Prédiction des décès dus au COVID-19 grâce au Machine Learning

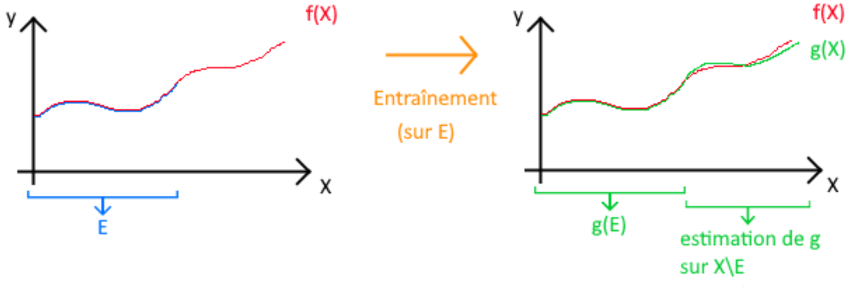
Enzo DE CARVALHO

Numéro d'inscription : 29448

2020-2021

Sommaire

Principe de la démarche



$\hookrightarrow$ le modèle $\hat{g}$ généralise les données connues E fournies

Première approche : regression linéaire

En utilisant ElasticNet

Modèle :

$$\hat{g}_{deces}(\omega, t) = \omega_0 + \omega_1 t$$

t le temps

$\omega = \begin{pmatrix} \omega_0 \\ \omega_1 \end{pmatrix}$ un paramètre à déterminer

Première approche : regression linéaire

En utilisant ElasticNet

Modèle :

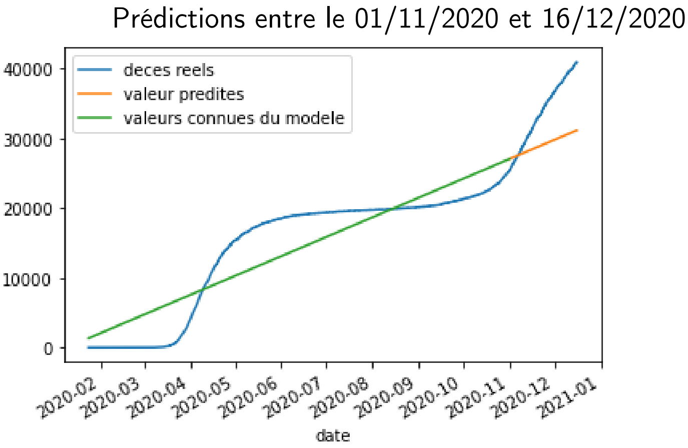
$$\hat{g}_{deces}(\omega, t) = \omega_0 + \omega_1 t$$

t le temps

$\omega = \begin{pmatrix} \omega_0 \\ \omega_1 \end{pmatrix}$ un paramètre à déterminer

Le modèle ElasticNet détermine alors ω
Le résultat dépend des hyperparamètres α et ρ

Première application



$\rho = 0.9$

$\alpha = 0.1$

Figure – Résultats peu satisfaisants...

SVR

Modèle SVR (Régresseur à Support Vectoriel)

Hyperparamètres :

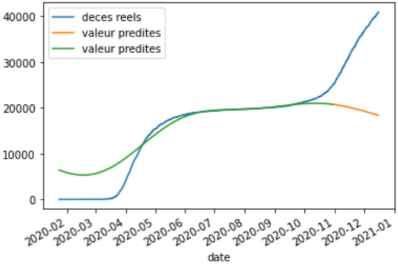
- C le paramètre de régularisation
- ϵ la taille du tube de « non-pénalité »
- γ paramètre du noyeau (`rbf` ici)

SVR

Modèle SVR (Régresseur à Support Vectoriel)

Hyperparamètres :

- C le paramètre de régularisation
- ϵ la taille du tube de « non-pénalité »
- γ paramètre du noyeau (`rbf` ici)



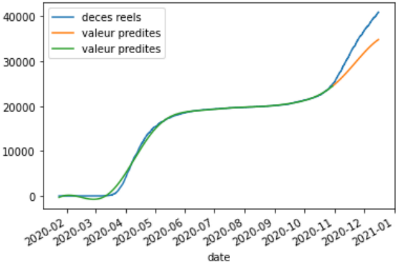


Figure – SVR avec $C = 100$, puis $C = 100000$

Validation Croisée

Grille d'hyperparamètres :

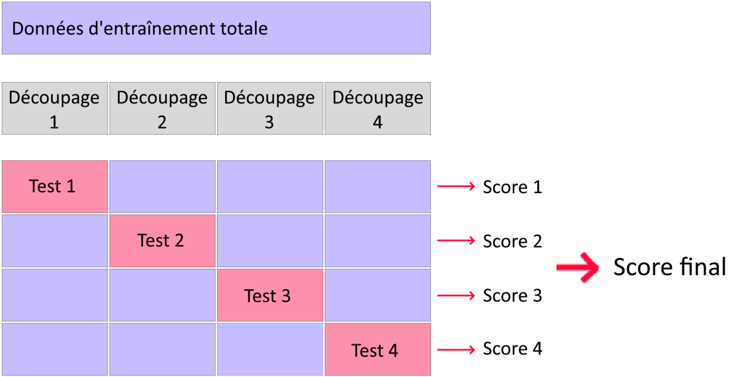
```
params = {'svr__C' : [10, 100, ...],
          'svr__eps' : [0.1, 0.01, ..], ... }
```

Validation Croisée

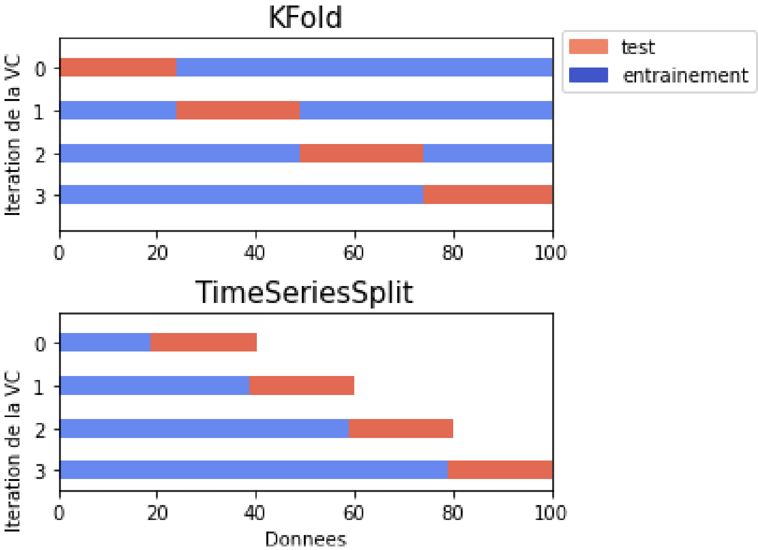
Grille d'hyperparamètres :

```
params = {'svr__C' : [10, 100, ...],
          'svr__eps' : [0.1, 0.01, ..], ... }
```

Pour une combinaison d'hyperparamètres :

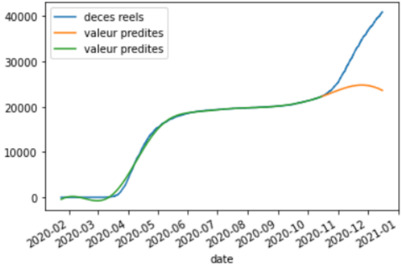


Stratégie pour la Validation Croisée

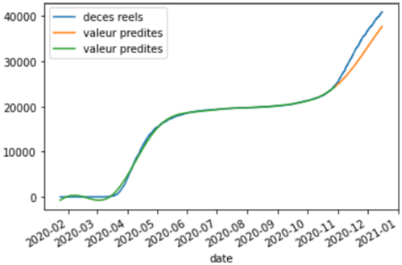


Application avec SVR

```
params = {'svr__C' : [10**i for i in range(1,8)]}
```



{'svr__C' : 100000}



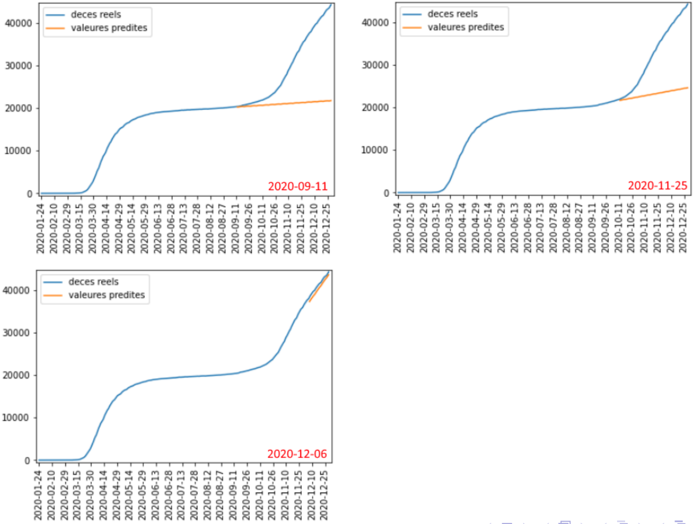
{'svr__C' : 1000000}

Figure – à partir du 15/10/2020, puis du 01/11/2020

$\Rightarrow$ Échec de géneralisation

Prophet

Approche avec le modèle Prophet de Facebook



RegressorChain SVR

Approche multivariée avec RegressorChain

```
params = {
'svr__C' : [10**i for i in range(1,8)],
'svr__epsilon': [0.1,0.01,0.001]}
```

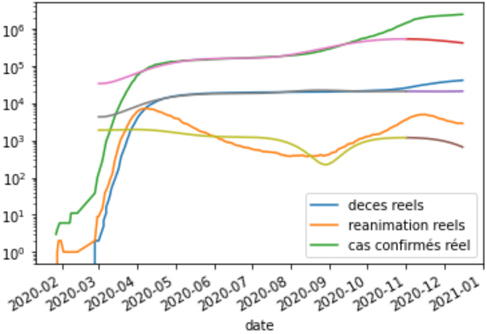
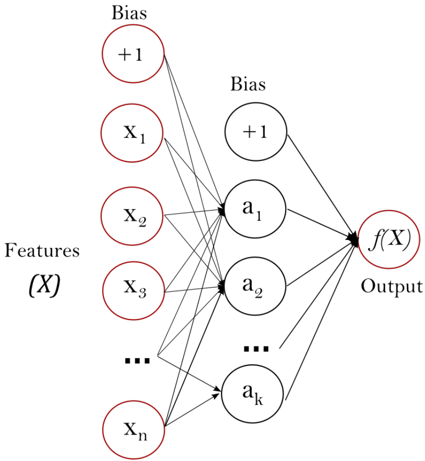


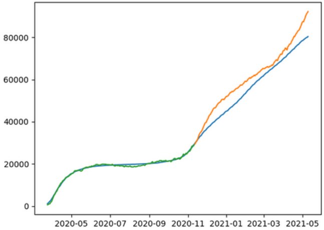
Figure – RegressorChain avec $C = 10000$ et $\epsilon = 0.001$

Réseau neuronal

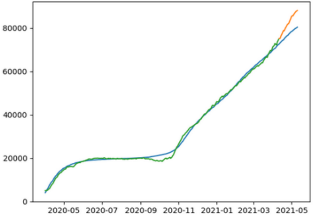
Approche avec des réseaux neuronaux



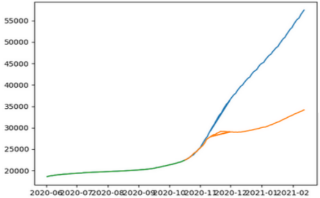
Application avec réseau neuronal



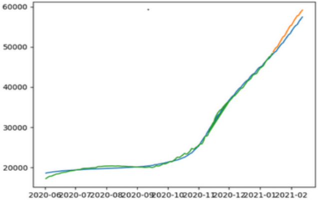
max_iter
100k
date
11/11/20
décalage
7j



max_iter
90k
date
10/04/21
décalage
15j



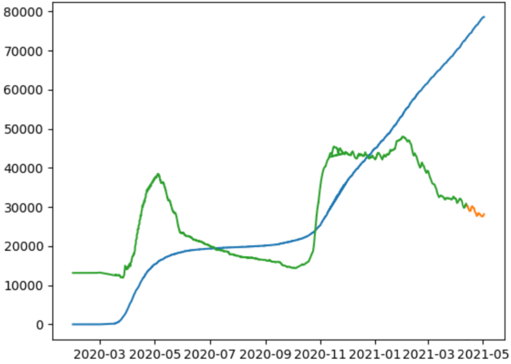
max_iter
90k
date
15/10/20
décalage
7j



max_iter
90k
date
10/01/21
décalage
7j

Application avec réseau neuronal

Taux de corrélation entre
`total_cas_confirmes` et `total_deces_hopital` : 0.977939



max_iter
140k
date
10/04/20
décalage
7j

⇒ échec du modèle sans la courbe des cas confirmés

Conclusion

Plusieurs essais sur plusieurs modèles :

Conclusion

Plusieurs essais sur plusieurs modèles :

$\Rightarrow$ Prédictions justes sur les périodes sans variations

Conclusion

Plusieurs essais sur plusieurs modèles :

$\Rightarrow$ Prédictions justes sur les périodes sans variations

$\Rightarrow$ Difficulté de prédictions sans le point d'inflexion

fonction d'objectif d'ElasticNet

ElasticNet cherche ω tel que :

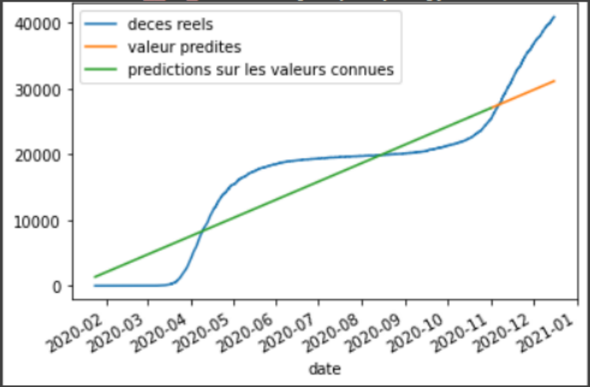
$$\min_{\omega} \frac{1}{2n_{deces}} ||\hat{g}_{deces}(\omega, t) - f(t)||_2^2 + \alpha\rho|\omega| + \frac{\alpha(1-\rho)}{2}||\omega||_2^2$$

α et ρ les hyperparamètres définissant le modèle,

f la courbe réelle des décès.

Détails sur la regression linéaire

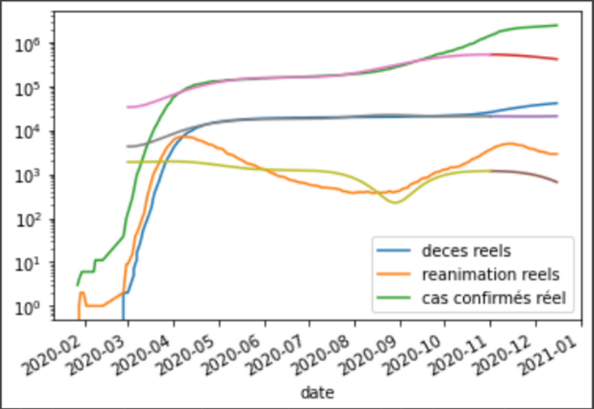
```
params = {
    'elasticnet__alpha' : [i/100 for i in range(10,100)],
    'elasticnet__l1_ratio' : [0.5,0.8,0.9]}
```




```
deces     622
dtype: int64
{'elasticnet__alpha': 0.1, 'elasticnet__l1_ratio': 0.9}
```

Détails sur la regression multivariée

```
model = make_pipeline(StandardScaler(), RegressorChain(SVR(), order=[0,2,1]
params = {'regressorchain__base_estimator__C': [10**i for i in range(2,8)],
          'regressorchain__base_estimator__epsilon': [0.001,0.1,0.01],
          'regressorchain__base_estimator__kernel': ['rbf']}
```




```
{'regressorchain__base_estimator__C': 10000, 'regressorchain__base_estimator__epsilon': 0.001,
'regressorchain__base_estimator__kernel': 'rbf'}
```

Annexes

```
##Methode SVR##
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import ElasticNet
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error
from sklearn.model selection import TimeSeriesSplit
```

Annexes

```
################################################################################
svr = SVR()
params = {'svr__C' : [10**i for i in range(1,8)]}
```

```
###########################Traitement de données################################
cdata = pd.read_csv('covid_numbers.csv',index_col='date',parse_dates=True)
cdata = cdata[cdata['granularite']=='pays']

for i in cdata:
    if not (cdata[i].name in ["deces"]):
        cdata.drop([i], axis=1, inplace = True)

cdata = cdata.dropna(axis=0)

print(cdata.count())
cdata

##########################Rangement de données##################################
```

Annexes

```
date = '2020-10-15'

y = cdata[:date]
X = pd.to_datetime(y.index)

size = len(X)

y = y.values.reshape(size,)
X = X.values.reshape(size,1)

X_train, y_train = X, y

model = make_pipeline(StandardScaler(), svr)
scorer = make_scorer(mean_squared_error, greater_is_better=False)
tscv = TimeSeriesSplit(n_splits=10)
grid = GridSearchCV(model, params, scorer, cv = tscv )

grid.fit(X_train, y_train)

x_prevu = pd.to_datetime(cdata[date:].index)
x_prevu = x_prevu.values.reshape(len(x_prevu),1)

cdata['deces'].plot(label="deces reels")
plt.plot(x_prevu,grid.predict(x_prevu),label="valeur predites")
plt.plot(X_train, grid.predict(X_train), label = "valeur predites")
plt.legend()
plt.show()
print(grid.best_params_)
```

```python
### Multi regresseur avec le modele SVR ###

import matplotlib.pyplot as plt
import pandas as pd
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.multioutput import RegressorChain
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import TimeSeriesSplit
```

Annexes

```
##############################################################################
tscv = TimeSeriesSplit(n_splits=15)
scorer = make_scorer(mean_squared_error, greater_is_better=False)
model = make_pipeline(StandardScaler(), RegressorChain(SVR(), order=[0,2,1]))
params = {'regressorchain__base_estimator__C': [10**i for i in range(2,8)],
          'regressorchain__base_estimator__epsilon': [0.001,0.1,0.01],
          'regressorchain__base_estimator__kernel': ['rbf']}

grid = GridSearchCV(model, params, cv = tscv, verbose = 1)

##############################################################################
cdata = pd.read_csv('covid_numbers.csv',index_col='date',parse_dates=True)

cdata = cdata[cdata['granularite']=='pays']
for i in cdata:
    if not (cdata[i].name in ["deces", "reanimation", "cas_confirmes"]):
        cdata.drop([i], axis=1, inplace = True)

cdata = cdata.dropna(axis=0)
```

Annexes

```
date = '2020-11-01'

Y = cdata['2020-03-01':date]
x = pd.to_datetime(Y.index)

x = x.values.reshape(len(x),1)

grid.fit(x,Y)

x_futur = pd.to_datetime(cdata[date:].index)
x_futur = x_futur.values.reshape(len(x_futur),1)

cdata['deces'].plot(label="deces reels")
cdata['reanimation'].plot(label="reanimation reels")
cdata['cas_confirmes'].plot(label="cas confirmés réel")

plt.yscale('log')
plt.legend()
plt.plot(x_futur,grid.predict(x_futur),label="valeur predites")
plt.plot(x, grid.predict(x),label="valeur predites")

print(grid.best_params_)
```

Annexes

```python
''''                    Predictions a l'aide du module Prophet                    '''

import matplotlib.pyplot as plt
import pandas as pd
import re
from fbprophet import Prophet
##  debug ##
def clean_dates(D):
    V = D.ds.values
    txt_d = "[0123456789]{4}-[0123456789]{2}-[0123456789]{2}"
    for i in range(len(V)):
        x = re.search(txt_d, V[i])
        if x == None:
            cdata.drop(index=i, inplace = True)
    return
## Importation des donnees ##
'''
On s'interessera que sur les donnees sur les deces pour l'instant
'''

cdata = pd.read_csv('chiffres-cles.csv',index_col='date',parse_dates=True)
cdata = cdata[cdata['granularite']=='pays']
```

Annexes

```
for i in cdata:
    if not (cdata[i].name in ["deces"]):#,"cas_confirmes"]):
        cdata.drop([i], axis=1, inplace = True)

cdata = cdata.dropna(axis=0)
cdata.reset_index(level=0, inplace=True)
cdata.columns = ['ds', 'y']
clean_dates(cdata)

### Importation du Modele ##
model = Prophet()

## Prediction ##
date = '2020-11-25' ##max 2020-12-29
d = 34 #nombre de jour entre la date donnée et la date max de cdata
cdata_less = cdata[cdata['ds'] <= date]
model.fit(cdata_less)


future_df = model.make_future_dataframe(periods=d)
prediction = model.predict(future_df)
```

Annexes

```
## Plot ##
#plot = model.plot(prediction)
pred = prediction[['ds','yhat']]

cdata = cdata.set_index('ds')
cdata = cdata.groupby(['ds']).max()
pred = pred[pred['ds'] > date]
pred = pred.set_index('ds')

date_x = cdata.index
x = [i for i in date_x if i > date]
plt.plot(date_x, cdata['y'],label="deces reels")
plt.plot(x, pred['yhat'], label = "valeures predites")
plt.xticks(date_x[::15],  rotation='vertical')
plt.margins(0.01)
plt.legend()
plt.show()
```

Annexes

```python
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.pipeline import make_pipeline
from sklearn.neural_network import MLPRegressor
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error

scorer = make_scorer(mean_squared_error, greater_is_better=False)
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import GridSearchCV

### décale un matrice de données indéxées par un date à n jours en avance
def timeshift_day(data,n):
    data.index = data.index.shift(periods = n, freq ='D')
```

Annexes

```
tscv = TimeSeriesSplit(n_splits=5)

covid = pd.read_excel('chiffre_covid_10052021.xlsx',index_col='date',parse_dates=True)

covid = covid[covid['granularite']=='pays']

print(covid.keys())
for i in covid:
    if not (covid[i].name in ["deces",'reanimation','hospitalises','nouvelles_hospitalisations']):
        covid.drop([i], axis=1, inplace = True)


covid = covid.dropna(axis=0)
print(covid.count())
timeshift_day(covid['deces'],-7)
covid=covid[7:]

full_size=covid.count()[0]
print(full_size)
subject=['hospitalises','reanimation','nouvelles_hospitalisations']
result=["deces"]
predit_jour=20
size=full_size-predit_jour
```

Annexes

```
X = covid[:size][subject]
y = covid[:size][result]
y = y.values.reshape(size,)
X_result=covid[size:][subject]

y_result=covid[size:][result]
y_result = y_result.values.reshape(full_size-size,)

params={
    'mlpregressor__max_iter': [90000],
    'mlpregressor__tol': [0.0001],
    'mlpregressor__n_iter_no_change': [2],
}

model=make_pipeline(StandardScaler(),MLPRegressor())


grid=GridSearchCV(model,param_grid=params,cv=tscv,scoring=scorer)
```

Annexes

```
print(model.get_params())
grid.fit(X,y)
print(grid.score(X,y))
print(grid.score(X_result,y_result))
print(grid.best_params_)



timeshift_day(covid['deces'],7)
covid=covid[:full_size-7]


plt.figure()
plt.plot(covid.index,covid['deces'])
plt.plot(covid.index[size-7:],grid.predict(covid[size-7:][subject]))
plt.plot(covid.index[:size-7],grid.predict(covid[:size-7][subject]))
plt.show()
```