Régression linéaire, multiple et polynomiale

Christian Mbarga Mvogo, Jérémy Le Joncour

Sommaire

- Rappel sur la régression linéaire simple, multiple et polynomiale
- Présentation des fonctions
- Présentation des résultats des modèles
- Evaluation des modèles avec et sans SkLearn
- Comparaison
- Conclusion

Régressions

Les modèles créés permettent d'émettre des prédictions éventuelles :

- Régression Linéaire : Une features x et une target y sous forme de fonction affine.
- Régression Polynomiale : Pareil avec une évolution caractéristique polynomiale.
- Régression Multiple : Plusieurs features x et une target y.

Présentation des fonctions (Sans SKlearn)

Ajout de la colonne de biais à x

- X = np.hstack((x, np.ones(x.shape)))

la fonction de notre modèle : $f(x) = X.\theta$

la fonction Cout : $J(\theta) = \frac{1}{2m}\sum(X.\theta - Y)^2$

le gradient : $\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m}X^T.(X.\theta - Y)$

la descente de gradient : $\theta = \theta - \alpha \times \frac{\partial J(\theta)}{\partial \theta}$

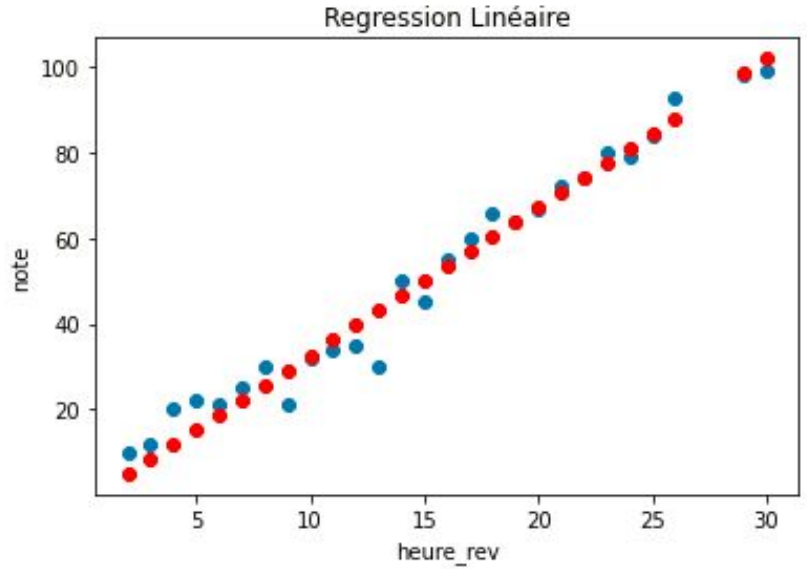
```python
#Fonction Modèle
def model(X, theta):
    return X.dot(theta)

#Fonction Cout
def cout_fonction(X, y, theta):
    m = len(y)
    return 1/(2*m) * np.sum((model(X, theta) - y)**2)

#Fonction Gradient
def gradient(X, y, theta):
    m = len(y)
    return 1/m * X.T.dot(model(X, theta) - y)

#Création d'un tableau de stockage pour enregistrer l'évolution du Cout du modele
def gradient_descente(X, y, theta, alpha, n_iterations):
    cout_historique = np.zeros(n_iterations)

    for i in range(0, n_iterations):
        #Mise a jour du parametre theta
        theta = theta - alpha * gradient(X, y, theta)
        #Enregistrement de la valeur du Cout au tour i dans cost_history[i]
        cout_historique[i] = cout_fonction(X, y, theta)

    return theta, cout_historique
```

Présentation des fonctions (Avec SKlearn)

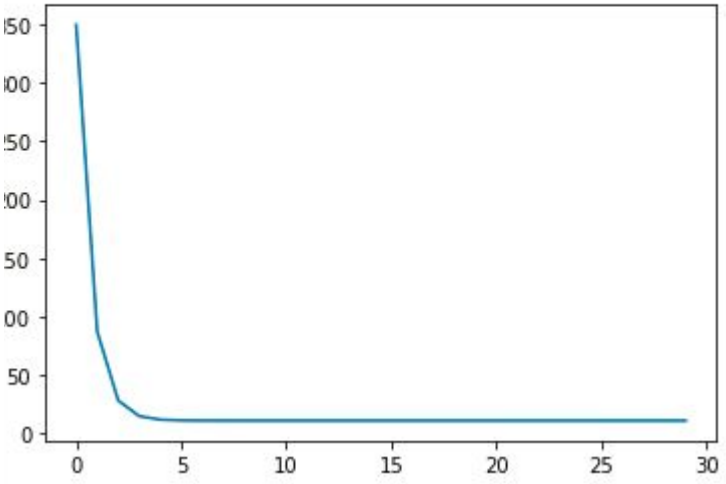
```
#Standardisation
scaler = StandardScaler()
x = scaler.fit_transform(x).round(2)
y = scaler.fit_transform(y).round(2)

#Apprentissage
x_app, x_test, y_app, y_test = train_test_split(x,y,test_size=0.2,random_state=5)

#Modèle
model = linear_model.LinearRegression()
model.fit(x_app, y_app)
print(model.coef_)
print(model.intercept_)

#Prédiction
y_pred = model.predict(x_test)
pred = model.predict(x)
```

Présentation des résultats des modèles

```
[[ 3.46702127]
 [-2.01923751]]
```

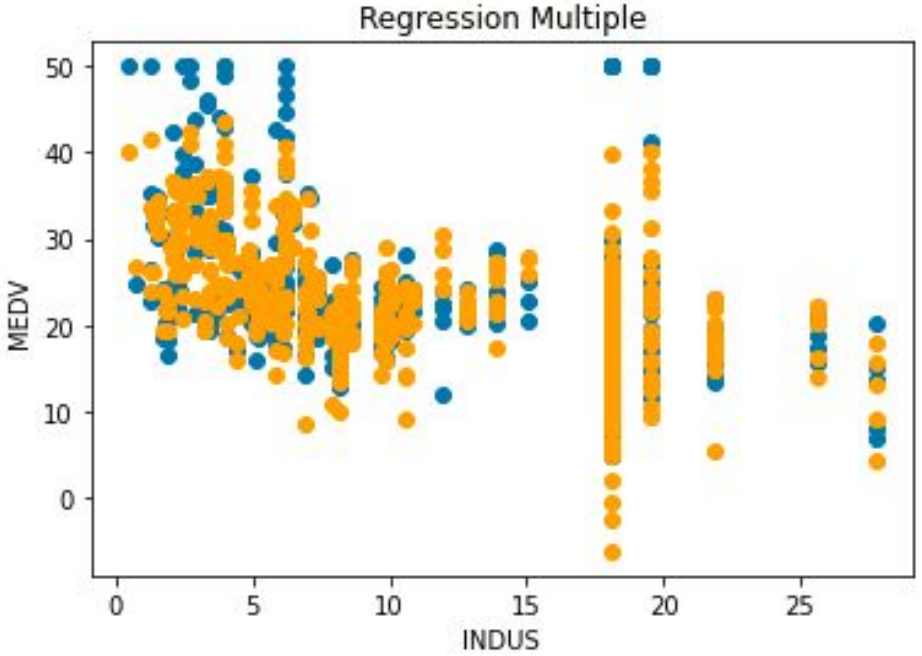


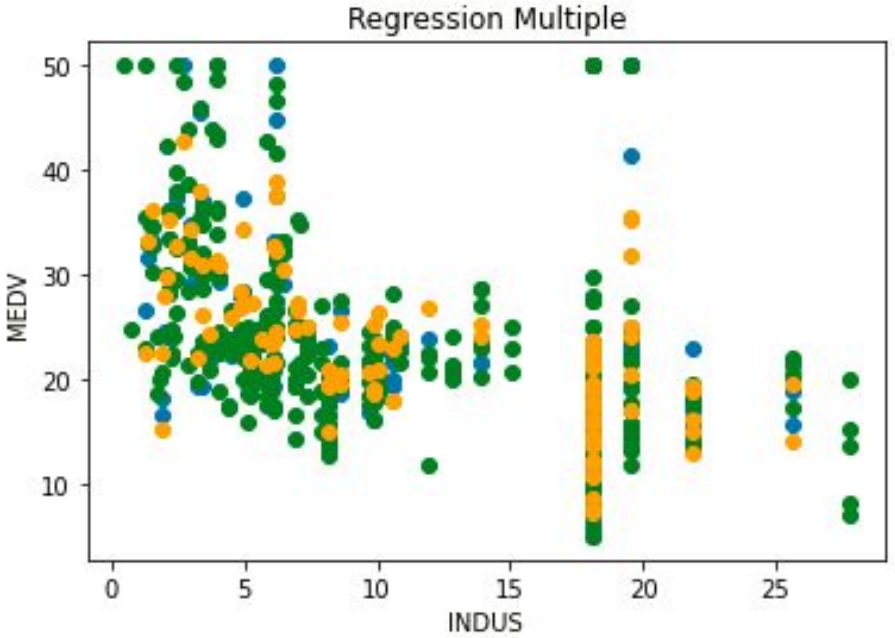

```
plt.plot(range(n_iterations), cout_historique)
plt.show()
```



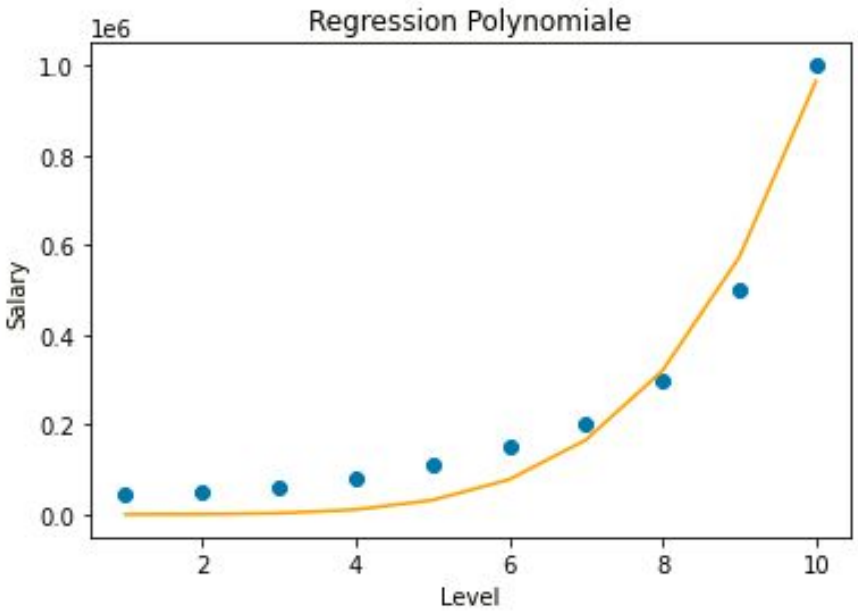
Présentation des résultats des modèles

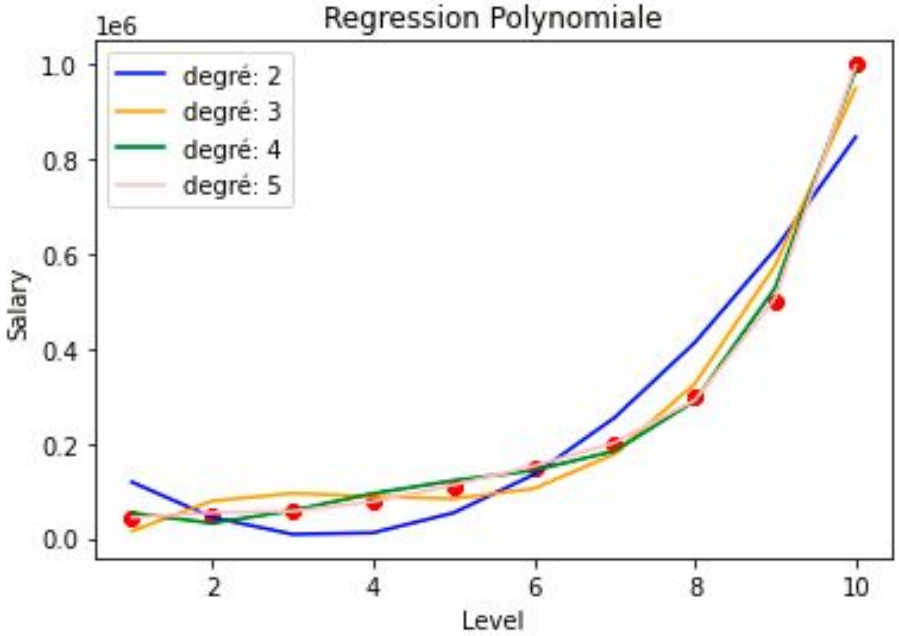
```
plt.scatter(x_app['INDUS'],y_app, c='green')
plt.scatter(x_test['INDUS'], y_pred, c='orange')
plt.show()
```



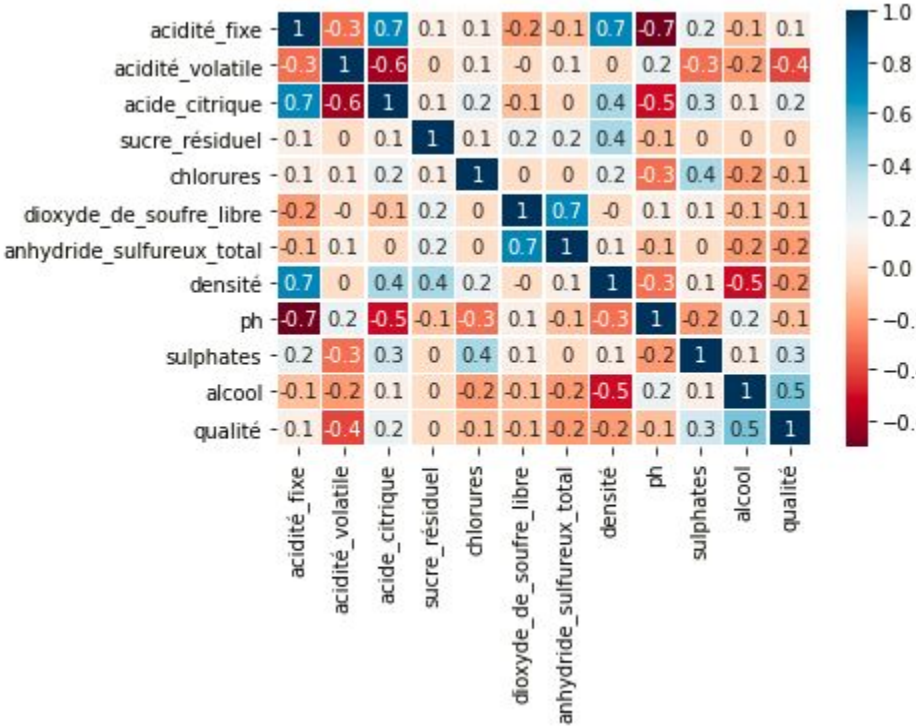


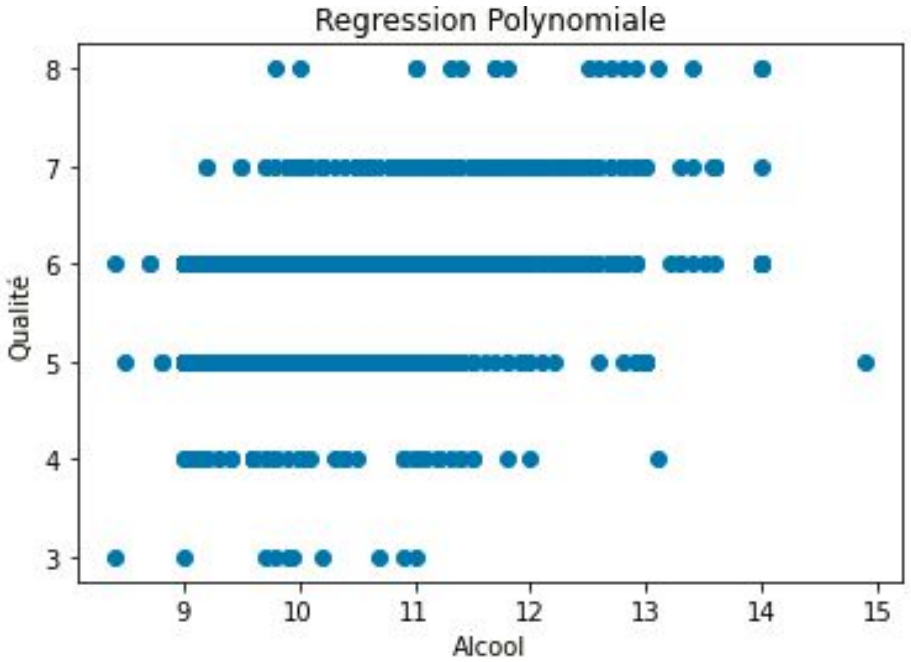
Présentation des résultats des modèles





Présentation des résultats des modèles





Evaluation des modèles avec et sans SkLearn

$$R^2 = \frac{SSR}{SST}$$

$$SSR = \sum_i (\hat{y}_i - \bar{y})^2$$

$$SST = \sum_i (y_i - \bar{y})^2$$

```python
def coefficient_determination(y, pred):
    #Formule de la somme des carrées résiduels
    u = ((y - pred)**2).sum()

    #Formule de la somme des carrées
    v = ((y - y.mean())**2).sum()
    return 1-u/v

coefficient_determination(y, predictions).round(3)
```

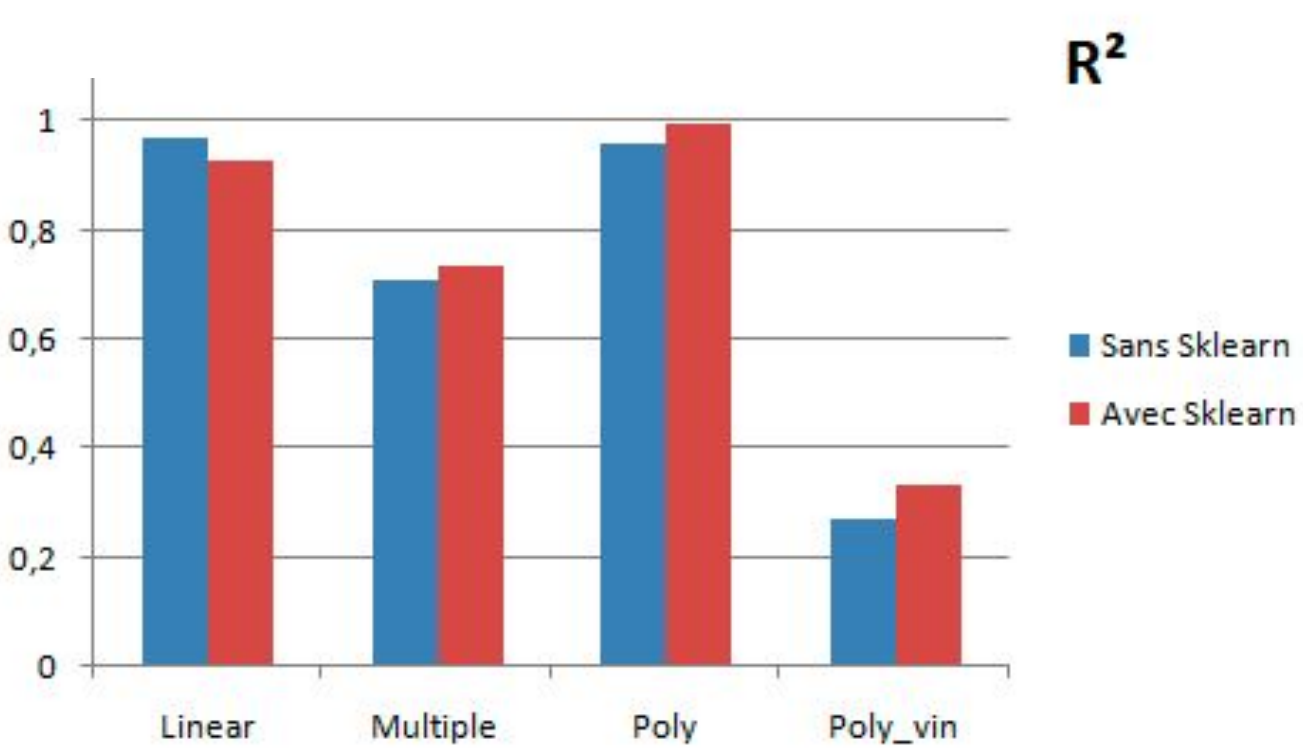
Evaluation des modèles avec et sans SkLearn

```
from sklearn.preprocessing import StandardScaler

#Apprentissage
x_app, x_test, y_app, y_test = train_test_split(x,y,test_size=0.2,random_state=5)

#Modèle
model = linear_model.LinearRegression()
model.fit(x_app, y_app)
print(model.coef_)
print(model.intercept_)

#Prédiction
y_pred = model.predict(x_test)
pred = model.predict(x)
```

```
#Mesure du "taux d'erreur" de notre modèle.
mean_squared_error(y, pred)
```

```
#Renvoie le R² de notre modèle
print(model.score(x_test, y_test))
print(model.score(x, y))
```

Comparaison



Conclusion

- Comprendre le principe de gradient descent à travers la programmation manuelle.
- Caractériser le bon modèle et utiliser les bons outils sur SkLearn.
- Trouver un procédé de sélection des features “viables”.
- Évaluer notre modèle et jouer sur le pas et le nombre d’itération pour obtenir un meilleur coefficient/score.

Conclusion

- Une meilleure vision de l'apprentissage supervisé.
- Une meilleure compréhension des mécanismes et du protocole à suivre pour créer un modèle de prédiction.

- Difficulté sur la partie Qualité_Vin :
  - Visualisation, Sélection des Features, Score faible.

- Encore quelques notions à appréhender pour évaluer correctement le score de notre modèle (Sur test ? Pred/y ? etc..)