

# Prédiction du taux de victoire au jeu League of Legends

Projet IAS

Hongyu YAN, Shuangrui CHEN, Shiqing HUANG, Zhenhai XU, Liuyi CHEN

Tuteur: Kirian Guiller

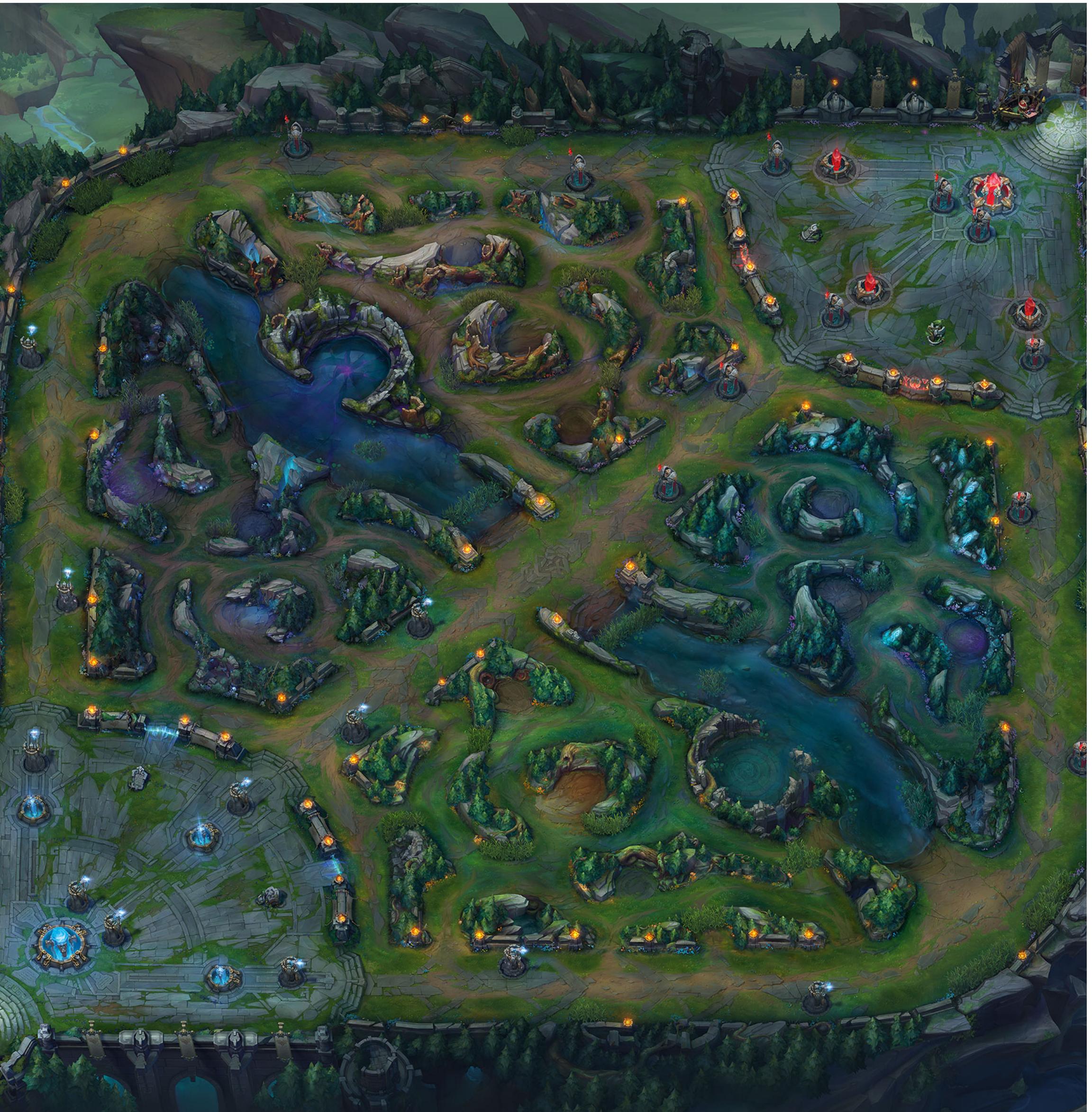
Prof: François Landes, Kim Gerdes

# Introduction

# Introduction

## Multiplayer Online Battle Arena

- Jeu incomplet
- Équité au niveau du jeu
- Inégalité au niveau du choix
- Nombreux facteurs



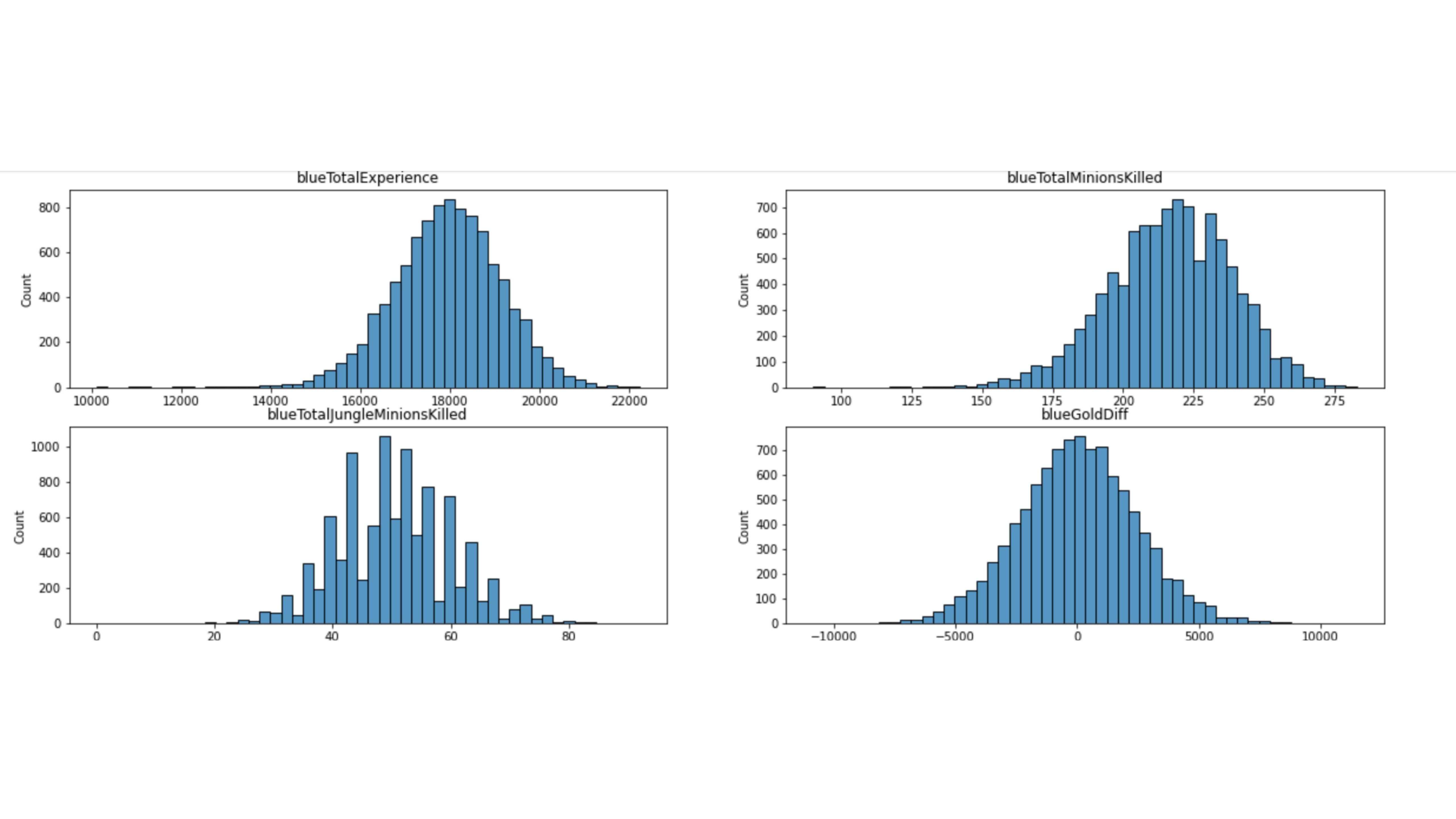
# Collection des données

# Observation du dataset initiale

## League of Legends(LOL) - Ranked Games 2020

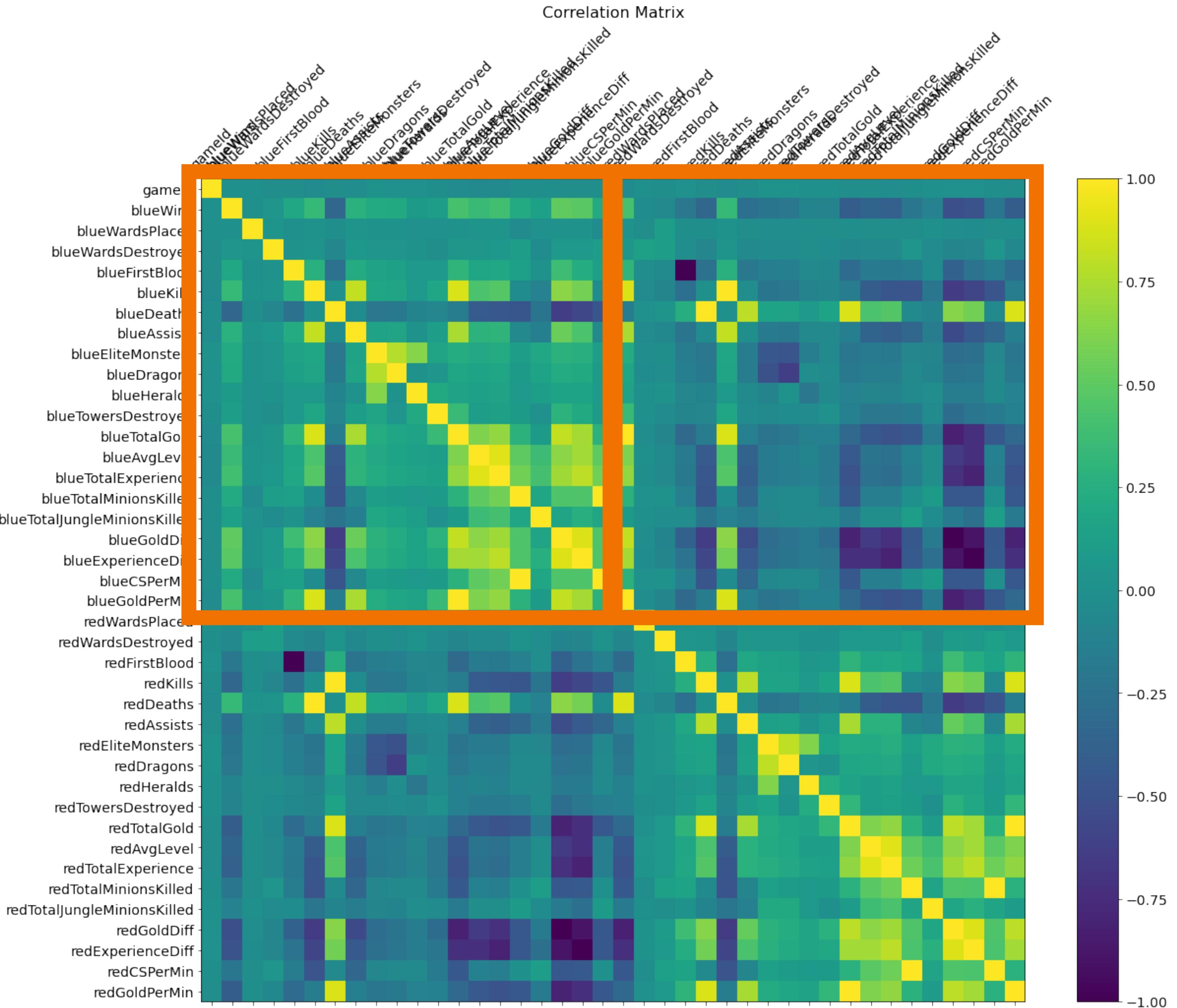
- Dimension: (9879, 40)
- 38 Features

	gameId	blueWins	blueWardsPlaced	blueWardsDestroyed	blueFirstBlood	blueKills	blueDeaths	blueAssists	blueEliteMonsters	blueDragons	...
0	4519157822	0	28	2	1	9	6	11	0	0	...
1	4523371949	0	12	1	0	5	5	5	0	0	...
2	4521474530	0	15	0	0	7	11	4	1	1	...
3	4524384067	0	43	1	0	4	5	5	1	0	...
4	4436033771	0	75	4	0	6	6	6	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...
9874	4527873286	1	17	2	1	7	4	5	1	1	...
9875	4527797466	1	54	0	0	6	4	8	1	1	...
9876	4527713716	0	23	1	0	6	7	5	0	0	...
9877	4527628313	0	14	4	1	2	3	3	1	1	...
9878	4523772935	1	18	0	1	6	6	5	0	0	...



# Matrix de Correlation

## Matplotlib + Seaborn



# Plus de donnée

Prédiction en temps réel

# LOLWatcher

- Donnée de Temps réel
- Format identique
- Plusieurs models si besoin
- Analyse de différentes étapes d'un jeu

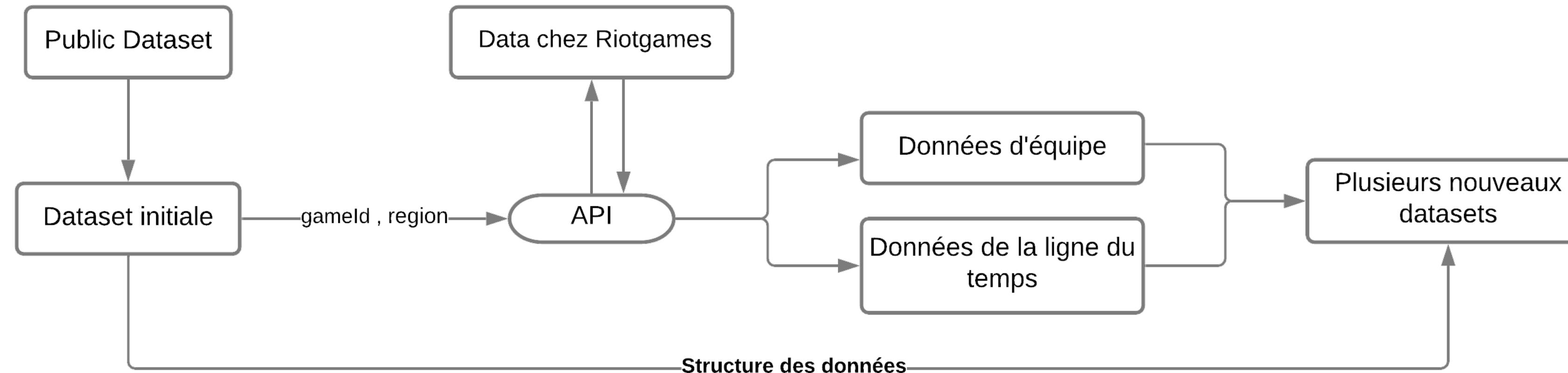
+



# Création des nouveaux dataset

## Procédure de Crawler et d'organiser

- Il nous manque des données pour différentes moments du jeu
- Des nouveaux données obtenues d'après un api public chez Riot Games.



# Pre-Processing

# PCA

**trouver la direction qui maximise la variance des données projetées**

1. Calculer la matrice de covariance C
2. Diagonaliser C
3. Garder seulement les p premières valeur propres
4. Transformer

$$\sigma_i^2 = \frac{1}{N} \sum_n (x'_n - \langle x' \rangle)^2 = \vec{u}_i^T C \vec{u}_i$$

$$C = \frac{1}{N} \sum_n (\vec{x}_n - \langle \vec{x} \rangle)(\vec{x}_n - \langle \vec{x} \rangle)^T$$

$$\sigma_i^2 = \vec{u}_i^T (C \vec{u}_i) = \vec{u}_i^T \lambda_i \vec{u}_i = \lambda_i \vec{u}_i^2 = \lambda_i$$

$$C = U \Lambda U^{-1}$$

$$P = \left( \begin{pmatrix} \vec{u}_1 \\ \vdots \end{pmatrix} \begin{pmatrix} \vec{u}_2 \\ \vdots \end{pmatrix} \dots \begin{pmatrix} \vec{u}_1 \\ \vdots \end{pmatrix} \right)$$

$$\vec{x}_{n,transformed} = P \cdot (\vec{x}_n - \langle \vec{x} \rangle)$$

# Standard Scaler

## Normalisation

- $x = (x - \text{mean})/\text{std}$
- transformer des données de format différentes en la même format
- transformer des données de différentes distributions en une distribution spécifique.

# Choisir de Pipeline

## Pas d'influence évident

- conserver que les 20 à 25 features pour PCA
- Improvement d'efficacité
  - GBR : -0.181714 +/- (0.006420)
  - GBR - PCA - 20: -0.181295 +/- (0.006808)
  - GBR - PCA - 21: -0.181227 +/- (0.006733)
  - GBR - PCA - 22: -0.180856 +/- (0.006738)
  - GBR - PCA - 23: -0.181234 +/- (0.006937)
  - GBR - PCA - 24: -0.181105 +/- (0.007181)
  - GBR - PCA - 25: -0.181475 +/- (0.007013)
  - GBR - PCA - 26: -0.181413 +/- (0.007080)
  - GBR - PCA - 27: -0.181354 +/- (0.007021)
  - GBR - PCA - 28: -0.181346 +/- (0.007063)
  - GBR - PCA - 29: -0.181438 +/- (0.006879)

# Sélection du model

# Modèles à sélectionner

## Tester par SkLearn Pipeline

- Tree LinearRegression
  - Agrégation Lasso
  - Lineaire ElasticNet
- KNeighborsRegressor
- DecisionTreeRegressor
- GradientBoostingRegressor
- RandomForestRegressor

# Test Résultat

Sur l'ensemble de donnée en 10ème min

Test neg\_mean\_squared\_error :

LR: -0.181060 +/- (0.005322)

LASSO: -0.250062 +/- (0.000100)

EN: -0.250062 +/- (0.000100)

KNN: -0.215678 +/- (0.006887)

CART: -0.363409 +/- (0.018109)

GBM: -0.181719 +/- (0.006451)

RFR: -0.188095 +/- (0.005822)

Test neg\_mean\_absolute\_error :

LR: -0.372796 +/- (0.005849)

LASSO: -0.500059 +/- (0.000094)

EN: -0.500059 +/- (0.000094)

KNN: -0.367813 +/- (0.008821)

CART: -0.365563 +/- (0.019347)

GBM: -0.359290 +/- (0.006576)

RFR: -0.367437 +/- (0.005293)

Test r2 :

LR: 0.274981 +/- (0.021468)

LASSO: -0.001320 +/- (0.001862)

EN: -0.001320 +/- (0.001862)

KNN: 0.136377 +/- (0.027217)

CART: -0.456190 +/- (0.063436)

GBM: 0.272326 +/- (0.025810)

RFR: 0.249210 +/- (0.022572)

# Model Gradient Boosting Regression

## Algorithme

Initialisation du modèle :  $F_0(x) = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$

pour  $m = 1$  à  $M$  :

A.  $r_{im} = -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}, i = 1, 2 \dots N$

B. créer un nouvel arbre de régression et des feuilles  $R_{jm}, j = 1 \dots J_m$

C. Pour  $j = 1 \dots J_m, \gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)\nu$

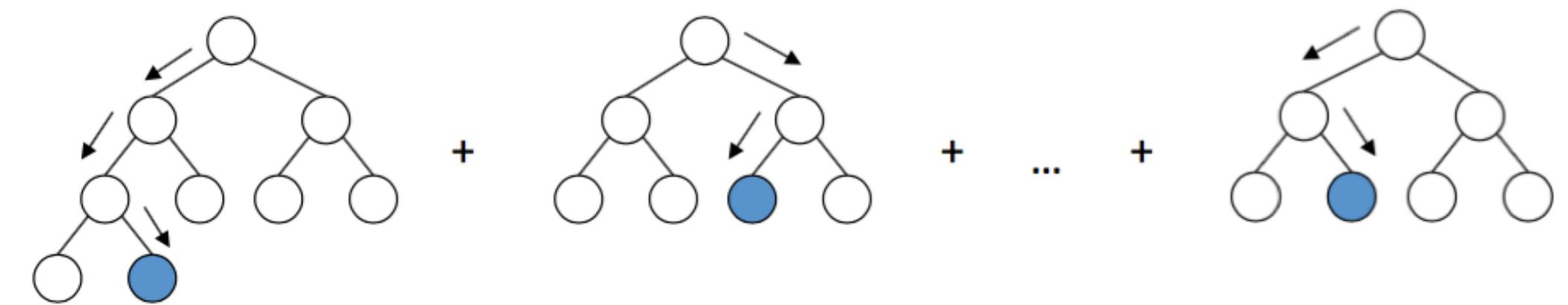
D.  $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

Sortie  $F_M(x)$

# Avantages

## Gradient Boosting Regression

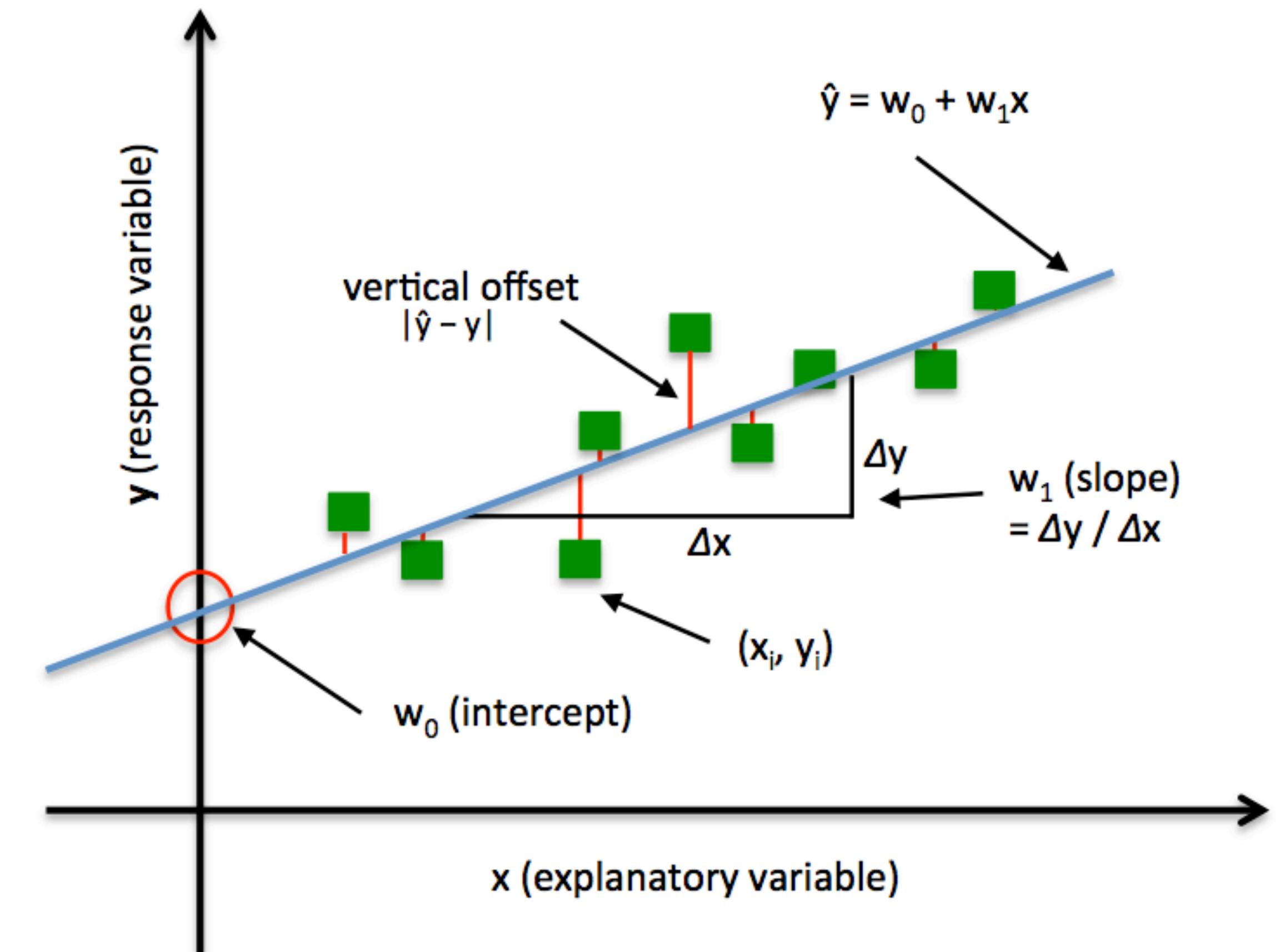
- Assurer un faible écart
- Bien traiter notre dataset
- Résistance élevée à l'overfitting



# Linear Regression

## Mathématique

- Formule principale:  
 $y^{(i)} = w_i x^{(i)} + w_0$
- Model simple et efficace
- Sans besoin de trouver les meilleurs hyper-paramètres
- Pas assez compliqué pour différentes étapes d'un jeu



# Fonction de score

## 3 Choix

- $R^2 = 1 - \frac{\sum_i (\hat{y}^{(i)} - y^{(i)})^2}{\sum_i (\bar{y}^{(i)} - y^{(i)})^2}$
- $MSE = \sum_{i=1}^n (y_i - y_i^p)^2$
- $MAE = \sum_{i=1}^n |y_i - y_i^p|$
- Pourquoi nous choisissons R2?

# Optimisation de hyper-paramètre

# Ajuster les paramètres avec GridSearchCV

## Méthode de SkLearn

Méthode d'appel :

```
sklearn.model_selection.GridSearchCV(estimator, param_grid)
```

Entrez le estimateur que nous utilisons:

```
param_grid = {  
    'pca_n_components': range(20, 25),  
    'gbr_n_estimators': [50, 100, 150],  
    'gbr_learning_rate': [0.1, 0.3, 0.01],  
    'gbr_loss': ['ls', 'lad', 'huber', 'quantile']  
}
```

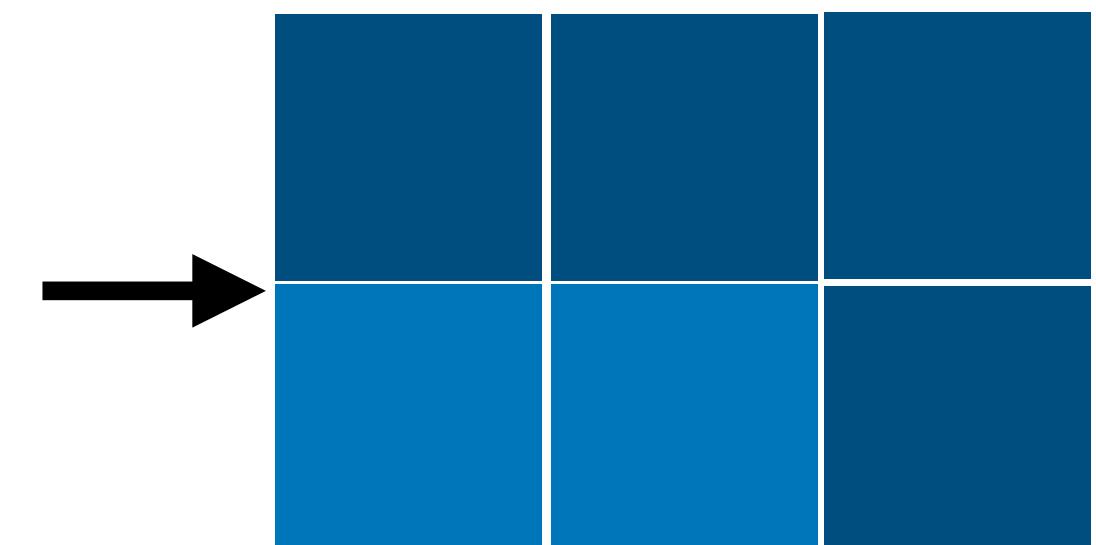
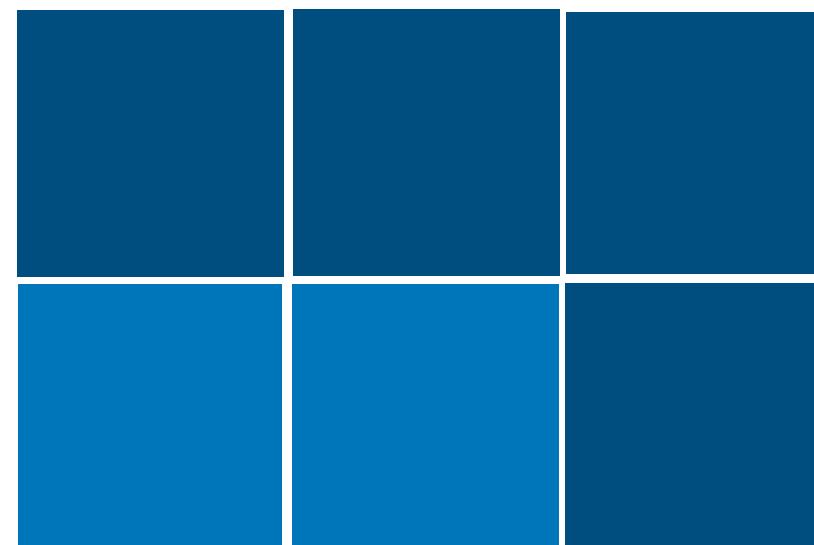
# Nombre d'weak-learner

Comment fonctionner?

Données réelles:



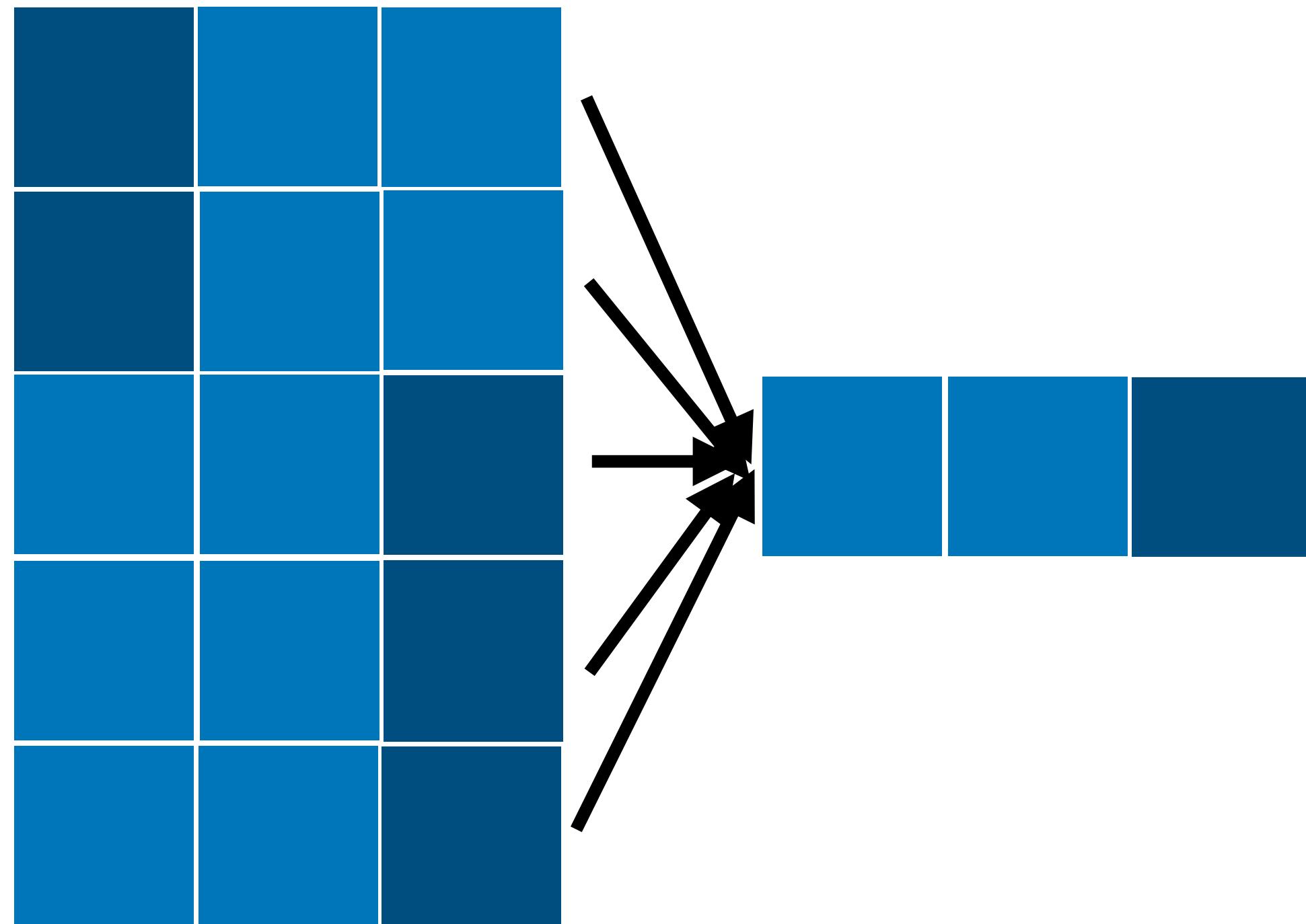
Taux de prédictat: 66%



Taux de prédictat: 66%

**Under Fitting**

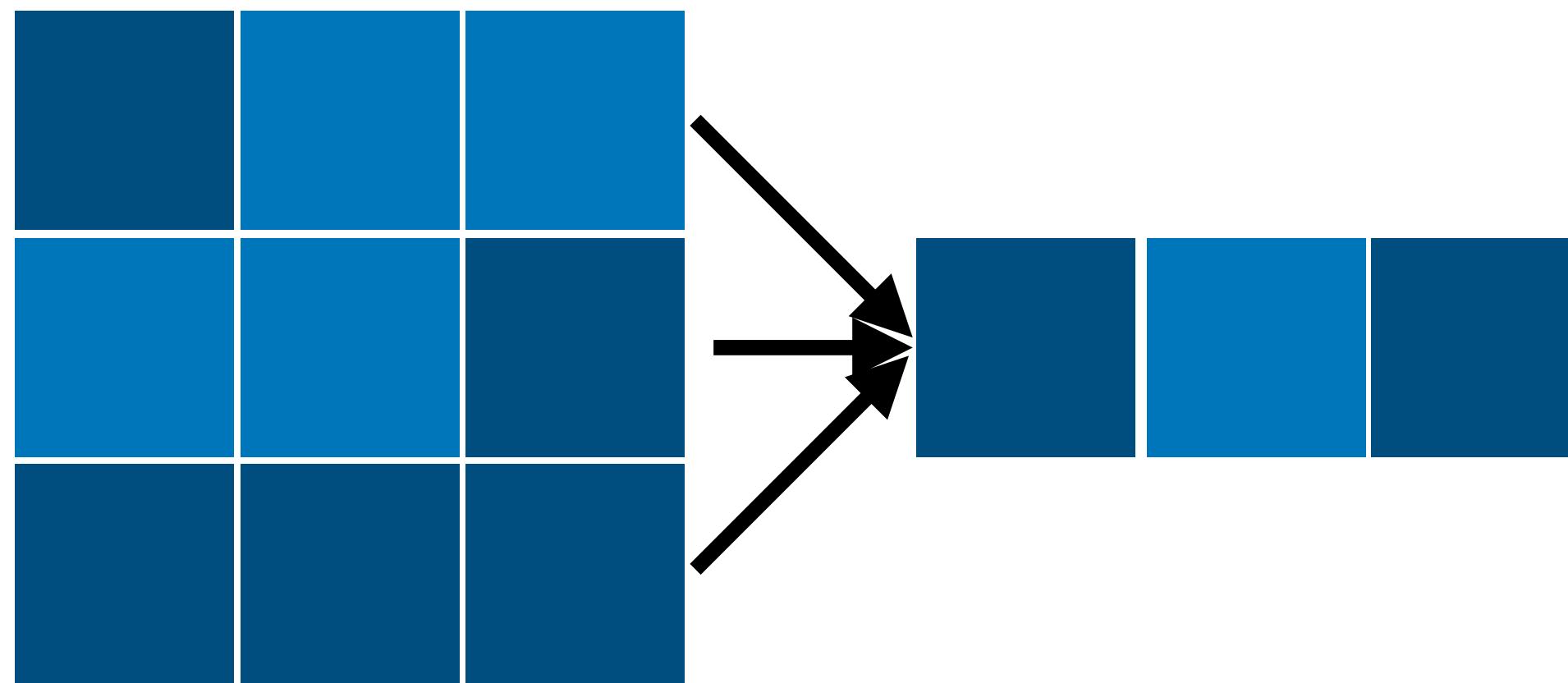
Taux de prédictat: 66%



Taux de prédictat: 66%

**Over Fitting**

Taux de prédicat: 66%



Taux de prédicat: 100%

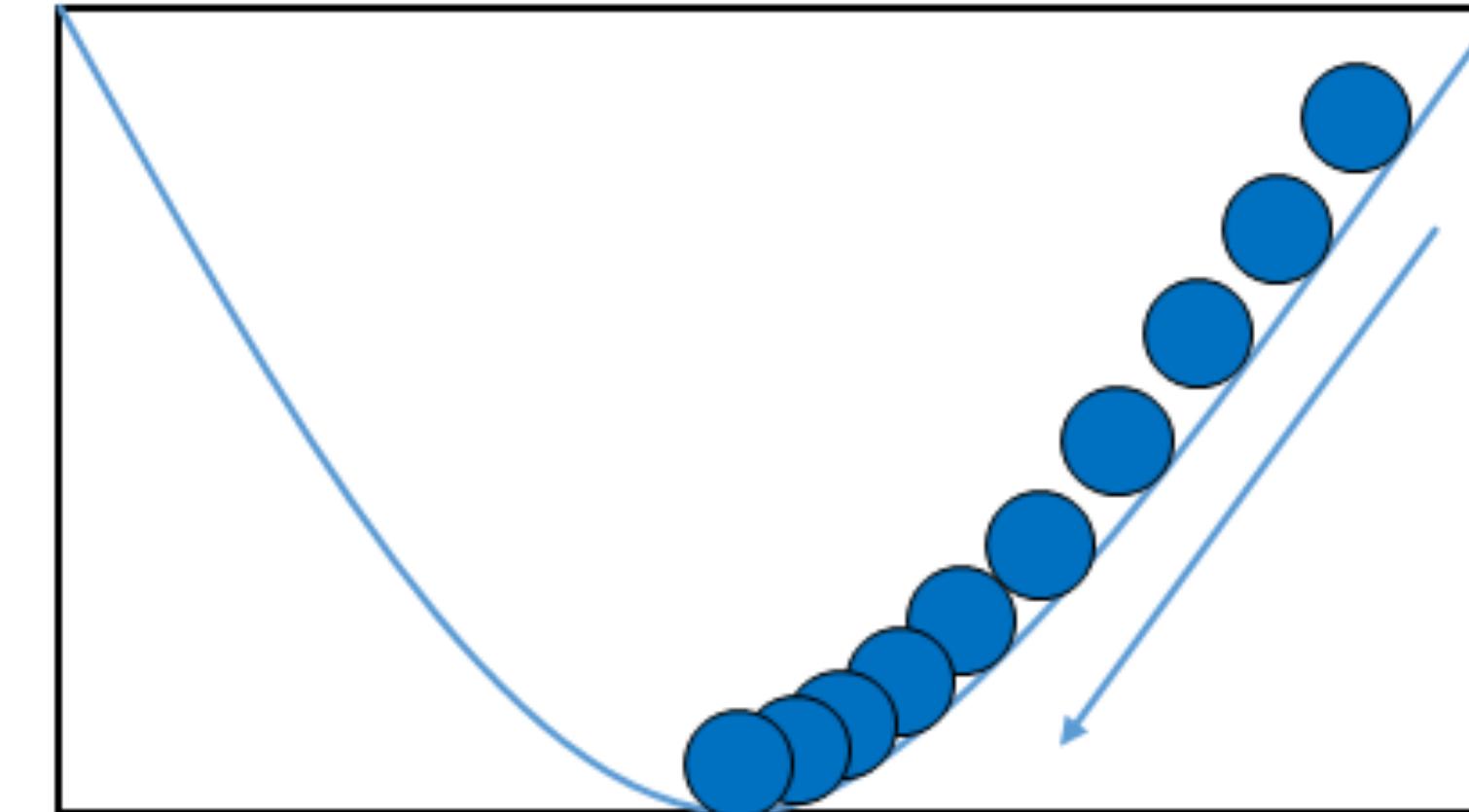
**Best Fitting**

# Taux d'apprentissage

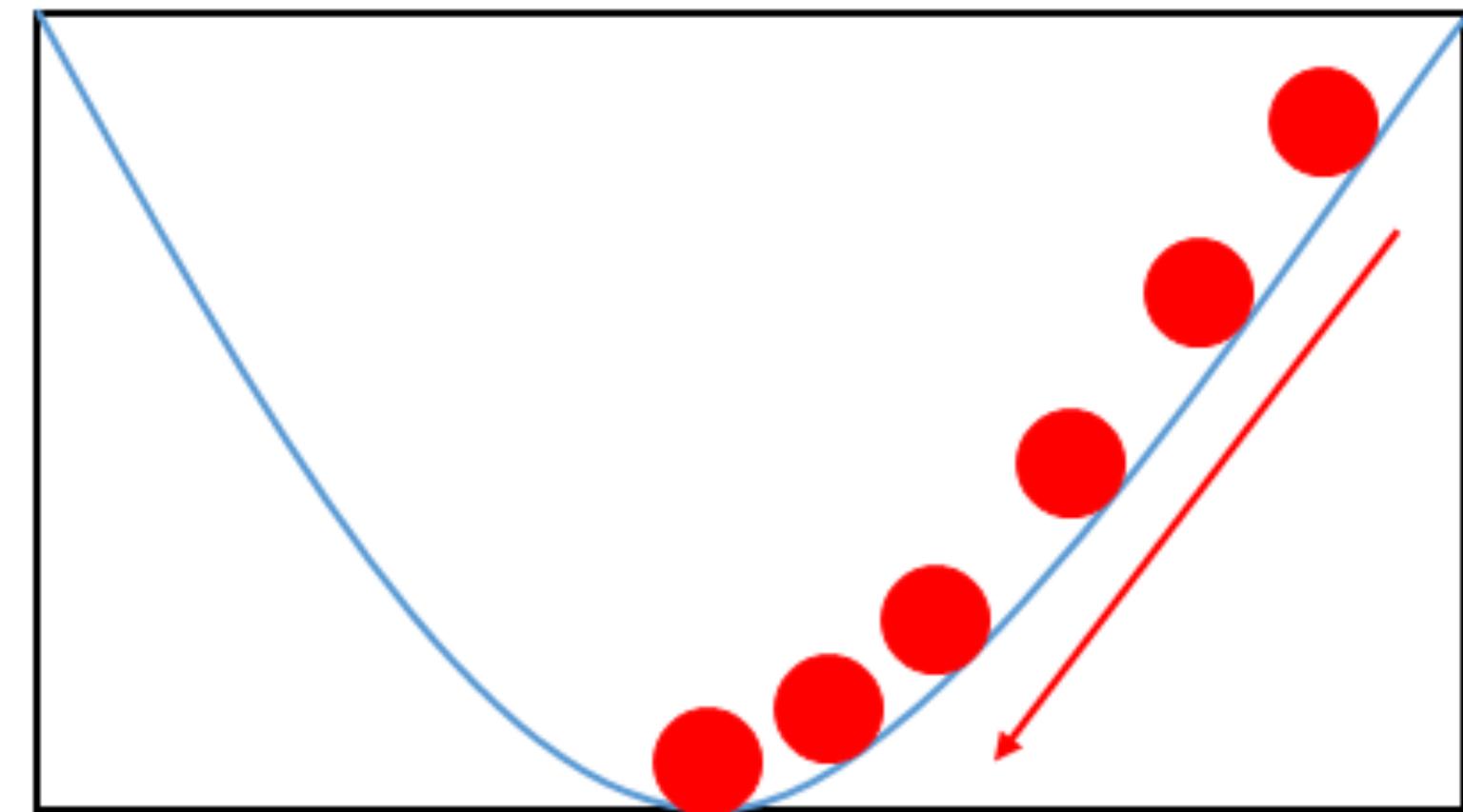
## Learning rate

- Nous devons tenir compte du nombre d'apprenants et du taux d'apprentissage

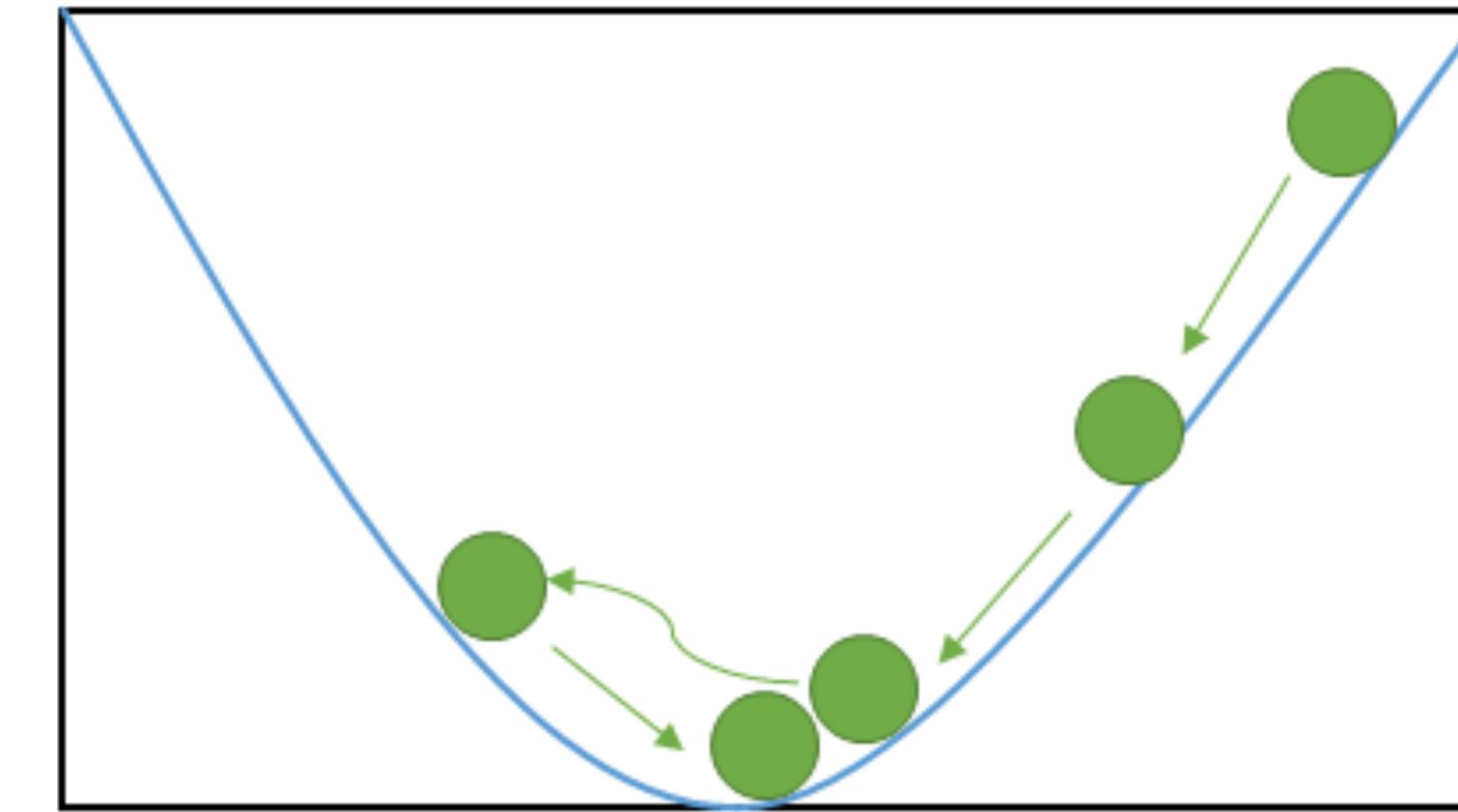
Learning rate trop faible



Learning rate approprié



Learning rate excessif



# Loss Mathématiques

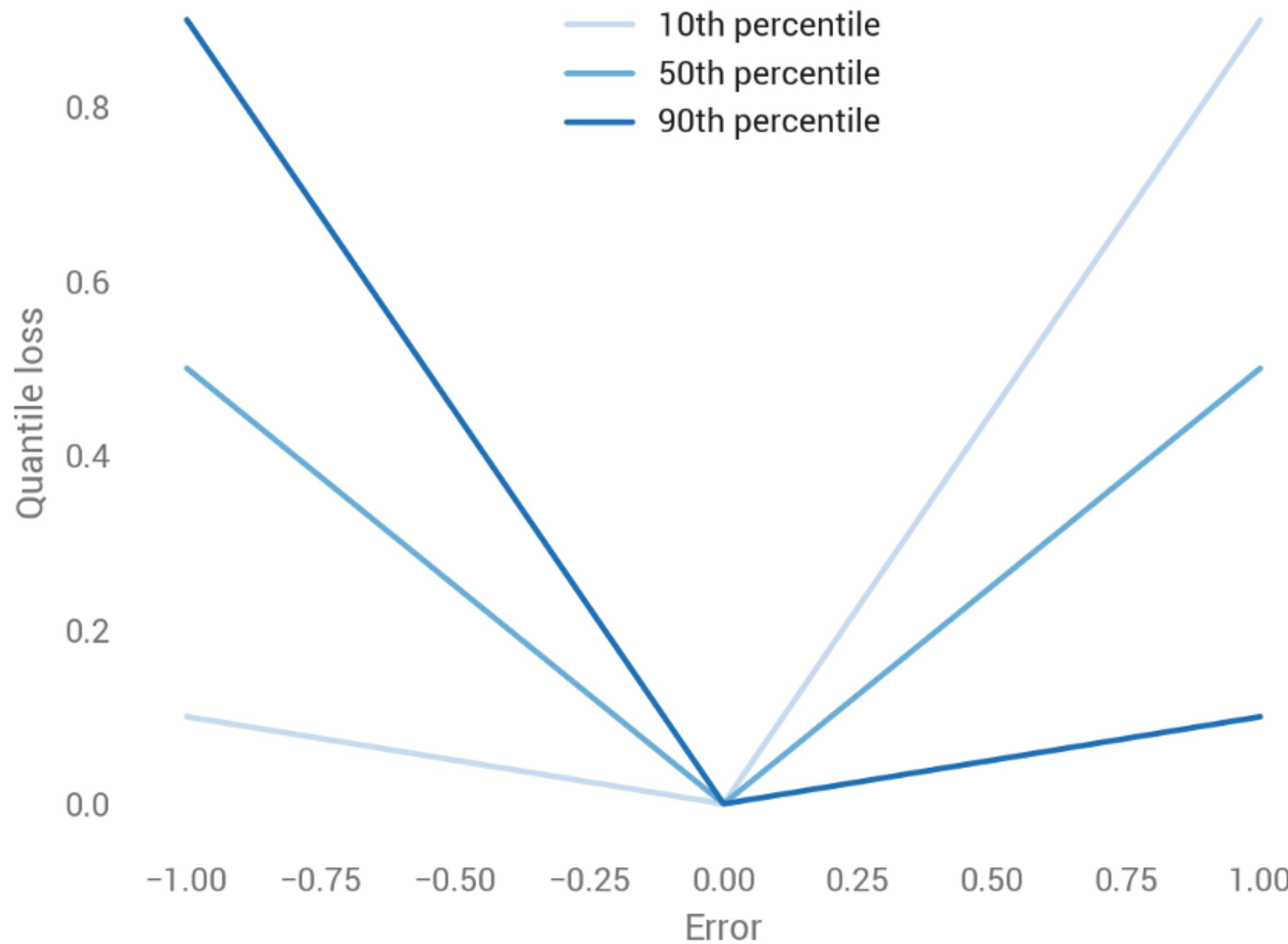
$$MAE = \sum_{i=1}^n (y_i - y_i^p)$$

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2, & |y - f(x)| \leq \delta \\ \delta |y - f(x)| - \frac{1}{2}\delta^2, & |y - f(x)| > \delta \end{cases}$$

$$MSE = \sum_{i=1}^n (y_i - y_i^p)^2$$

$$L_\gamma(y, y^p) = \sum_{i:y_i < y_i^p} (1 - \gamma) |y_i - y_i^p| + \sum_{i:y_i \geq y_i^p} \gamma |y_i - y_i^p|$$

## Quantile loss by error and quantile



# Conclusion

# Résultat

## Prédiction de chaque minute

Data 10 mins - Test:

LR: 0.27 +/- (0.02)

GBR: 0.27 +/- (0.02)

Data 30 mins - Test:

LR: 0.48 +/- (0.03)

GBR: 0.49 +/- (0.05)

Data 15 mins - Test:

LR: 0.39 +/- (0.02)

GBR: 0.41 +/- (0.03)

Data 35 mins - Test:

LR: 0.40 +/- (0.05)

GBR: 0.38 +/- (0.07)

Data 20 mins - Test:

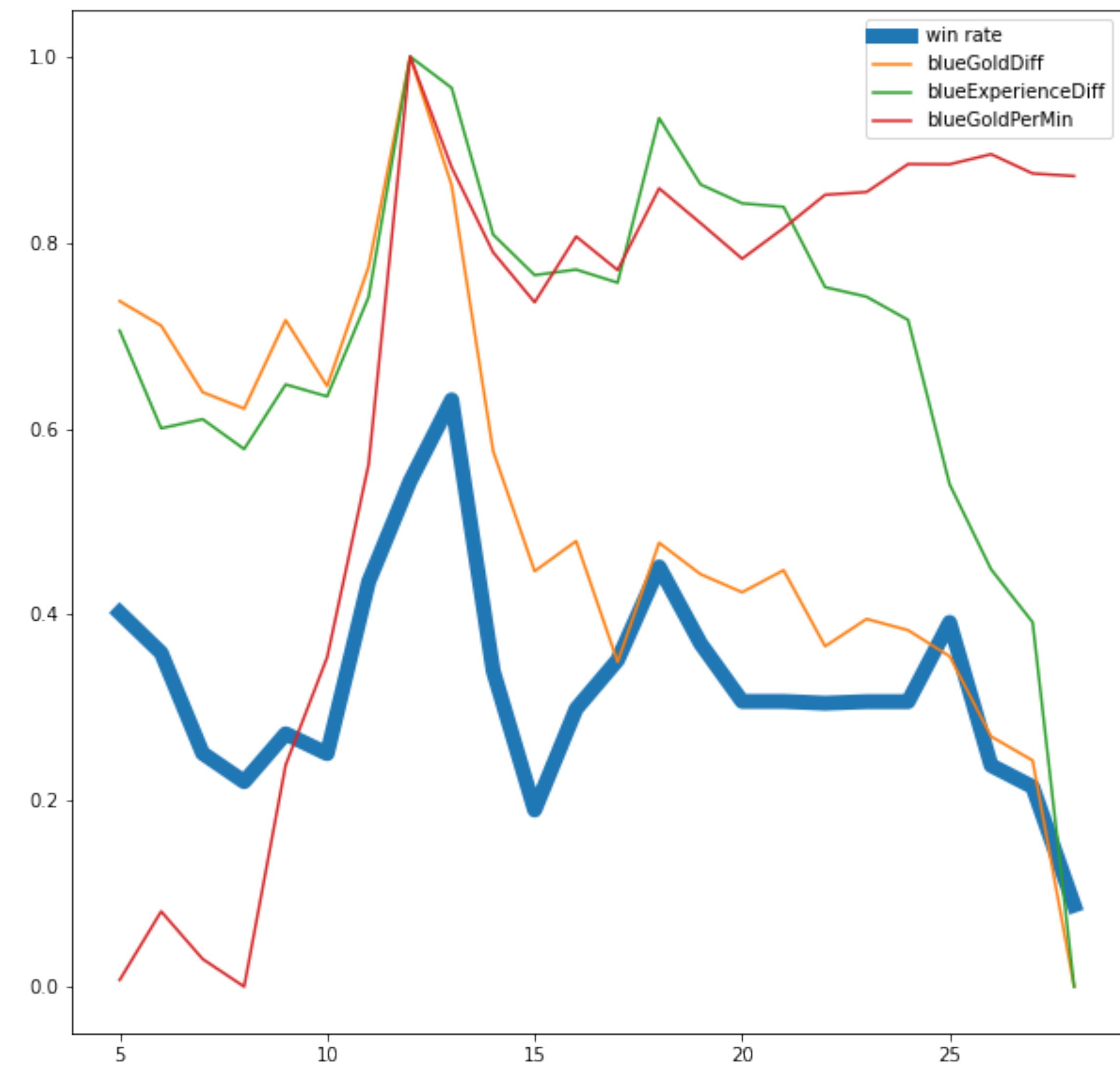
LR: 0.46 +/- (0.01)

GBR: 0.48 +/- (0.01)

Data 25 mins - Test:

LR: 0.50 +/- (0.02)

GBR: 0.52 +/- (0.03)



# À l'avenir

## Encore beaucoup à faire

- Prédiction par ID de joueur
- Interaction graphique réel
- Quantifier l'impact de la sélection des héros et des équipements
- Ajouter l'impact de la position du héros par la carte de chaleur

Merci