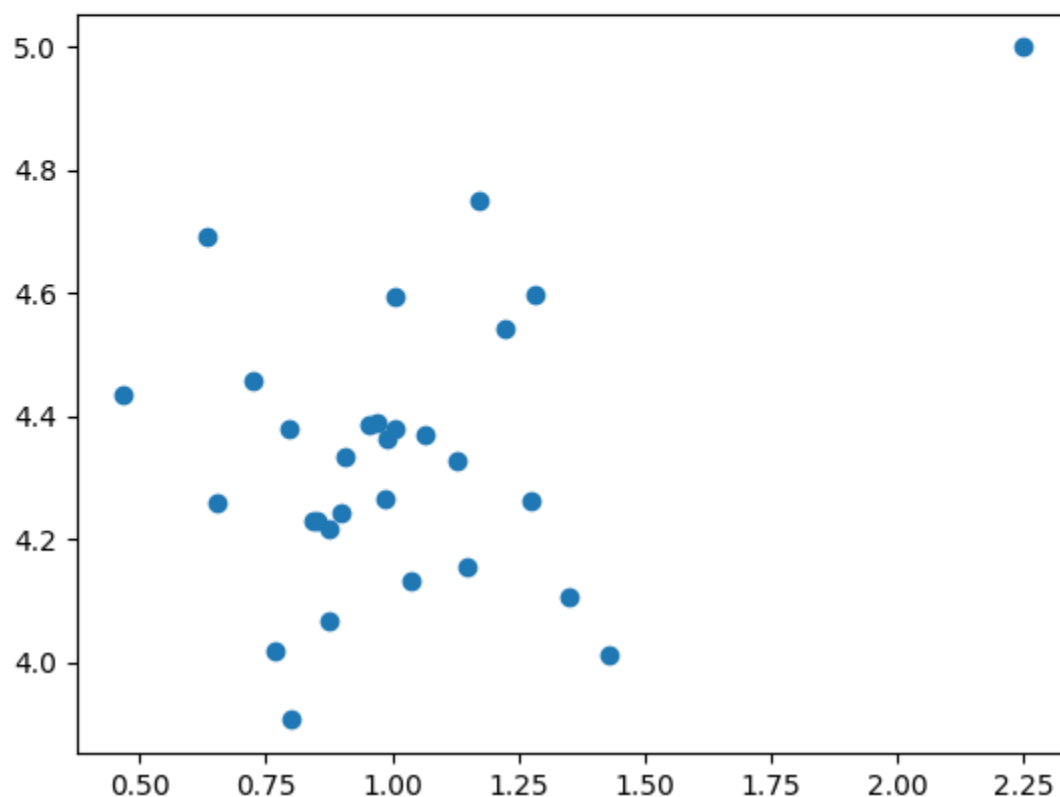


Application - Isolation Forest

Voici notre jeu de données créé encore une fois grâce à la méthode `make_blobs`.
Il faut ensuite ajouter une donnée assez éloigné de notre cluster.

```
X, y = make_blobs(n_samples=30, centers=1, cluster_std=0.2, random_state=0)
X[-1,:] = np.array([2.25, 5])
plt.scatter(X[:,0], X[:, 1])
plt.show()
```

Figure 1



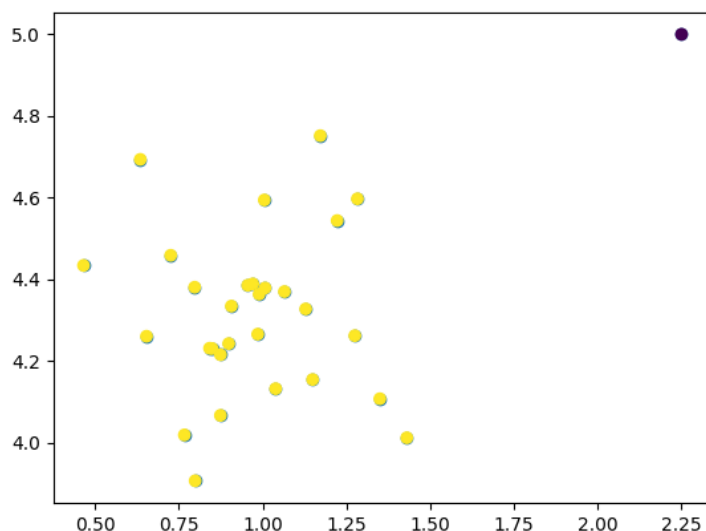
Une fois le jeu de données bien défini, il faut utiliser le model IsolationForest en lui donnant en paramètre le taux de contamination qu'on pense y avoir. Ici, on initialise la variable contamination à 0,01, ce qui équivaut à 1%. Cela veut donc dire qu'il y'a 1% de déchet que la machine doit trouver dans notre DataSet

Ensuite après avoir entraîné le model, il est nécessaire de re-afficher notre graphique mais en changeant les couleurs pour voir l'échantillon qui se démarque des autres. `model.predict(X)` retourne une liste de chiffres reliée à l'échantillon au niveau de l'index. 1 Signifie que l'échantillon n'ai pas une anomalie et -1 signifie que l'échantillon est une anomalie.

```
model = IsolationForest(contamination=0.01)
model.fit(X)
plt.scatter(X[:,0], X[:, 1], c=model.predict(X))
print(model.predict(X))
plt.show()
```

```
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1 -1]
```

Voici le résultat, on voit bien que la donnée en haut à droite de notre graphique est d'une autre couleur que le reste des données.

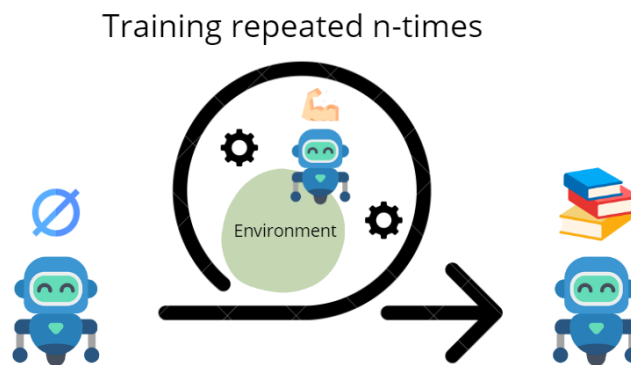


Lien pour tester le code → https://github.com/Hugo-CASTELL/modelisation-mathematiques/blob/main/types_apprentissage/non_sup%C3%A9vis%C3%A9/NonSupervise_IsolationForest.py

3. Types d'apprentissage par renforcement.

L'objectif de l'apprentissage par renforcement est d'apprendre à un agent, un modèle, à se comporter dans un environnement où les données d'entraînement proviennent de l'environnement lui-même, sous forme simulée ou réelle. On citera donc des exemples d'IA utilisées dans des domaines variés : systèmes de recommandation (Netflix, YouTube; publicité ciblée), AlphaGo (Go; jeux de sociétés), Nexto (Rocket League; jeux-vidéos) et DALL-E (Génération d'images; art)...

À la manière d'un enfant apprenant à se comporter dans le monde réel, l'agent, petit à petit, forme son dataset après de multiples essais en tentant toujours de minimiser les coûts et maximiser les récompenses.



L'agent décide de faire une action, l'environnement en déduit et lui renvoie un état et une récompense pour que l'agent puisse faire une nouvelle action.

Exploration et exploitation à travers e-greedy (epsilon-greedy)

L'apprentissage par renforcement aborde deux principes, l'exploration et l'exploitation.

L'exploration consiste à explorer les possibilités, aléatoirement, pour tenter de trouver des solutions qui apportent plus de récompenses tandis que l'exploitation est le fait d'utiliser les connaissances emmagasinées pour faire les choix les plus rentables.

Une totale exploration n'amène à rien de rentable de manière constante et peut conduire à consommer inefficacement du temps et des ressources tout comme une totale exploitation peut limiter le gain possible de récompenses en empêchant la potentielle découverte de meilleures stratégies.

Dans le but d'optimiser les avantages de chaque paradigme, il faut trouver un équilibre entre une majorité d'exploration et une majorité d'exploitation.

L'algorithme qui le modélise se nomme e-greedy (epsilon-greedy) : epsilon pour représenter le taux d'exploration et greedy le taux d'exploitation à un instant T , soit $e = x$ et $greedy = 1 - e$.

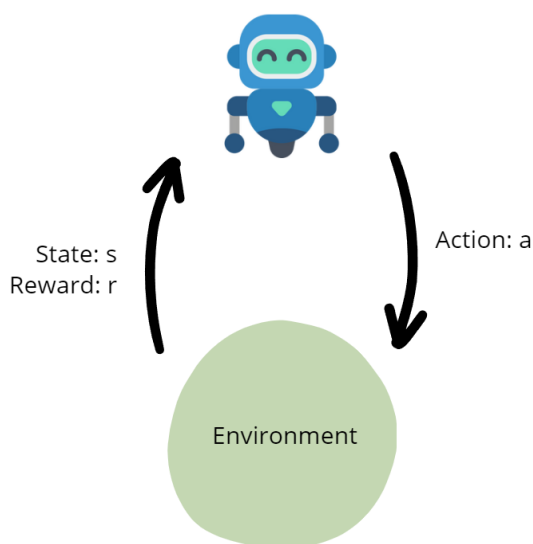
Coefficient d'apprentissage

Souvent noté lr pour learning rate, le coefficient d'apprentissage est un facteur important de l'entraînement d'une IA par renforcement. Son rôle principal est de contrôler la taille des ajustements effectués lors de la mise à jour des valeurs dans le processus d'entraînement.

Le choix du lr est adapté au problème et il n'existe pas encore de méthode précise pour le calculer. Il faudra tester les valeurs de manière empirique et comparer les résultats.

Exemple d'apprentissage: Fonction de valeur SARSA

Lorsque l'agent effectue une action, il reçoit alors un état et une récompense associé et l'on revient au moment du choix de l'action. On appelle cette boucle une transition.



À chaque transition, l'agent s'appuie sur une fonction pour choisir sa prochaine action. Le but de cette fonction est d'estimer l'espérance du nombre de récompenses lors d'une transition de l'état présent à l'état futur. Voici comment elle se présente :

$$V(s) = V(s) + \text{learning_rate} * (\text{reward} + V(s') - V(s))$$

Soit la valeur à l'état présent est égale à la somme entre elle-même et les récompenses espérées déduites de la valeur de l'état futur moins la valeur de l'état présent multiplié par un coefficient d'apprentissage. Plus simplement, en une fonction qui permet de calculer la croissance de $V(s)$ par rapport à ce que l'agent connaît de l'action future.

Si on décortique la fonction, on remarque que l'espérance $\text{learning_rate} * (V(s') - V(s))$ dépend d'un état déjà calculé, or si l'état futur n'a jamais été rencontré, l'état présent calculé est égal à la récompense obtenue avant ajustement.

SARSA est l'acronyme de State-Action-Reward-State-Action, ainsi on remarque :

- State-Action (présent) : état présent | $V(s)$
- Reward : récompense à l'état présent | reward
- State-Action (futur) : état futur | $V(s')$

Autres exemples de fonctionnement

Pour faire comprendre à un agent l'objectif dans l'environnement dans lequel il évolue, il existe un certain nombre de variantes, en voici une liste non exhaustive :

- Apprentissage par renforcement basé sur la valeur¹ : Q-learning, SARSA²
- Apprentissage par renforcement basé sur la politique³ : Policy Gradient, Trust Region.
- Apprentissage par renforcement basé sur l'avantage⁴ : Actor-Critic
- Apprentissage par renforcement basé sur la fonction de récompense⁵: Imitation Learning (utilisé pour simuler les actions de quelqu'un)

Application sur un cas réel (python)

Contexte

Pour mieux comprendre son fonctionnement, nous nous sommes lancés dans le développement from scratch d'une IA.

Après plusieurs réflexions, nous sommes tombés sur l'idée de l'IA qui se déplace dans un graphe et qui cherche à maximiser les récompenses qu'elle y trouve.

Mise en place

Nous y avons implémenté un mécanisme epsilon-greedy.

¹ Apprentissage par renforcement basé sur la valeur: Value-Based Reinforcement Learning

² SARSA: State-Action-Reward-State-Action

³ Apprentissage par renforcement basé sur la politique: Policy-Based Reinforcement Learning

⁴ Apprentissage par renforcement basé sur l'avantage: Advantage-Based Reinforcement Learning

⁵ Apprentissage par renforcement basé sur la fonction de récompense: Reward Function-Based Reinforcement Learning :


```

1 usage  Hugo-CASTELL
def play(self, state, next_possible_nodes: list[Node]):
    # Initializing state value
    if state.numero not in self.V:
        self.V[state.numero] = 0.0

    # Algorithm epsilon-greedy
    if random.uniform(a: 0, b: 1) < self.epsilon:
        return self.explore(next_possible_nodes)
    else:
        return self.greedy(next_possible_nodes)

```

Et dans un second temps, nous avons mis en place la méthode SARSA.

```

1 usage  Hugo-CASTELL
def train(self):
    for transition in reversed(self.history):
        # Transition values
        node_numero, next_node_numero, reward = transition

        # Updating V
        self.V[node_numero] = self.V[node_numero] + self.learning_rate * (reward + self.V[next_node_numero] - self.V[node_numero])

    # Clearing history
    self.history.clean()

```

Et nous avons pour objectif de choisir le nœud qui lui apportera le plus de récompenses.

```

1 usage  Hugo-CASTELL
def explore(self, next_possible_nodes: list[Node]):
    # Choosing a random node
    rand_next_node_index = random.randint(a: 0, len(next_possible_nodes) - 1)
    return next_possible_nodes[rand_next_node_index]

1 usage  Hugo-CASTELL
def greedy(self, next_possible_nodes: list[Node]):
    # Choosing the best node
    best_node = None
    for node in next_possible_nodes:
        if best_node is None or self.V[best_node.numero] < self.V[node.numero]:
            best_node = node
    return best_node

```

Ensuite, nous avons programmé le jeu :

```

2 usages  Hugo-CASTELL
def play(self, ia: IA):
    ia.route_taken.clear()
    state = self.start_node
    while state != self.end_node:
        ia.route_taken.append(state.numero)
        # Environment returns to the agent the possible next nodes and the reward
        next_possible_nodes, reward = self.step(state)
        # IA plays
        next_node = ia.play(state, next_possible_nodes)
        # IA updates its history depending on the reward
        ia.update_history((state.numero, next_node.numero, reward))
        # Passing action to the environment
        state = next_node
    # Saving final state
    ia.V[self.end_node.numero] = 0.0
    ia.route_taken.append(self.end_node.numero)
    ia.update_history((self.end_node.numero, self.start_node.numero, self.end_node.reward))

```

Finalement, voici notre main qui entraîne l'IA pour la fait jouer juste avec ses connaissances (epsilon à 0) pour voir ce qu'elle a appris :

```

ia = IA()
game = Game(routes, start_node, end_node)

# Training
trainings = 1000000
for i in range(trainings):
    game.play(ia)
    ia.train()
    ia.epsilon = max((trainings - i) / trainings, 0.05)

# Testing with only greedy knowledge
ia.epsilon = 0
print("--- Value function:")
print(ia.V)
print("\n")

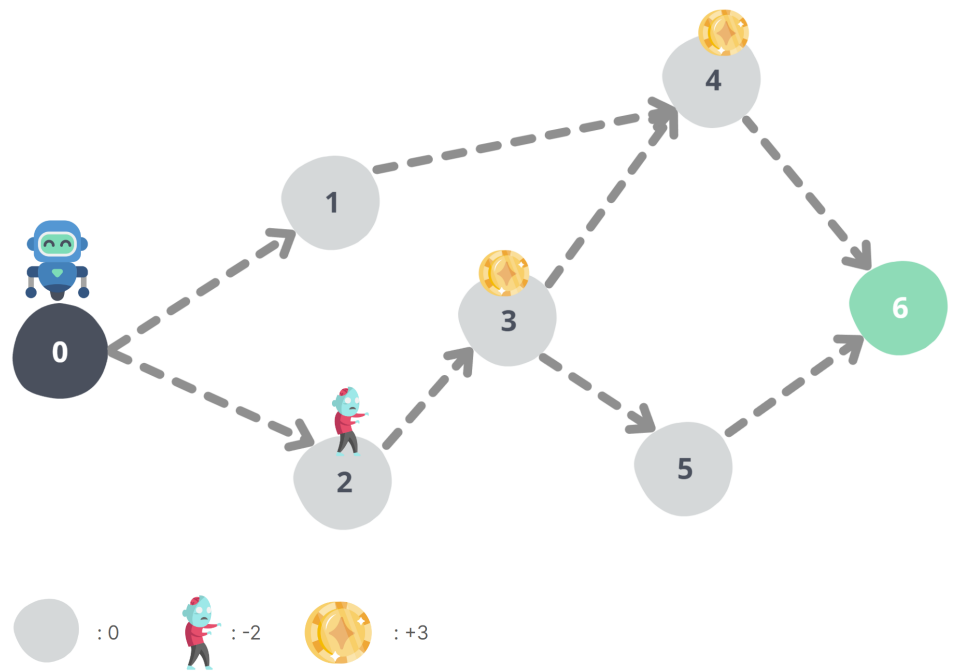
# Starting game
game.play(ia)

# Printing results
print("--- Route taken:")
print(ia.route_taken)
print("\n")

```

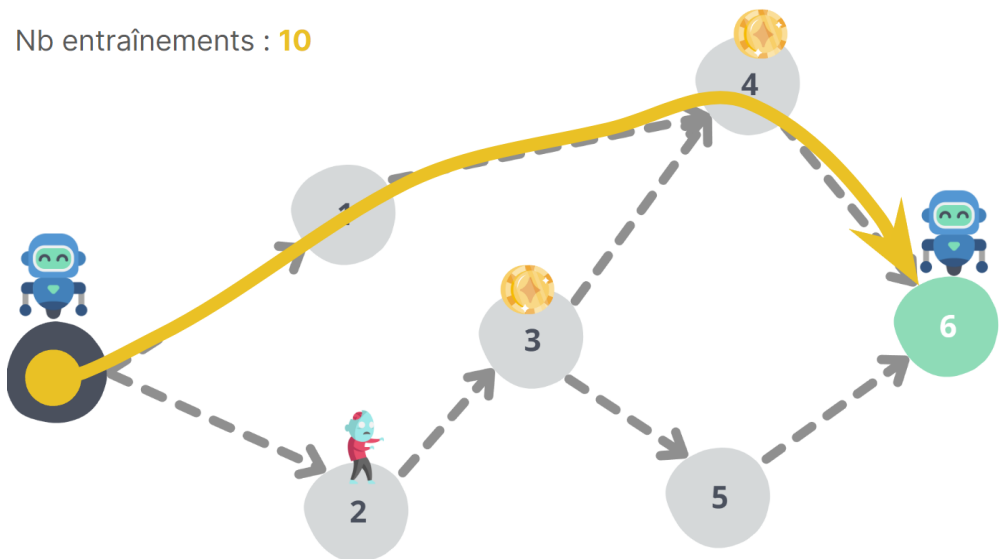
Résultats

Graphe :



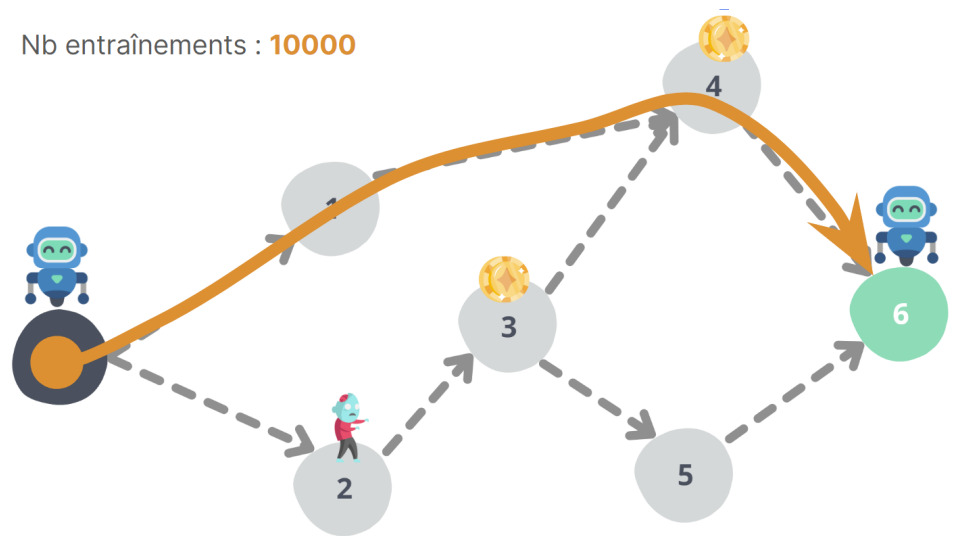
10 entraînements :

Nb entraînements : 10



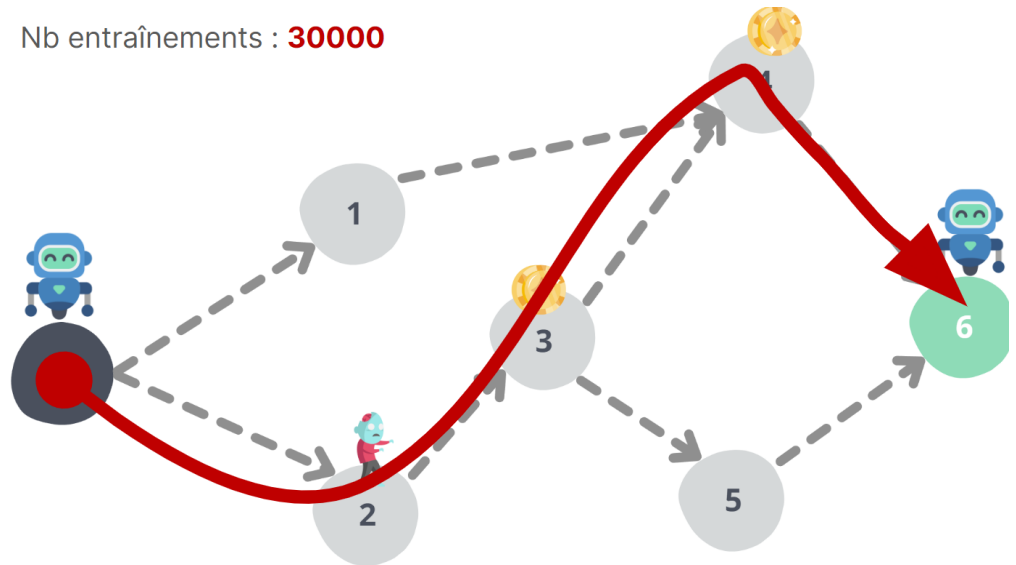
10 000 entraînements

Nb entraînements : 10000



30 000 entraînements : meilleure route prise

Nb entraînements : 30000



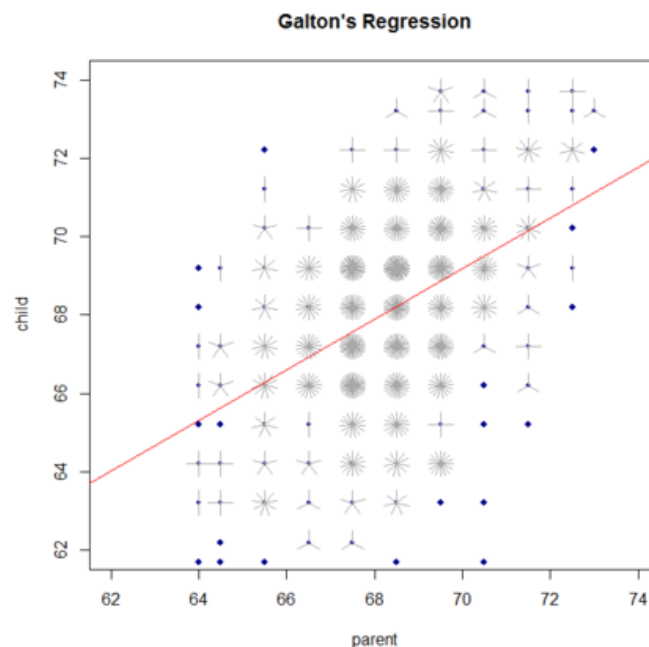
Conclusion

Lors de nos essais, nous nous sommes rendus compte que plus le graphe augmentait en nombre de nœuds, plus il fallait d'entraînements pour choisir la bonne route.

4. Régression linéaire

La régression linéaire est une technique d'analyse de données qui prédit la valeur de données inconnues en utilisant une autre valeur de données connue. Les modèles de régression linéaire sont relativement simples et fournissent une formule mathématique facile à interpréter pour générer des prévisions.

Dans le cas d'une régression linéaire, on souhaite obtenir une fonction affine, d'où son nom car on obtient une droite d'ajustement.



On notera qu'il existe plusieurs types de régression choisies de manière empirique pour résoudre un problème: linéaire, polynomiale...

Méthode des moindres carrés

Pour tracer une droite d'ajustement, on peut utiliser la méthode des moindres carrés: c'est-à-dire qu'on cherche la droite qui rend minimale la somme des carrés des écarts des valeurs observées y à la droite $y = ax + b$. On cherche donc à minimiser a et b dans :

$$\sum_{i=0}^n (y_i - (ax_i + b))^2$$

Pour cela, on trouvera chaque coefficient de la manière suivante:
pour a on utilisera la covariance de X et de Y dans l'ensemble de valeur et
pour b on la déduira de a.

$$\frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

En voici un exemple de résolution :

$$a = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} \quad \text{ou bien} \quad a = \frac{\sum X_i \cdot Y_i}{\sum X_i^2}$$

avec $X_i = x_i - \bar{x}$ $Y_i = y_i - \bar{y}$
 $\bar{x} = \frac{\sum x_i}{n}$ $\bar{y} = \frac{\sum y_i}{n}$

on a alors : $b = \bar{y} - a \cdot \bar{x}$

Qualité de la régression

Une fois la fonction trouvée, il est possible d'en vérifier la qualité grâce au coefficient de corrélation linéaire qui est défini par :

$$r_{xy} = \frac{COV_{xy}}{S_x S_y}.$$

On obtient un nombre entre -1 et 1 et on cherchera à ce que la valeur absolue du coefficient se rapproche le plus de 1. On estime de manière arbitraire que la régression est acceptable lorsqu'elle est au dessus de 0,85 :

$$|r_{xy}| \geq \frac{\sqrt{3}}{2} = \sqrt{0,75} = 0.866 \dots$$

Parfois, on calcule plutôt le coefficient de détermination r^2 par souci de simplicité pour ne plus gérer la valeur absolue noté :

$$R^2 = r_{xy}r_{xy}$$

Fiabilité des prévisions

D'après le rapport de l'université de mathématiques de Côte d'Azur, dans le cas de la régression linéaire, "il peut sembler naturel d'utiliser une valeur prédite pour compléter les données initiales dans l'intervalle des valeurs de X , on se gardera de prédire sans de multiples précautions supplémentaires des valeurs de X en dehors de cet intervalle. En effet il se peut que la relation entre X et Y ne soit pas du tout linéaire mais qu'elle nous soit apparue comme telle à tort parce que les x sont proches les uns des autres".

Néanmoins sa simplicité fait que cette méthode est très couramment utilisée de nos jours dans de nombreux domaines.

Application sur un cas réel (python)

Comme vu précédemment, nous allons tenter de vérifier une prédiction dans un intervalle donné plutôt que dans un intervalle inconnu.

Contexte

Nous allons tenter de vérifier une affirmation tenue de la communauté de joueurs des jeux-vidéos Riot Games.

Ce sont des jeux compétitifs exclusivement en ligne mettant face à face deux équipes composées de cinq joueurs chacune. Pour cet exemple, nous prendrons le jeu League Of Legends qui a pour but qu'une des deux équipes détruise la base adverse.

Il est dit que pour maintenir l'attention d'un joueur sans le lasser du jeu, il faut le maintenir à environ 50% de parties gagnées. Riot Games aurait donc mis en place un fonctionnement interne qui

essaie de tendre les joueurs à ce pourcentage en jouant sur la composition des équipes à chaque partie (matchmaking).

Plus un joueur gagnerait, plus il aurait de chances de tomber avec des alliés faibles et/ou toxiques.

Plus un joueur perdrait, plus il aurait de chances de tomber avec des alliés forts et/ou respectueux.

Et bien d'autres exemples émis par la communauté...

Nous allons donc tenter de vérifier ce fait grâce à la régression linéaire.

Raisonnement

Pour cela, nous allons récupérer un certain nombre de nos parties respectives et vérifier le coefficient de croissance (a) en se basant sur les victoires et les défaites.

On va tracer un nuage de points en suivant ce système (je n'ai pas réussi à faire un système en LaTeX) :

$r = 1$ si victoire

$r = -1$ si défaite

$$U_{n+1} = U_n + r$$

avec n le numéro de la partie.

Ainsi, si le coefficient de croissance tend à 50% tandis que le nombre de parties analysées augmentent, l'affirmation est vraie, sinon soit elle est fausse, soit ce n'est pas le bon moyen de le prouver.

Mise en place

Nous avons construit dans le main.py un menu paramétrable pour gérer les préférences en terme de joueur, du nombre, du type et des informations de parties mais aussi de la méthode de régression (linéaire ou polynomiale) etc...


```

#-----#
# PARAMETERS #
#-----#
# Related to RIOT
SUMMONER_NAME = "Worst Picks Ever"
NB_GAMES = 166
GAME_TYPE = QueueType.RANKED
EXTRA_INFOS = ['championName', 'kills', 'deaths', 'assists']

# Related to regression
REGRESSION_METHOD = RegressionMethod.ORDINARY_LEAST_SQUARES
REGRESSION_FOR_EACH_RESULT = False

# Related to chart style
TITLE = f"Tendance de victoire sur les {NB_GAMES} dernières parties de {SUMMONER_NAME} en {GAME_TYPE.value}"
COLOR_COLUMN = 'championName'
DARK_MODE = False

#-----#
# LOADING PLOT #
#-----#
fig = plot_games(SUMMONER_NAME, NB_GAMES, GAME_TYPE, EXTRA_INFOS,
                 REGRESSION_METHOD, REGRESSION_FOR_EACH_RESULT,
                 TITLE, COLOR_COLUMN, DARK_MODE)
fig.show()
exit(0)

```

Pour récupérer les parties, nous allons utiliser l'[API de Riot Games](#) au travers d'un [paquet python](#), maintenu par Hugo CASTELL, qui aide à requêter l'api et gérer ses spécificités (maximum de requêtes à la minute...)

```

try:
    api = API_LOL(APIKEY, Region.EUROPE, Server.EU_WEST)

    print("Connecting to RIOT API")
    print(f"Games' data loading from API : {nb_games} games")
    last_games = (api.get_summoner_by_name(summoner_name)
                  .get_match_history(nb_matches=nb_games, queue=game_type,
                                     load_infos=True))

    api.close()

    print("Games' data reading")
    for index, game in enumerate(last_games):
        player_infos = game.get_infos_of_summoner()

        for col in columns:
            filtered_games[col].append(player_infos[col])

        print(
            f"recovered: GAME {index + 1} PLAYING {player_infos['championName']}"
        )

except Exception as e:
    print(e)
    print("Il y'a un problème sur l'API, le nom du joueur ou le server")
    exit(1)

```

Puis nous construirons notre suite ainsi :

```

1 usage  Hugo-CASTELL
def calc_progression(win_loss_table):
    final_table = []

    for game_result in win_loss_table:
        # Récupération du dernier score
        if len(final_table) != 0:
            last_result = final_table[-1]
        else:
            last_result = 0

        # Calcul du nouveau score
        current_score = last_result + (1 if game_result is True else -1)

        # Ajout à la liste finale
        final_table.append(current_score)
    return final_table

```

Finalement, pour faire la régression linéaire et afficher la courbe, nous avons utilisé la [librairie Plotly](#) (trendline équivaut à la régression linéaire).

```

1 usage  Hugo-CASTELL
def plotly_build_linear_regression(games, abscissas_column_name, ordinales_column_name, regression_type,
                                regression_for_each_result,
                                color_column_name=None, title=None, darkmode=False):
    # Si c'est une colonne de bool, on met le true en vert et le faux en rouge
    custom_win_lose_color_if_bool = {True: 'green', False: 'red'} if color_column_name is not None and type(
        games[color_column_name][0]) is bool else None

    # Dessins des points avec régression selon le type fourni
    fig = px.scatter(pd.DataFrame(games),
                    x=abscissas_column_name,
                    y=ordinales_column_name,
                    trendline=regression_type.value,
                    trendline_scope="trace" if regression_for_each_result is True else "overall",

                    title=title,
                    hover_data=[column_name for column_name in games.keys()],
                    color=color_column_name,
                    color_discrete_map=custom_win_lose_color_if_bool,
                    template="plotly_dark" if darkmode else "plotly_white",
                    )

    return fig

```

Résultats

Nous allons tout d'abord partir sur deux tentatives avec un faible intervalle :

100 parties : (Compte de Egxon)



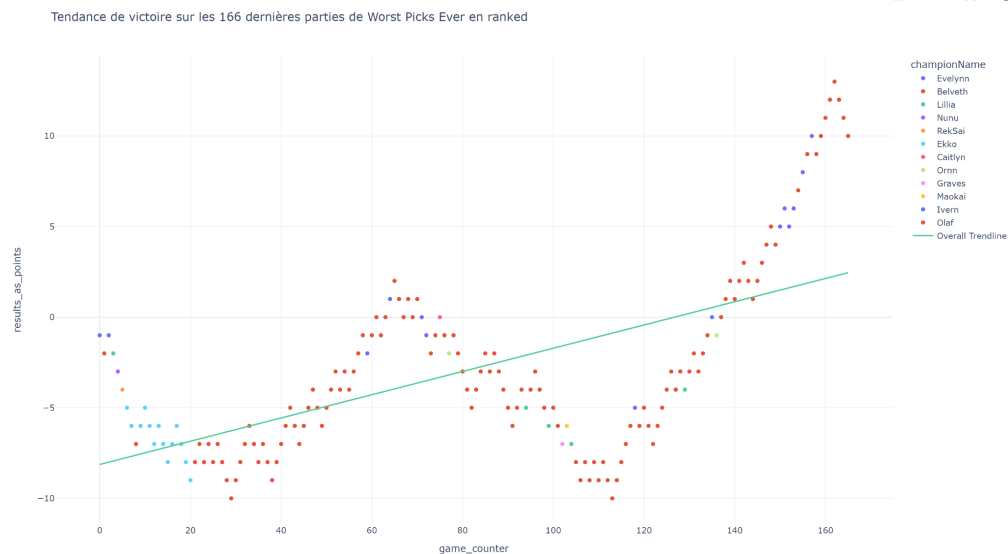
OLS trendline

$$\text{results_as_points} = -0.0327153 * \text{game_counter} + 2.33941$$
$$R^2 = 0.196367$$

game_counter=99

results_as_points=-0.8994059 (trend)

166 parties : (Compte de Hugo)

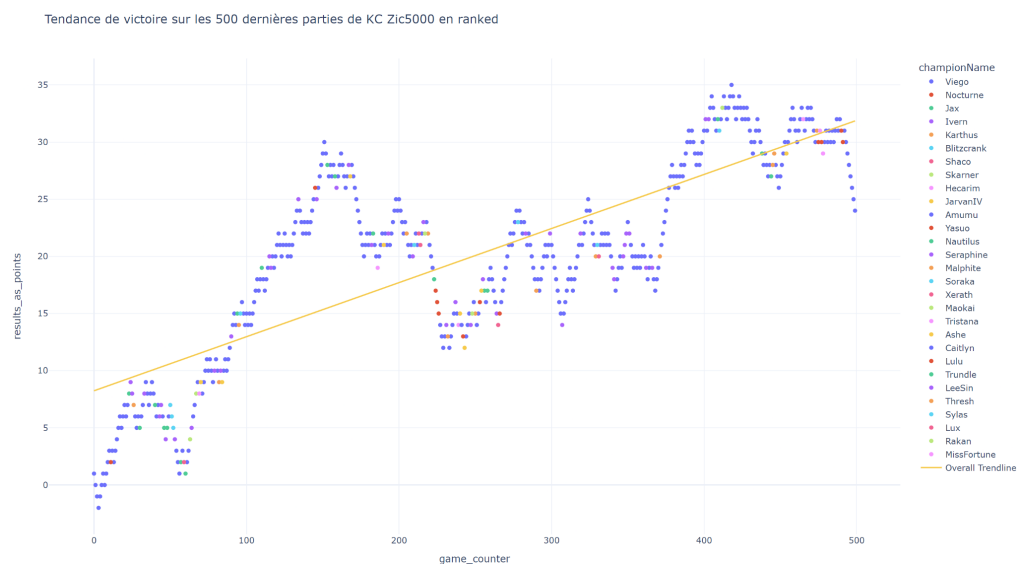


OLS trendline
 $\text{results_as_points} = 0.064142 * \text{game_counter} + -8.12907$
 $R^2=0.364486$
 $\text{game_counter}=165$
 $\text{results_as_points}=2.454368$ (trend)

Jusqu'ici, on remarque que les coefficients de croissance sont très disparates pour que les données nous apportent un résultat exploitable : -0.032 et 0.064 équivalent à 32% et 64% de victoire sur les échantillons respectifs.

Nous avons alors pris le maximum de parties sur le compte qui en avait le plus pour avoir le résultat le plus précis.

500 parties : (Compte de Egxon)



OLS trendline
 $\text{results_as_points} = 0.0473042 * \text{game_counter} + 8.2456$
 $R^2=0.634904$
 $\text{game_counter}=499$
 $\text{results_as_points}=31.8504$ (trend)

On obtient alors un coefficient à 0.047 soit 47% de taux de victoire.

Conclusion

D'après nos résultats, le taux de victoire tend bien vers 50% lorsque l'échantillon de parties augmente.


Néanmoins, il faudrait essayer sur plus de comptes pour réellement prouver cette théorie.

De plus, l'échantillon de parties ne peut pas se faire sur plusieurs saisons de jeu (une saison par an) car il est impossible de récupérer les matchs de la saison précédente. Il faut donc attendre la fin de la saison pour effectuer les tests avec suffisamment de comptes.


L'affirmation est donc partiellement prouvée.

5. Sources


Apprentissage supervisé :

- <https://blent.ai/blog/a/apprentissage-supervise-definition>
- [https://fr.wikipedia.org/wiki/Apprentissage_supervis%C3%A9#:~:text=L'aprentissage%20supervis%C3%A9%20\(supervised%20learning,r%C3%A9gression%20des%20probl%C3%A8mes%20de%20classement.](https://fr.wikipedia.org/wiki/Apprentissage_supervis%C3%A9#:~:text=L'aprentissage%20supervis%C3%A9%20(supervised%20learning,r%C3%A9gression%20des%20probl%C3%A8mes%20de%20classement.)
- <https://scikit-learn.org/stable/index.html>
- https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
-  PYTHON SKLEARN: KNN, LinearRegression et SUPERVISED LEARNING...

Apprentissage non - supervisé :

-  APPRENTISSAGE NON-SUPERVISÉ avec Python (24/30)
- https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html

Apprentissage par renforcement :

-  Apprentissage par renforcement #1 : Introduction
- https://www-igm.univ-mlv.fr/~dr/XPOSE2014/Machine_Learning/A_Sarsa.html

Régression linéaire :

- <https://aws.amazon.com/fr/what-is/linear-regression/>
- <https://math.univ-cotedazur.fr/~diener/MAB07/MCO.pdf>
- https://fr.m.wikipedia.org/wiki/M%C3%A9thode_des_moindres_carr%C3%A9s
- https://fr.wikipedia.org/wiki/R%C3%A9gression_lin%C3%A9aire
- <http://www.profecogest.fr/statistiques/ajustement-statistique/4-methode-des-moindres-carres/>
-  The Riot API with Python Part 1: Our First API Call

6. Modules Python utilisés

Liste des technologies utilisées :

- <https://matplotlib.org/>
- <https://plotly.com/python/>
- <https://numpy.org/>
- <https://pandas.pydata.org/>
- <https://developer.riotgames.com/>
- <https://scikit-learn.org/stable/>
- https://github.com/Hugo-CASTELL/riot_api_manipulation