meteo

October 15, 2018

0.1 Rapport sur le TP METEO

L'objectif de ce TP est de prédire la puissance en fonction des données météos qui nous sont fournies, et ce à l'aide d'une regression linéaire et d'un Random Forest.

On dispose d'un jeu de données qui contient les puissances associées aux dates auquelles elles ont été relevée, et un jeu de données qui contient les conditions météorologiques de ces jours associés. Le premier jeu de donnée à une périodicité de 1h par jour, tandis que le second a une périodicité de 3H. Nous allons essayer déterminer si il est préférable d'établir un modèle sur la base d'une périodicité de 3H ou si les performances sont meilleures avec une transformation du 2eme jeu de donnée sur une périodicité d'1H.

```
In [60]: import pandas as pd
         import matplotlib.pyplot as plt
         import seaborn as sns
         import numpy as np
         import datetime
         from dateutil import parser
         from sklearn.metrics import mean_squared_error
         from sklearn.model_selection import train_test_split
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.linear_model import LinearRegression
```

On récupère les jeux de données et on les visualise pour pouvoir les traiter au mieux

```
In [61]: data_conso = pd.read_csv("data/conso_train.csv",sep=";")
         data_meteo = pd.read_csv("data/meteo_train.csv",sep=";")

In [62]: data_conso.head()
```

Out[62]:

	date	puissance
0	2015-09-13T00:59:59+02:00	526.166667
1	2015-09-13T01:59:59+02:00	495.000000
2	2015-09-13T02:59:59+02:00	446.166667
3	2015-09-13T03:59:59+02:00	365.833333
4	2015-09-13T04:59:59+02:00	341.000000

```
In [63]: data_meteo.head()
```

```
Out[63]:          Date UTC  Tñ (C)  P (hPa)  HR (%)  P.rosĺe (ñC)  Visi (km)  \
         0  13/09/15 00h00     12.5   1008.7    81.0            9.3       40.0
         1  13/09/15 03h00     12.3   1006.4    83.0            9.5       40.0
         2  13/09/15 06h00     12.3   1004.7    82.0            9.3       40.0
         3  13/09/15 09h00     14.2   1002.9    80.0           10.8       40.0
         4  13/09/15 12h00     13.3   1000.8    93.0           12.2        4.0

            Vt. moy. (km/h)  Vt. raf. (km/h)  Vt. dir (ñ)  RR 3h (mm)  Neige (cm)  \
         0            9.260           18.520        140.0         0.0         NaN
         1           11.112           16.668        120.0         0.0         NaN
         2           14.816           22.224        130.0         NaN         NaN
         3           18.520           31.484        140.0         NaN         NaN
         4           18.520           38.892        140.0         4.0         NaN

            Nebul. (octats)
         0              8.0
         1              8.0
         2              7.0
         3              7.0
         4              7.0
```

On vérifie les valeurs nulles présentes dans les jeux de données.

```
In [64]: data_conso.isnull().sum()

Out[64]: date         0
         puissance    0
         dtype: int64

In [65]: print(data_meteo.isnull().sum())
         print(data_meteo.shape)

Date UTC              0
Tñ (C)               64
P (hPa)              64
HR (%)               64
P.rosĺe (ñC)         64
Visi (km)            64
Vt. moy. (km/h)      64
Vt. raf. (km/h)      67
Vt. dir (ñ)          64
RR 3h (mm)          336
Neige (cm)         1957
Nebul. (octats)     409
dtype: int64
(2928, 12)
```

On renome les colonnes qui contiennent trop de caractères spéciaux, puis on remplace les valeurs manquantes par les valeurs médianes de chaque colonnes.

```
In [66]: data_meteo.columns = ['Date', 'Temp', 'Pressure', 'Humid', 'Temp2', 'Visi',
                               'VitMoy','VitRaf', 'VitDir', 'RR', 'Neige', 'Nebul']
         median = data_meteo.median()
         median.head()

Out[66]: Temp          12.4
         Pressure    1018.4
         Humid         84.0
         Temp2          9.5
         Visi          16.0
         dtype: float64

In [67]: data_meteo['Temp'].fillna(median['Temp'], inplace=True)
         data_meteo['Pressure'].fillna(median['Pressure'], inplace=True)
         data_meteo['Humid'].fillna(median['Humid'], inplace=True)
         data_meteo['Temp2'].fillna(median['Temp2'], inplace=True)
         data_meteo['Visi'].fillna(median['Visi'], inplace=True)
         data_meteo['VitMoy'].fillna(median['VitMoy'], inplace=True)
         data_meteo['VitRaf'].fillna(median['VitRaf'], inplace=True)
         data_meteo['VitDir'].fillna(median['VitDir'], inplace=True)
         data_meteo['RR'].fillna(median['RR'], inplace=True)
         data_meteo['Nebul'].fillna(median['Nebul'], inplace=True)
         data_meteo = data_meteo.drop('Neige',1)
         print(data_meteo.isnull().sum())
         print(data_meteo.shape)

Date        0
Temp        0
Pressure    0
Humid       0
Temp2       0
Visi        0
VitMoy      0
VitRaf      0
VitDir      0
RR          0
Nebul       0
dtype: int64
(2928, 11)
```

On remarquera dans le jeu de données des puissances que les dates ont un format particulier. Il manque 1 seconde pour les arrondir sur les 2639 premières données. Il faut leur ajouter 1 seconde pour pouvoir fusionner les 2 DataFrame avec les dates qui correspondent. La méthode 'to_datetime' va calibrer la date sur le fuseau horaire UTC automatiquement.

```
In [68]: data_conso['date'] = pd.to_datetime(data_conso['date'])
         data_conso['date'] = (data_conso['date'][0:2638] +
                               pd.Timedelta(seconds=1)).append(data_conso['date'][2638:])
```

```
data_conso.columns = ['Date','Puissance']
data_conso.head()
```

Out[68]:

	Date	Puissance
0	2015-09-12 23:00:00	526.166667
1	2015-09-13 00:00:00	495.000000
2	2015-09-13 01:00:00	446.166667
3	2015-09-13 02:00:00	365.833333
4	2015-09-13 03:00:00	341.000000

```
In [69]: data_conso.shape
```

Out[69]: (8760, 2)

On sépare la date qu'on va transformer en valeur numérique pour l'utiliser comme paramètre.

```
In [70]: data_meteo['Date'] = pd.to_datetime(data_meteo['Date'], format="%d/%m/%y %Hh%M")
         data_meteo['jour'] = (data_meteo["Date"].dt.weekday).astype(float)
         data_meteo['mois'] = data_meteo["Date"].dt.month.astype(float)
         data_meteo['annee'] = data_meteo["Date"].dt.year.astype(float)
         data_meteo['heure'] = data_meteo["Date"].dt.hour.astype(float)
         data_meteo.head()
```

Out[70]:

	Date	Temp	Pressure	Humid	Temp2	Visi	VitMoy	VitRaf	\
0	2015-09-13 00:00:00	12.5	1008.7	81.0	9.3	40.0	9.260	18.520	
1	2015-09-13 03:00:00	12.3	1006.4	83.0	9.5	40.0	11.112	16.668	
2	2015-09-13 06:00:00	12.3	1004.7	82.0	9.3	40.0	14.816	22.224	
3	2015-09-13 09:00:00	14.2	1002.9	80.0	10.8	40.0	18.520	31.484	
4	2015-09-13 12:00:00	13.3	1000.8	93.0	12.2	4.0	18.520	38.892	

	VitDir	RR	Nebul	jour	mois	annee	heure
0	140.0	0.0	8.0	6.0	9.0	2015.0	0.0
1	120.0	0.0	8.0	6.0	9.0	2015.0	3.0
2	130.0	0.0	7.0	6.0	9.0	2015.0	6.0
3	140.0	0.0	7.0	6.0	9.0	2015.0	9.0
4	140.0	4.0	7.0	6.0	9.0	2015.0	12.0

Dans un premier temps, on va créer un modèle sur un jeu de données d'une périodicité de 3H, on fait donc un merge simple des deux tableaux et on obtient donc bien 2928 données, ce qui correspond au nombre de donnée du tableau contenant le moins de cellules.

On vérifie également que le tableau ne contient toujours pas de données nulles.

```
In [71]: three_hours_df = pd.merge(data_conso,data_meteo, on='Date', how='right')
         three_hours_df.head()
```

Out[71]:

	Date	Puissance	Temp	Pressure	Humid	Temp2	Visi	VitMoy	\
0	2015-09-13 00:00:00	495.000000	12.5	1008.7	81.0	9.3	40.0	9.260	
1	2015-09-13 03:00:00	341.000000	12.3	1006.4	83.0	9.5	40.0	11.112	
2	2015-09-13 06:00:00	459.166667	12.3	1004.7	82.0	9.3	40.0	14.816	

```
3 2015-09-13 09:00:00  744.666667  14.2    1002.9   80.0   10.8  40.0  18.520
4 2015-09-13 12:00:00  615.166667  13.3    1000.8   93.0   12.2   4.0  18.520

   VitRaf  VitDir   RR  Nebul  jour  mois   annee  heure
0  18.520   140.0  0.0    8.0   6.0   9.0  2015.0    0.0
1  16.668   120.0  0.0    8.0   6.0   9.0  2015.0    3.0
2  22.224   130.0  0.0    7.0   6.0   9.0  2015.0    6.0
3  31.484   140.0  0.0    7.0   6.0   9.0  2015.0    9.0
4  38.892   140.0  4.0    7.0   6.0   9.0  2015.0   12.0
```

In [72]:
```
print(three_hours_df.shape)
```

```
(2928, 16)
```

In [73]:
```
three_hours_df.isnull().sum()
```

Out[73]:
```
Date         0
Puissance    0
Temp         0
Pressure     0
Humid        0
Temp2        0
Visi         0
VitMoy       0
VitRaf       0
VitDir       0
RR           0
Nebul        0
jour         0
mois         0
annee        0
heure        0
dtype: int64
```

In [74]:
```
three_hours_df.head()
```

Out[74]:
```
                 Date   Puissance  Temp  Pressure  Humid  Temp2  Visi  VitMoy  \
0 2015-09-13 00:00:00  495.000000  12.5    1008.7   81.0    9.3  40.0   9.260
1 2015-09-13 03:00:00  341.000000  12.3    1006.4   83.0    9.5  40.0  11.112
2 2015-09-13 06:00:00  459.166667  12.3    1004.7   82.0    9.3  40.0  14.816
3 2015-09-13 09:00:00  744.666667  14.2    1002.9   80.0   10.8  40.0  18.520
4 2015-09-13 12:00:00  615.166667  13.3    1000.8   93.0   12.2   4.0  18.520

   VitRaf  VitDir   RR  Nebul  jour  mois   annee  heure
0  18.520   140.0  0.0    8.0   6.0   9.0  2015.0    0.0
1  16.668   120.0  0.0    8.0   6.0   9.0  2015.0    3.0
2  22.224   130.0  0.0    7.0   6.0   9.0  2015.0    6.0
3  31.484   140.0  0.0    7.0   6.0   9.0  2015.0    9.0
4  38.892   140.0  4.0    7.0   6.0   9.0  2015.0   12.0
```

On créer les deux modèles et on remarquera que l'utilisation d'un Random Forest est beaucoup plus efficace. Les valeurs des feuilles minimales et du nombre d'arbre utilisés sont issus de nombreux test effectués au préalables.

```
In [75]: list_test_size = [a/20 for a in list(range(20))][1:]
         scores = []
         scores2 = []

         keep = ['Temp', 'Pressure', 'Humid', 'Temp2', 'Visi', 'VitMoy',
                       'VitRaf', 'VitDir', 'RR', 'Nebul', 'jour','mois','annee','heure']

         linreg = LinearRegression()
```

```
In [76]: for ts in list_test_size:
             X_train, X_test, y_train, y_test = train_test_split(three_hours_df[keep],
                                                                 three_hours_df["Puissance"],
                                                                 test_size=0.3,
                                                                 random_state=0)

             linreg.fit(X_train, y_train)
             y_pred = linreg.predict(X_test)
             scores.append(linreg.score(X_test, y_test))

         print('Accuracy LinReg: {:.2f}%'.format(np.array(scores).mean()*100))
```

```
Accuracy LinReg: 67.87%
```

```
In [77]: X_train, X_test, y_train, y_test = train_test_split(three_hours_df[keep],
                                                            three_hours_df["Puissance"],
                                                            test_size=0.05,
                                                            random_state=0)

         rf = RandomForestRegressor(n_estimators = 150,
                                    min_samples_leaf = 8)
         rf.fit(X_train, y_train)
         y_pred = rf.predict(X_test)
         print('Accuracy RF ({},{}): {:.2f}%'.format(150,
                                                     8,
                                                     rf.score(X_test, y_test)*100))
```

```
Accuracy RF (150,8): 91.63%
```

Nous allons maintenant créer le même tableau, mais sur une périodicité de 1H. Pour ce faire, on crée les valeurs manquantes entre les 3 heures. Les données que nous avons ajoutés sont des copies des données qui les précèdent sur les 3h précédentes.

```
In [78]: data_meteo_bis = data_meteo.copy()
         data_meteo_bis.head()
```

Out[78]:

	Date	Temp	Pressure	Humid	Temp2	Visi	VitMoy	VitRaf
0	2015-09-13 00:00:00	12.5	1008.7	81.0	9.3	40.0	9.260	18.520
1	2015-09-13 03:00:00	12.3	1006.4	83.0	9.5	40.0	11.112	16.668
2	2015-09-13 06:00:00	12.3	1004.7	82.0	9.3	40.0	14.816	22.224
3	2015-09-13 09:00:00	14.2	1002.9	80.0	10.8	40.0	18.520	31.484
4	2015-09-13 12:00:00	13.3	1000.8	93.0	12.2	4.0	18.520	38.892

	VitDir	RR	Nebul	jour	mois	annee	heure
0	140.0	0.0	8.0	6.0	9.0	2015.0	0.0
1	120.0	0.0	8.0	6.0	9.0	2015.0	3.0
2	130.0	0.0	7.0	6.0	9.0	2015.0	6.0
3	140.0	0.0	7.0	6.0	9.0	2015.0	9.0
4	140.0	4.0	7.0	6.0	9.0	2015.0	12.0

In [79]:
```
data_meteo_bis = data_meteo_bis.set_index('Date')[['Temp', 'Pressure',
                                                   'Humid', 'Temp2',
                                                   'Visi', 'VitMoy',
                                                   'VitRaf', 'VitDir',
                                                   'RR','Nebul']].resample("1H").mean
```

In [80]:
```
data_meteo_bis = data_meteo_bis.fillna(method='pad')
data_meteo_bis.head()
```

Out[80]:

Date	Temp	Pressure	Humid	Temp2	Visi	VitMoy	VitRaf
2015-09-13 00:00:00	12.5	1008.7	81.0	9.3	40.0	9.260	18.520
2015-09-13 01:00:00	12.5	1008.7	81.0	9.3	40.0	9.260	18.520
2015-09-13 02:00:00	12.5	1008.7	81.0	9.3	40.0	9.260	18.520
2015-09-13 03:00:00	12.3	1006.4	83.0	9.5	40.0	11.112	16.668
2015-09-13 04:00:00	12.3	1006.4	83.0	9.5	40.0	11.112	16.668

Date	VitDir	RR	Nebul
2015-09-13 00:00:00	140.0	0.0	8.0
2015-09-13 01:00:00	140.0	0.0	8.0
2015-09-13 02:00:00	140.0	0.0	8.0
2015-09-13 03:00:00	120.0	0.0	8.0
2015-09-13 04:00:00	120.0	0.0	8.0

In [81]:
```
data_meteo_bis['Date_bis'] = data_meteo_bis.index
data_meteo_bis['Date_bis'] = pd.to_datetime(data_meteo_bis['Date_bis'],
                                            format="%d/%m/%y %Hh%M")
data_meteo_bis['jour'] = (data_meteo_bis["Date_bis"].dt.weekday).astype(float)
data_meteo_bis['mois'] = data_meteo_bis["Date_bis"].dt.month.astype(float)
data_meteo_bis['annee'] = data_meteo_bis["Date_bis"].dt.year.astype(float)
data_meteo_bis['heure'] = data_meteo_bis["Date_bis"].dt.hour.astype(float)
data_meteo_bis.head()
```

Out[81]:

Date	Temp	Pressure	Humid	Temp2	Visi	VitMoy	VitRaf

```
2015-09-13 00:00:00  12.5    1008.7   81.0    9.3  40.0   9.260  18.520
2015-09-13 01:00:00  12.5    1008.7   81.0    9.3  40.0   9.260  18.520
2015-09-13 02:00:00  12.5    1008.7   81.0    9.3  40.0   9.260  18.520
2015-09-13 03:00:00  12.3    1006.4   83.0    9.5  40.0  11.112  16.668
2015-09-13 04:00:00  12.3    1006.4   83.0    9.5  40.0  11.112  16.668

                     VitDir   RR  Nebul           Date_bis  jour  mois  \
Date
2015-09-13 00:00:00   140.0  0.0    8.0 2015-09-13 00:00:00   6.0   9.0
2015-09-13 01:00:00   140.0  0.0    8.0 2015-09-13 01:00:00   6.0   9.0
2015-09-13 02:00:00   140.0  0.0    8.0 2015-09-13 02:00:00   6.0   9.0
2015-09-13 03:00:00   120.0  0.0    8.0 2015-09-13 03:00:00   6.0   9.0
2015-09-13 04:00:00   120.0  0.0    8.0 2015-09-13 04:00:00   6.0   9.0

                      annee  heure
Date
2015-09-13 00:00:00  2015.0    0.0
2015-09-13 01:00:00  2015.0    1.0
2015-09-13 02:00:00  2015.0    2.0
2015-09-13 03:00:00  2015.0    3.0
2015-09-13 04:00:00  2015.0    4.0
```

```
In [82]: one_hour_df = pd.merge(data_conso,data_meteo_bis, on='Date', how='right')
         one_hour_df.head()
```

```
Out[82]:                 Date   Puissance  Temp  Pressure  Humid  Temp2  Visi  VitMoy  \
         0 2015-09-13 00:00:00  495.000000  12.5    1008.7   81.0    9.3  40.0   9.260
         1 2015-09-13 01:00:00  446.166667  12.5    1008.7   81.0    9.3  40.0   9.260
         2 2015-09-13 02:00:00  365.833333  12.5    1008.7   81.0    9.3  40.0   9.260
         3 2015-09-13 03:00:00  341.000000  12.3    1006.4   83.0    9.5  40.0  11.112
         4 2015-09-13 04:00:00  352.333333  12.3    1006.4   83.0    9.5  40.0  11.112

            VitRaf  VitDir   RR  Nebul           Date_bis  jour  mois   annee  heure
         0  18.520   140.0  0.0    8.0 2015-09-13 00:00:00   6.0   9.0  2015.0    0.0
         1  18.520   140.0  0.0    8.0 2015-09-13 01:00:00   6.0   9.0  2015.0    1.0
         2  18.520   140.0  0.0    8.0 2015-09-13 02:00:00   6.0   9.0  2015.0    2.0
         3  16.668   120.0  0.0    8.0 2015-09-13 03:00:00   6.0   9.0  2015.0    3.0
         4  16.668   120.0  0.0    8.0 2015-09-13 04:00:00   6.0   9.0  2015.0    4.0
```

En vérifiant qu'il n'y a pas de valeur nulles on se rend compte qu'il en existe, on utilise la méthode de la médiane pour les combler

```
In [83]: one_hour_df.isnull().sum()
```

```
Out[83]: Date          0
         Puissance    26
         Temp          0
         Pressure      0
         Humid         0
```

```
Temp2         0
Visi          0
VitMoy        0
VitRaf        0
VitDir        0
RR            0
Nebul         0
Date_bis      0
jour          0
mois          0
annee         0
heure         0
dtype: int64
```

```
In [84]: median3=one_hour_df.median()
         one_hour_df['Puissance'] = one_hour_df['Puissance'].fillna(median3['Puissance'])
         one_hour_df.isnull().sum()
```

```
Out[84]: Date         0
         Puissance    0
         Temp         0
         Pressure     0
         Humid        0
         Temp2        0
         Visi         0
         VitMoy       0
         VitRaf       0
         VitDir       0
         RR           0
         Nebul        0
         Date_bis     0
         jour         0
         mois         0
         annee        0
         heure        0
         dtype: int64
```

On crée, comme précédemment, les deux modèles. On remarquera que le Random Forest est toujours plus performant, et également qu'il est préférable d'utiliser les Data avec une périodicité d'1H plutot que celles avec une périodicité de 3H.

```
In [85]: for ts in list_test_size:
             X_train, X_test, y_train, y_test = train_test_split(one_hour_df[keep],
                                                                 one_hour_df["Puissance"],
                                                                 test_size=0.3,
                                                                 random_state=0)

             linreg.fit(X_train, y_train)
             y_pred = linreg.predict(X_test)
```

```
            scores2.append(linreg.score(X_test, y_test))

        print('Accuracy LinReg: {:.2f}%'.format(np.array(scores2).mean()*100))
```

```
Accuracy LinReg: 65.59%
```

```
In [86]: X_train, X_test, y_train, y_test = train_test_split(one_hour_df[keep],
                                                            one_hour_df["Puissance"],
                                                            test_size=0.05,
                                                            random_state=0)

         rf = RandomForestRegressor(n_estimators = 150,
                                    min_samples_leaf = 8)
         rf.fit(X_train, y_train)
         y_pred = rf.predict(X_test)
         print('Accuracy RF ({},{}): {:.2f}%'.format(150,
                                                     8,
                                                     rf.score(X_test, y_test)*100))
```

```
Accuracy RF (150,8): 94.11%
```