Models using full features scope

September 2, 2018

1 Firts attempts & Base Model

```
In [1]: from pathlib import Path
        import pandas as pd
        import numpy as np
        from datetime import datetime
        import time
        import matplotlib.pyplot as plt
        %matplotlib inline
        #%pylab inline
        import itertools
        import pickle
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.linear_model import LogisticRegression
        from sklearn.model_selection import cross_val_score, GridSearchCV
        from sklearn.decomposition import PCA
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score
        from sklearn.preprocessing import LabelEncoder
        from sklearn.preprocessing import LabelBinarizer
        from sklearn.preprocessing import OneHotEncoder
        from sklearn.svm import SVC
        from sklearn.model_selection import StratifiedKFold
        from sklearn.feature_selection import RFECV, RFE

In [2]: path_project = Path.home() / Path('Google Drive/Felix')
        path_data = path_project / Path("data")
        path_dump = path_project / Path("dump")

In [3]: # loading cdv data
        file = path_data / Path("felix.csv")
        with Path.open(file, 'rb') as fp:
            cdv = pd.read_csv(fp,  encoding='cp1252',low_memory=False, index_col = 0)

In [4]: # loadind cdv data without format
        file = path_data / Path("felix_ssfmt.csv")
```

```
        with Path.open(file, 'rb') as fp:
            cdv_ssfmt = pd.read_csv(fp,  encoding='cp1252',low_memory=False, index_col = 0)
```

In [5]:
```
        # loading MergeCommunesEnvi data
        file = path_data / Path("MergeCommunesEnvi.csv")
        with Path.open(file, 'rb') as fp:
            MergeCommunesEnvi = pd.read_csv(fp,  encoding='cp1252',low_memory=False, sep=';', in
```

1.1 1) Feature engineering

In [6]:
```
        filename = path_dump / Path("dict_var_groups.sav")
        with open(filename, 'rb') as fp:
             dict_var_groups = pickle.load(fp)

        scope_2015_var = dict_var_groups['scope_2015_var']
        scope_2016_var = dict_var_groups['scope_2016_var']
        scope_2017_var = dict_var_groups['scope_2017_var']
        scope_2018_var = dict_var_groups['scope_2018_var']
        scope_2015_2018_var = dict_var_groups['scope_2015_2018_var']
        scope_2016_2018_var = dict_var_groups['scope_2016_2018_var']
        scope_2017_2018_var = dict_var_groups['scope_2017_2018_var']
        pred_var = dict_var_groups['pred_var']
        com_var = dict_var_groups['com_var']
        tech_var = dict_var_groups['tech_var']
        text_var = dict_var_groups['text_var']
        bizz_var = dict_var_groups['bizz_var']
        cat_var = dict_var_groups['cat_var']
        cat_max9_var = dict_var_groups['cat_max9_var']
        cat_min10_var = dict_var_groups['cat_min10_var']
        quant_var = dict_var_groups['quant_var']

        indiv_semi_act_var = dict_var_groups["indiv_semi_act_var"]
        indiv_act_var = dict_var_groups["indiv_act_var"]
        admin_semi_act_var = dict_var_groups["admin_semi_act_var"]
        admin_act_var = dict_var_groups["admin_act_var"]
        commune_var = dict_var_groups["commune_var"]

        scope_2015_ext_var = dict_var_groups["scope_2015_ext_var"]
        scope_2016_ext_var = dict_var_groups["scope_2016_ext_var"]
        scope_2017_ext_var = dict_var_groups["scope_2017_ext_var"]
        scope_2018_ext_var = dict_var_groups["scope_2018_ext_var"]
        scope_2015_2018_ext_var = dict_var_groups["scope_2015_2018_ext_var"]
        scope_2016_2018_ext_var = dict_var_groups["scope_2016_2018_ext_var"]
        scope_2017_2018_ext_var = dict_var_groups["scope_2017_2018_ext_var"]
```

In [7]:
```
        exclusion = com_var | tech_var | bizz_var | text_var
        scope_kept = scope_2015_2018_ext_var - exclusion
        print(f"{len(scope_kept)} are kept ")
```

```
        cat_var_kept = cat_max9_var & scope_kept
        scope_quant_var = (quant_var & scope_kept)
        quant_null = np.sum(MergeCommunesEnvi.loc[:,scope_quant_var].isnull())
        quant_var_kept = set(quant_null[quant_null < 200].index)
        print(f"out of the {len(scope_quant_var)} quantitatices variables:")
        print(f"{len(quant_var_kept)} have less than 200 missing values and are kept")
```

```
464 are kept
out of the 290 quantitatices variables:
263 have less than 200 missing values and are kept
```

```
In [8]: scope = cat_var_kept | quant_var_kept
        df = MergeCommunesEnvi.loc[:,scope]
        # ...
        df.loc[:,scope_2015_2018_var & scope] = cdv_ssfmt.loc[:,scope_2015_2018_var & scope]
        df.loc[:,cat_var_kept - {"HEUREUX"}] = cdv.loc[:,cat_var_kept - {"HEUREUX"}]
```

```
In [9]: print(f"Number of variable kept {df.shape[1]}")
```

```
Number of variable kept 419
```

1.1.1 Encoding 'HEUREUX'

```
In [10]: # reducing problem to a 2 class classification problem
         df["HEUREUX_CLF"] = 0
         df.loc[df["HEUREUX"]==4, "HEUREUX_CLF"] = 1
         df.loc[df["HEUREUX"]==3, "HEUREUX_CLF"] = 1
         df.loc[df["HEUREUX"]==5, "HEUREUX_CLF"] = None
```

```
In [11]: # Modelisation as a regression problem
         df["HEUREUX_REG"] = df["HEUREUX"]
         df.loc[df["HEUREUX"]==5, "HEUREUX_REG"] = None
```

1.1.2 Encoding categorial variables

```
In [12]: p = df.shape[1]
         print(f"{p} columns")
         print(f"out of which {len(cat_var_kept)-1} are corresponding to categorial features")
```

```
421 columns
out of which 155 are corresponding to categorial features
```

```
In [13]: df = pd.get_dummies(df,
                              columns=cat_var_kept - {"HEUREUX"},
                              dummy_na = True,
                              drop_first=1)
```

```
In [14]: q = df.shape[1]
         print(f"{q} columns after encoding")
         print(f"{len(cat_var_kept)-1} variables where re-encoded in {len(cat_var_kept)-1+q-p}")
```

```
838 columns after encoding
155 variables where re-encoded in 572
```

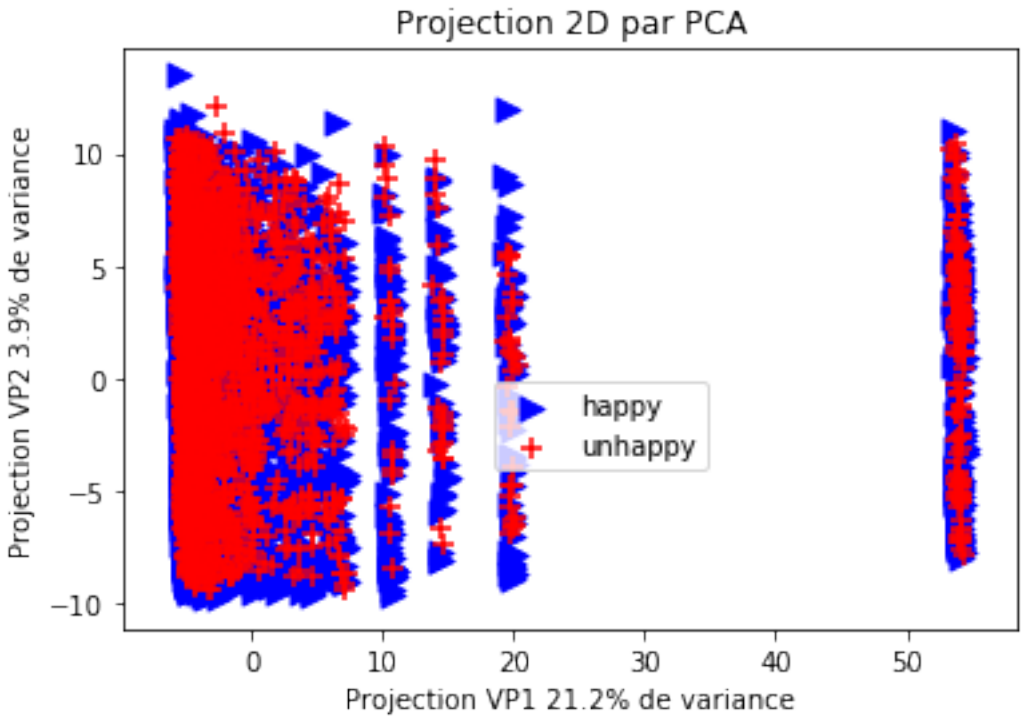
```
In [15]: def get_related_features(variable):
             '''return all columns in global variable df
             starting by variable'''
             if isinstance(variable, str):
                 scope = {variable}
             else:
                 scope = set(variable)
             features = set()
             for element in scope:
                 features = features  | {c for c in df.columns if len(c) > len(element)
                                        and c[0:len(element)] == element
                                        and c[len(element)]=='_'}
             return features
```

1.2 2) Dataset construction and visualisation

```
In [16]: # subseting dataframe
         features = df.columns.drop(['HEUREUX',"HEUREUX_CLF", "HEUREUX_REG"])
         pred = "HEUREUX_CLF"

         # treating remaining missing values
         df_tmp = df.loc[:,set(features) | {pred} ].dropna()

         X = df_tmp.loc[:,features]
         y = df_tmp[pred]

         X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                             test_size=0.2,
                                                             random_state=42)

         scaler = StandardScaler().fit(X_train)
         X_train = scaler.transform(X_train)
         X_test = scaler.transform(X_test)

         print(f"Number exemple: {y.shape[0]}\n- training set: \
         {y_train.shape[0]}\n- test set: {y_test.shape[0]}")
         print(f"Number of features: p={X_train.shape[1]}")
         print(f"Number of class: {len(np.unique(y))}")
         for c in np.unique(y):
             print(f"class {c:0.0f} : {100*np.sum(y==c)/len(y):0.1f}%")
```

```
Number exemple: 10445
- training set: 8356
- test set: 2089
Number of features: p=835
Number of class: 2
class 0 : 35.1%
class 1 : 64.9%
```

```
In [17]: # Reduction dim PCA
         pca = PCA(n_components=2)
         pca.fit(X_train)
         X_r = pca.transform(X_train)
         happy = (y_train==1)
         unhappy = (y_train==0)
         plt.scatter(X_r[happy,0], X_r[happy,1],
                     s=80, c='blue',marker=">", label="happy")
         plt.scatter(X_r[unhappy,0], X_r[unhappy,1],
                     s=80, c='red',marker='+', label="unhappy")
         plt.ylabel(f'Projection VP2 \
         {pca.explained_variance_ratio_[1]*100:0.1f}% de variance')
         plt.xlabel(f'Projection VP1 \
         {pca.explained_variance_ratio_[0]*100:0.1f}% de variance')
         plt.title("Projection 2D par PCA")
         plt.legend(bbox_to_anchor=(0.4, 0.25))
         plt.show()
```



1.3 3) Baseline model

Logistic regression with **4 features** to predict a **2 class** variable

1.3.1 a) Training set and test set preparation

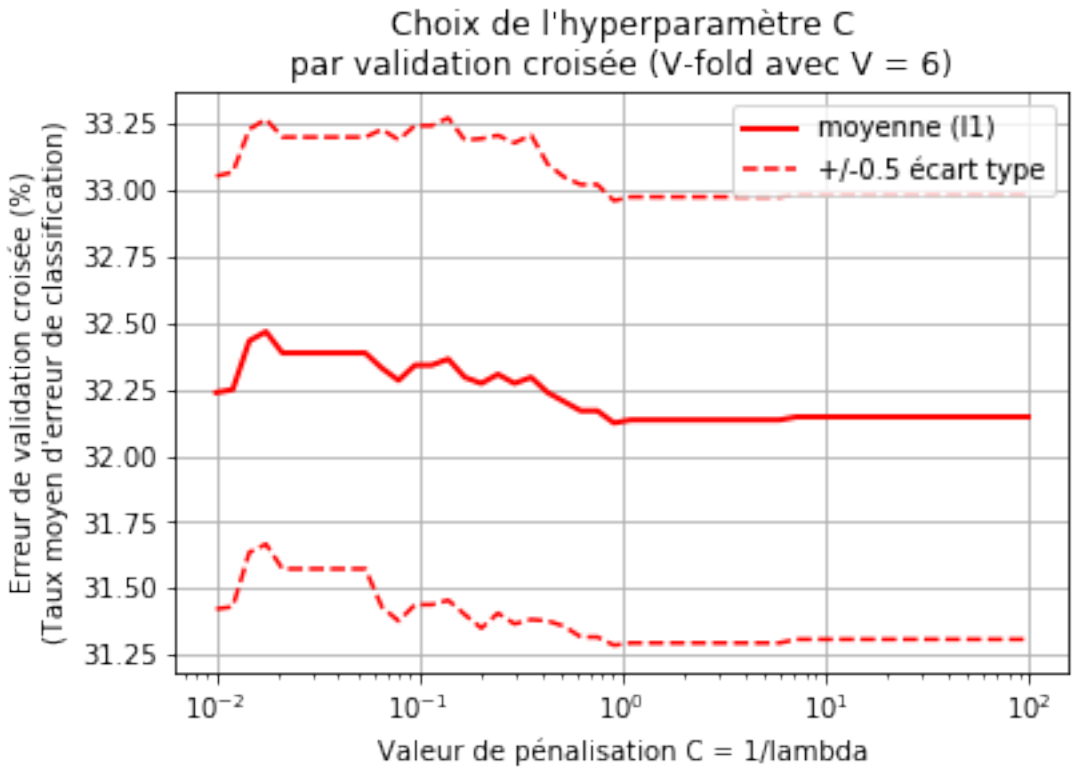
```
In [18]: # subseting dataframe
         features = {'ETATSAN','NOT_AMIS','DEPLOG','NOT_FAMI'}
         pred = "HEUREUX_CLF"

         # treating remaining missing values
         df_tmp = df.loc[:,set(features) | {pred} ].dropna()

         X = df_tmp.loc[:,features]
         y = df_tmp[pred]


         X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                             test_size=0.2,
                                                             random_state=42)

         print(f"Number exemple: {y.shape[0]}\n- training set: \
         {y_train.shape[0]}\n- test set: {y_test.shape[0]}")
         print(f"Number of features: p={X_train.shape[1]}")
         print(f"Number of class: {len(np.unique(y))}")
         for c in np.unique(y):
             print(f"class {c:0.0f} : {100*np.sum(y==c)/len(y):0.1f}%")
```

```
Number exemple: 10915
- training set: 8732
- test set: 2183
Number of features: p=4
Number of class: 2
class 0 : 34.9%
class 1 : 65.1%
```

1.3.2 b) Hyperparameters tuning

```
In [19]: nb_value = 50 # Nombre de valeurs testées pour l'hyperparamètre
         mean_score_l1 = np.zeros(nb_value)
         C_log = np.logspace(-2,2,nb_value)
         cv = 6 # V-fold, nombre de fold

         mean_score_l1 = np.empty(nb_value)
```

```
std_scores_l1 = np.empty(nb_value)

np.random.seed(seed=42)

startTime = time.time()

for i, C in enumerate(C_log):
    clf = LogisticRegression(C=C, penalty='l1',
                             tol=0.01, random_state=42,
                             class_weight='balanced')
    mean_score_l1[i] = 100*np.mean(1-cross_val_score(clf,
                                                     X_train,
                                                     y_train,
                                                     cv=cv,
                                                     scoring='accuracy'))
    std_scores_l1[i] = 100*np.std(1-cross_val_score(clf,
                                                    X_train,
                                                    y_train,
                                                    cv=cv,
                                                    scoring='accuracy'))



plt.figure()
plt.semilogx(C_log,mean_score_l1[:],'r',linewidth=2,label='moyenne (l1)')
plt.semilogx(C_log,mean_score_l1[:]-0.5*std_scores_l1[:],
             'r--', label=u'+/-0.5 écart type')
plt.semilogx(C_log,mean_score_l1[:]+0.5*std_scores_l1[:],'r--')


plt.xlabel("Valeur de pénalisation C = 1/lambda")
plt.ylabel(u"Erreur de validation croisée (%)\n(Taux moyen d'erreur de classification)"
plt.title(u"Choix de l'hyperparamètre C\npar validation croisée \
(V-fold avec V = %s)" % (cv))
plt.legend(bbox_to_anchor=(1, 1))
plt.grid()
plt.show()
print("Détermination des paramètres optimaux en %0.1f s" % (time.time() - startTime))
print("Pénalisation l1, valeur optimale : C = %0.2f" % (C_log[np.argmin(mean_score_l1)]
```




```
Détermination des paramètres optimaux en 16.2 s
Pénalisation l1, valeur optimale : C = 0.91
```

1.3.3 c) Model training and evaluation

```
In [20]: # Learning on full training set with optimals hyperparameters
         # and score evaluation on test set
         clf = LogisticRegression(C=C_log[np.argmin(mean_score_l1)],
                                  penalty='l1',
                                  tol=0.01,
                                  random_state=42,
                                  class_weight='balanced')
         clf.fit(X_train, y_train)
         y_test_pred = clf.predict(X_test)
         accuracy = clf.score(X_test, y_test)
         print(f"Model score\n- Accuracy : {accuracy*100:0.1f} %")
         f1 = f1_score(y_test, y_test_pred)
         p = precision_score(y_test, y_test_pred)
         r = recall_score(y_test, y_test_pred)
         print(f"- Precision : {p*100:0.1f} % (Happy # positive class)")
         print(f"- Recall : {r*100:0.1f} %")
         print(f"- F1 score : {f1*100:0.1f} %")
```

```
Model score
- Accuracy : 66.6 %
- Precision : 75.9 % (Happy # positive class)
- Recall : 71.5 %
- F1 score : 73.6 %
```

1.4 4) Model selection (2 class)

1.4.1 a) Finding optimal set of features by recursive elimination using "Lasso" (RFECV)

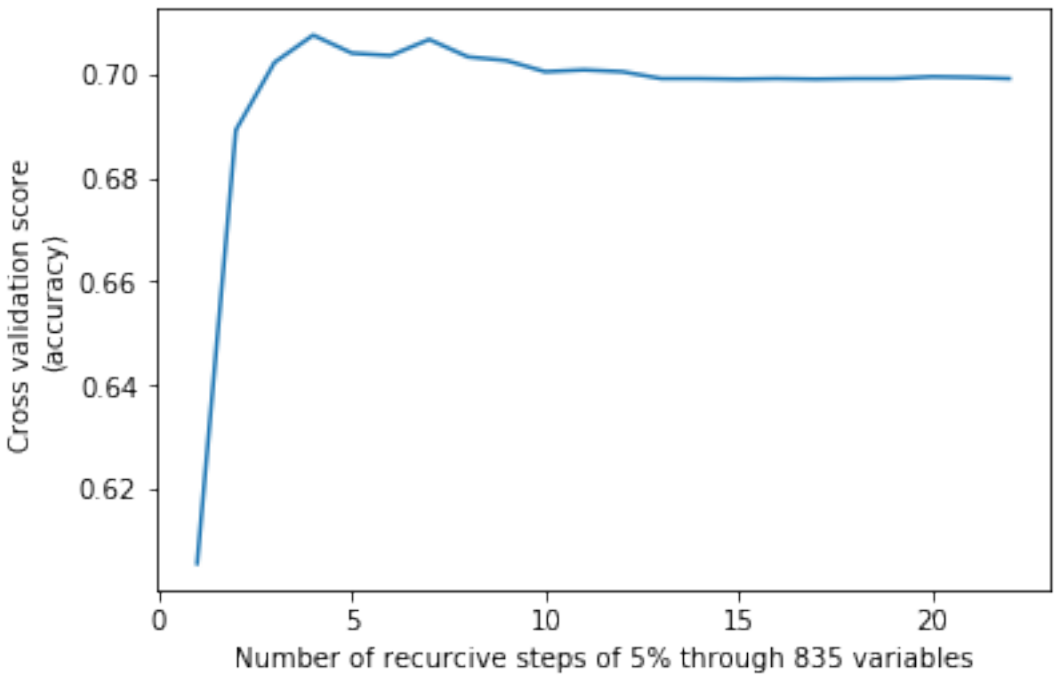
```
In [21]: # subseting dataframe
         features = df.columns.drop(['HEUREUX',"HEUREUX_CLF", "HEUREUX_REG"])
         pred = "HEUREUX_CLF"

         # treating remaining missing values
         print(f"{df.shape[0]} exemples before droping na")
         df_tmp = df.loc[:,set(features) | {pred} ].dropna()

         X = df_tmp.loc[:,features]
         y = df_tmp[pred]

         X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                             test_size=0.2,
                                                             random_state=42
                                                            )

         scaler = StandardScaler().fit(X_train)
         X_train = scaler.transform(X_train)
         X_test = scaler.transform(X_test)

         print(f"Number exemple: {y.shape[0]}\n- training set: \
         {y_train.shape[0]}\n- test set: {y_test.shape[0]}")
         print(f"Number of features: p={X_train.shape[1]}")
         print(f"Number of class: {len(np.unique(y))}")
         for c in np.unique(y):
             print(f"class {c:0.0f} : {100*np.sum(y==c)/len(y):0.1f}%")
```

```
11131 exemples before droping na
Number exemple: 10445
- training set: 8356
- test set: 2089
Number of features: p=835
Number of class: 2
class 0 : 35.1%
class 1 : 64.9%
```

```
In [22]: startTime = time.time()
```

```
scoring='accuracy'
step = 0.05

clf = LogisticRegression(C=1,
                         penalty='l1',
                         class_weight='balanced',
                         random_state=42)

rfecv = RFECV(estimator=clf, step=step, cv=StratifiedKFold(2),
              scoring=scoring)

rfecv.fit(X_train, y_train)

print("Optimal number of features : %d" % rfecv.n_features_)

# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel(f"Number of recurcive steps of {100*step:0.0f}% through {X_train.shape[1]} v
plt.ylabel(f"Cross validation score \n({scoring})")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
print(f"Détermination des features optimales en %0.1f s" % (time.time() - startTime))
```

```
Optimal number of features : 97
```



```
Détermination des features optimales en 1517.5 s
```

1.4.2 b) Using selected features to fit various models

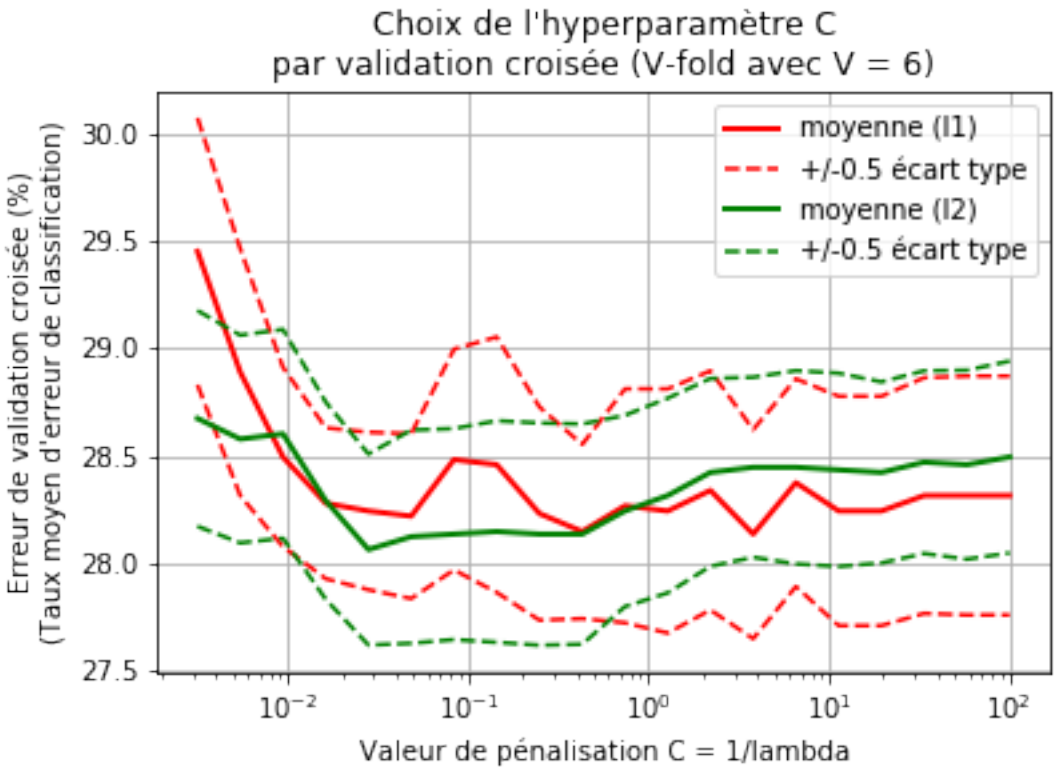
```
In [23]: lasso_mask = rfecv.support_.copy()
         X_train = X_train[:,lasso_mask]
         X_test = X_test[:,lasso_mask]
         print(f"Number of features: p={X_train.shape[1]}")
```

```
Number of features: p=97
```

Logistic regresion using L1 and L2

```
In [24]: nb_value = 20 # Nombre de valeurs testées pour l'hyperparamètre
         mean_score_l1 = np.zeros(nb_value)
         mean_score_l2 = np.zeros(nb_value)
         C_log = np.logspace(-2.5,2,nb_value)
         cv = 6 # V-fold, nombre de fold

         mean_score_l1 = np.empty(nb_value)
         std_scores_l1 = np.empty(nb_value)

         mean_score_l2 = np.empty(nb_value)
         std_scores_l2 = np.empty(nb_value)

         np.random.seed(seed=42)

         startTime = time.time()

         for i, C in enumerate(C_log):
             clf = LogisticRegression(C=C, penalty='l1',
                                      tol=0.01, random_state=42,
                                      class_weight='balanced')
             mean_score_l1[i] = 100*np.mean(1-cross_val_score(clf,
                                                              X_train,
                                                              y_train,
                                                              cv=cv,
                                                              scoring='accuracy'))
             std_scores_l1[i] = 100*np.std(1-cross_val_score(clf,
                                                             X_train,
                                                             y_train,
                                                             cv=cv,
                                                             scoring='accuracy'))
```

```
for i, C in enumerate(C_log):
    clf = LogisticRegression(C=C, penalty='l2', tol=0.01, random_state=42, class_weight
    mean_score_l2[i] = 100*np.mean(1-cross_val_score(clf,
                                                     X_train,
                                                     y_train,
                                                     cv=cv,
                                                     scoring='accuracy'))
    std_scores_l2[i] = 100*np.std(1-cross_val_score(clf,
                                                    X_train,
                                                    y_train,
                                                    cv=cv,
                                                    scoring='accuracy'))

plt.figure()
plt.semilogx(C_log,mean_score_l1[:],'r',linewidth=2,label='moyenne (l1)')
plt.semilogx(C_log,mean_score_l1[:]-0.5*std_scores_l1[:],
             'r--', label=u'+/-0.5 écart type')
plt.semilogx(C_log,mean_score_l1[:]+0.5*std_scores_l1[:],'r--')

plt.semilogx(C_log,mean_score_l2[:],'g',linewidth=2,label='moyenne (l2)')
plt.semilogx(C_log,mean_score_l2[:]-0.5*std_scores_l2[:], 'g--', label=u'+/-0.5 écart t
plt.semilogx(C_log,mean_score_l2[:]+0.5*std_scores_l2[:],'g--')

plt.xlabel("Valeur de pénalisation C = 1/lambda")
plt.ylabel(u"Erreur de validation croisée (%)\n(Taux moyen d'erreur de classification)"
plt.title(u"Choix de l'hyperparamètre C\npar validation croisée \
(V-fold avec V = %s)" % (cv))
plt.legend(bbox_to_anchor=(1, 1))
plt.grid()
plt.show()
print("Détermination des paramètres optimaux en %0.1f s" % (time.time() - startTime))
print("Pénalisation l1, valeur optimale : C = %0.2f" % (C_log[np.argmin(mean_score_l1)]
print("Pénalisation l2, valeur optimale : C = %0.2f" % (C_log[np.argmin(mean_score_l2)]
```




```
Détermination des paramètres optimaux en 181.8 s
Pénalisation l1, valeur optimale : C = 3.79
Pénalisation l2, valeur optimale : C = 0.03
```

```
In [25]: # Learning on full training set with optimals hyperparameters
         # and score evaluation on test set
         clf = LogisticRegression(C=C_log[np.argmin(mean_score_l1)],
                                  penalty='l1',
                                  random_state=42,
                                  class_weight='balanced')
         clf.fit(X_train, y_train)
         y_test_pred = clf.predict(X_test)
         accuracy = clf.score(X_test, y_test)
         print(f"Model score\n- Accuracy : {accuracy*100:0.1f} %")
         f1 = f1_score(y_test, y_test_pred)
         p = precision_score(y_test, y_test_pred)
         r = recall_score(y_test, y_test_pred)
         print(f"- Precision : {p*100:0.1f} % (Happy # positive class)")
         print(f"- Recall : {r*100:0.1f} %")
         print(f"- F1 score : {f1*100:0.1f} %")
```

```
Model score
- Accuracy : 70.6 %
```

```
- Precision : 80.5 % (Happy # positive class)
- Recall : 71.9 %
- F1 score : 76.0 %
```

```
In [26]: # Learning on full training set with optimals hyperparameters
         # and score evaluation on test set
         clf = LogisticRegression(C=C_log[np.argmin(mean_score_l2)],
                                  penalty='l2',
                                  random_state=42,
                                  class_weight='balanced')
         clf.fit(X_train, y_train)
         y_test_pred = clf.predict(X_test)
         accuracy = clf.score(X_test, y_test)
         print(f"Model score\n- Accuracy : {accuracy*100:0.1f} %")
         f1 = f1_score(y_test, y_test_pred)
         p = precision_score(y_test, y_test_pred)
         r = recall_score(y_test, y_test_pred)
         print(f"- Precision : {p*100:0.1f} % (Happy # positive class)")
         print(f"- Recall : {r*100:0.1f} %")
         print(f"- F1 score : {f1*100:0.1f} %")
```

```
Model score
- Accuracy : 71.2 %
- Precision : 81.1 % (Happy # positive class)
- Recall : 72.4 %
- F1 score : 76.5 %
```

Random Forest

```
In [27]: startTime = time.time()
         n_estimators_range = [16,32,64,128]
         max_depth_range = [2,4,8,16,32,64,128,256]
         param_grid = dict(n_estimators=n_estimators_range, max_depth = max_depth_range)

         params = {'max_features' :'sqrt', 'random_state' : 32,
                   'min_samples_split' : 2, 'class_weight' : 'balanced'}
         clf = RandomForestClassifier(**params)

         grid = GridSearchCV(clf, scoring='accuracy', param_grid=param_grid)
         grid.fit(X_train, y_train)
         print(f"Determination of optimal hyperparameters in {time.time() - startTime:0.1f} s")
         print(f"Optimal values are {grid.best_params_} \n\
         Accuracy Score of cross valdation {100*grid.best_score_:0.2f}%")

         # Learning on full training set with optimals hyperparameters and score on test set
         params = {'max_features' :'sqrt', 'random_state' : 32,
                   'min_samples_split' : 2, 'class_weight' : 'balanced',
```

```
                'n_estimators' : grid.best_params_['n_estimators'],
                'max_depth' : grid.best_params_['max_depth']}
        clf = RandomForestClassifier(**params).fit(X_train, y_train)
        clf.fit(X_train, y_train)
        y_test_pred = clf.predict(X_test)

        print(f"Random Forest, p={X_train.shape[1]}")
        accuracy = clf.score(X_test, y_test)
        f1 = f1_score(y_test, y_test_pred)
        p = precision_score(y_test, y_test_pred)
        r = recall_score(y_test, y_test_pred)
        print(f"Model score\n- Accuracy : {accuracy*100:0.1f} %")
        print(f"- Precision : {p*100:0.1f} % (Happy # positive class)")
        print(f"- Recall : {r*100:0.1f} %")
        print(f"- F1 score : {f1*100:0.1f} %")
```

```
Determination of optimal hyperparameters in 69.1 s
Optimal values are {'max_depth': 16, 'n_estimators': 128}
Accuracy Score of cross valdation 73.55%
Random Forest, p=97
Model score
- Accuracy : 72.5 %
- Precision : 74.7 % (Happy # positive class)
- Recall : 86.8 %
- F1 score : 80.3 %
```

1.4.3 b) Finding optimal set of features of given size by recursive elimination using "Lasso" (RFE)

In [28]:
```
# subseting dataframe
features = df.columns.drop(['HEUREUX',"HEUREUX_CLF", "HEUREUX_REG"])
pred = "HEUREUX_CLF"

# treating remaining missing values
print(f"{df.shape[0]} exemples before droping na")
df_tmp = df.loc[:,set(features) | {pred} ].dropna()

X = df_tmp.loc[:,features]
y = df_tmp[pred]

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42
                                                   )

scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
```

```
        X_test = scaler.transform(X_test)

        print(f"Number exemple: {y.shape[0]}\n- training set: \
        {y_train.shape[0]}\n- test set: {y_test.shape[0]}")
        print(f"Number of features: p={X_train.shape[1]}")
        print(f"Number of class: {len(np.unique(y))}")
        for c in np.unique(y):
            print(f"class {c:0.0f} : {100*np.sum(y==c)/len(y):0.1f}%")
```

```
11131 exemples before droping na
Number exemple: 10445
- training set: 8356
- test set: 2089
Number of features: p=835
Number of class: 2
class 0 : 35.1%
class 1 : 64.9%
```

```
In [29]: startTime = time.time()
         n_features_to_select = 20
         step = 0.05
         clf = LogisticRegression(C=1,
                                  penalty='l1',
                                  class_weight='balanced',
                                  random_state=42)
         selector = RFE(estimator=clf, n_features_to_select=n_features_to_select, step=step)
         selector.fit(X_train, y_train)
         print(f"Optimal support of size {n_features_to_select} found in {time.time() - startTim
```

```
Optimal support of size 20 found in 726.4 s
```

```
In [30]: small_mask = selector.support_.copy()
         X_train = X_train[:,small_mask]
         X_test = X_test[:,small_mask]
         print(f"Number of features: p={X_train.shape[1]}")
```

```
Number of features: p=20
```

```
In [31]: print(f"Selected {X_train.shape[1]} features:\n")
         print(", ".join(X.columns[small_mask]))
```

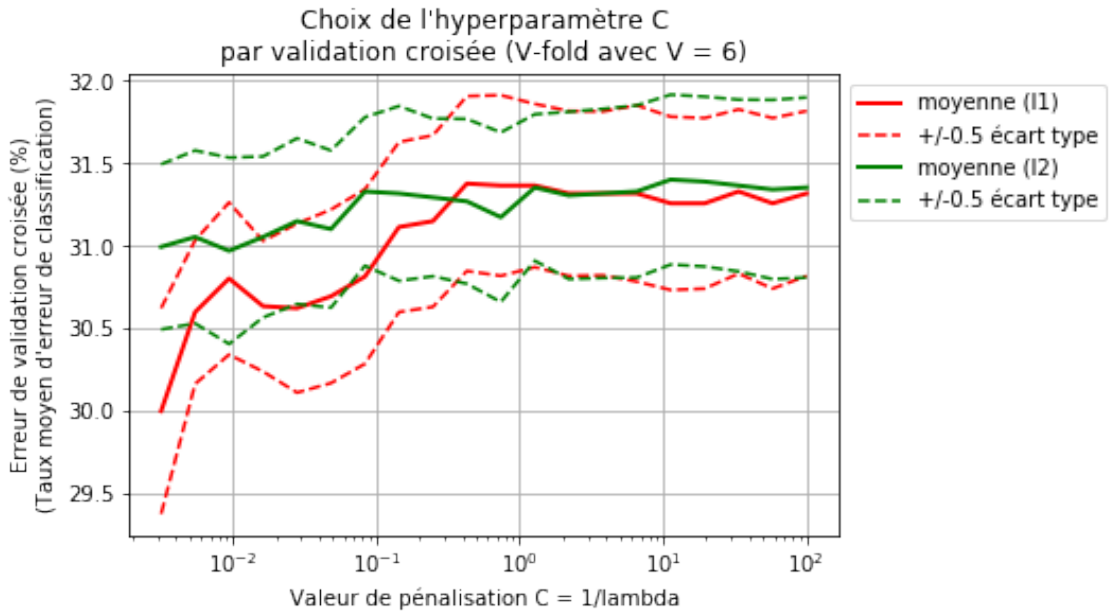
```
Selected 20 features:

NB_A301, NB_D108, NB_D703, JUSTICE, ACM8, NB_A115, NB_D101, NB_F121_NB_COU, NB_F116_NB_COU, NB_A
```

1.4.4 c) Using selected features to fit various models

Logistic regresion using L1 and L2

```
In [32]: nb_value = 20 # Nombre de valeurs testées pour l'hyperparamètre
         mean_score_l1 = np.zeros(nb_value)
         mean_score_l2 = np.zeros(nb_value)
         C_log = np.logspace(-2.5,2,nb_value)
         cv = 6 # V-fold, nombre de fold

         mean_score_l1 = np.empty(nb_value)
         std_scores_l1 = np.empty(nb_value)

         mean_score_l2 = np.empty(nb_value)
         std_scores_l2 = np.empty(nb_value)

         np.random.seed(seed=42)

         startTime = time.time()

         for i, C in enumerate(C_log):
             clf = LogisticRegression(C=C, penalty='l1',
                                      tol=0.01, random_state=42,
                                      class_weight='balanced')
             mean_score_l1[i] = 100*np.mean(1-cross_val_score(clf,
                                                              X_train,
                                                              y_train,
                                                              cv=cv,
                                                              scoring='accuracy'))
             std_scores_l1[i] = 100*np.std(1-cross_val_score(clf,
                                                             X_train,
                                                             y_train,
                                                             cv=cv,
                                                             scoring='accuracy'))


         for i, C in enumerate(C_log):
             clf = LogisticRegression(C=C, penalty='l2', tol=0.01, random_state=42, class_weight
             mean_score_l2[i] = 100*np.mean(1-cross_val_score(clf,
                                                              X_train,
                                                              y_train,
                                                              cv=cv,
                                                              scoring='accuracy'))
             std_scores_l2[i] = 100*np.std(1-cross_val_score(clf,
                                                             X_train,
                                                             y_train,
                                                             cv=cv,
                                                             scoring='accuracy'))
```

```
plt.figure()
plt.semilogx(C_log,mean_score_l1[:],'r',linewidth=2,label='moyenne (l1)')
plt.semilogx(C_log,mean_score_l1[:]-0.5*std_scores_l1[:],
             'r--', label=u'+/-0.5 écart type')
plt.semilogx(C_log,mean_score_l1[:]+0.5*std_scores_l1[:],'r--')

plt.semilogx(C_log,mean_score_l2[:],'g',linewidth=2,label='moyenne (l2)')
plt.semilogx(C_log,mean_score_l2[:]-0.5*std_scores_l2[:], 'g--', label=u'+/-0.5 écart t
plt.semilogx(C_log,mean_score_l2[:]+0.5*std_scores_l2[:],'g--')

plt.xlabel("Valeur de pénalisation C = 1/lambda")
plt.ylabel(u"Erreur de validation croisée (%)\n(Taux moyen d'erreur de classification)"
plt.title(u"Choix de l'hyperparamètre C\npar validation croisée \
(V-fold avec V = %s)" % (cv))
plt.legend(bbox_to_anchor=(1, 1))
plt.grid()
plt.show()
print("Détermination des paramètres optimaux en %0.1f s" % (time.time() - startTime))
print("Pénalisation l1, valeur optimale : C = %0.2f" % (C_log[np.argmin(mean_score_l1)]
print("Pénalisation l2, valeur optimale : C = %0.2f" % (C_log[np.argmin(mean_score_l2)]
```




```
Détermination des paramètres optimaux en 28.2 s
Pénalisation l1, valeur optimale : C = 0.00
Pénalisation l2, valeur optimale : C = 0.01
```

In [33]:
```
# Learning on full training set with optimals hyperparameters
# and score evaluation on test set
```

```
        clf = LogisticRegression(C=C_log[np.argmin(mean_score_l1)],
                                 penalty='l1',
                                 random_state=42,
                                 class_weight='balanced')
        clf.fit(X_train, y_train)
        y_test_pred = clf.predict(X_test)
        accuracy = clf.score(X_test, y_test)
        print(f"Model score\n- Accuracy : {accuracy*100:0.1f} %")
        f1 = f1_score(y_test, y_test_pred)
        p = precision_score(y_test, y_test_pred)
        r = recall_score(y_test, y_test_pred)
        print(f"- Precision : {p*100:0.1f} % (Happy # positive class)")
        print(f"- Recall : {r*100:0.1f} %")
        print(f"- F1 score : {f1*100:0.1f} %")
```

```
Model score
- Accuracy : 70.3 %
- Precision : 73.9 % (Happy # positive class)
- Recall : 83.7 %
- F1 score : 78.5 %
```

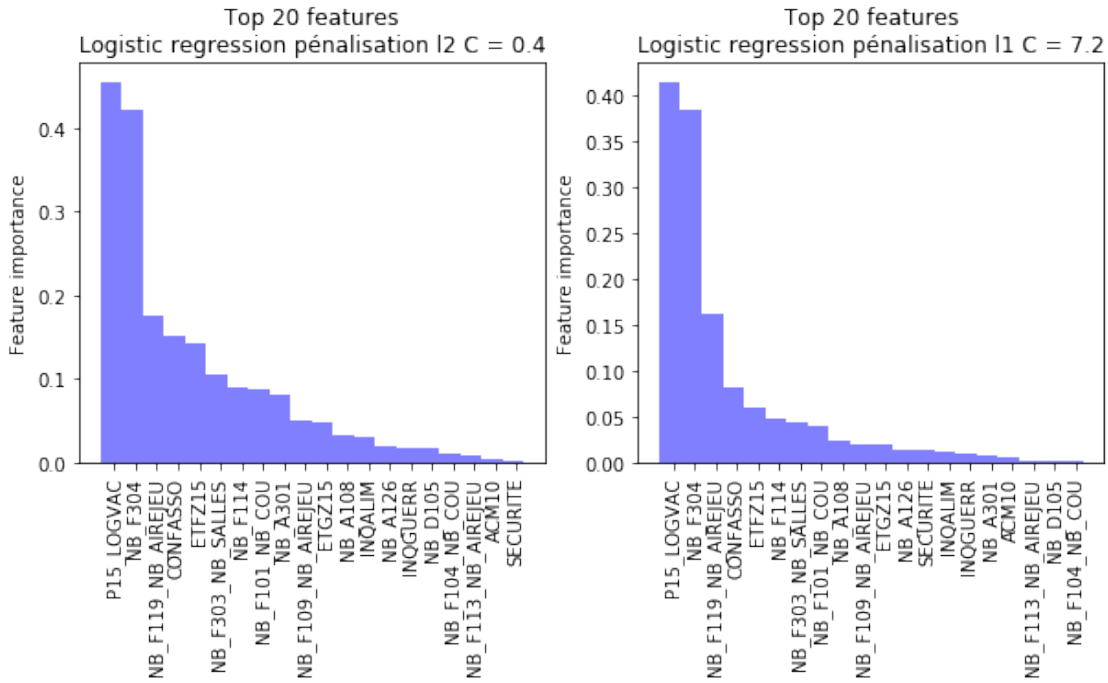
```
In [34]: # Learning on full training set with optimals hyperparameters
         # and score evaluation on test set
         clf = LogisticRegression(C=C_log[np.argmin(mean_score_l2)],
                                  penalty='l2',
                                  random_state=42,
                                  class_weight='balanced')
         clf.fit(X_train, y_train)
         y_test_pred = clf.predict(X_test)
         accuracy = clf.score(X_test, y_test)
         print(f"Model score\n- Accuracy : {accuracy*100:0.1f} %")
         f1 = f1_score(y_test, y_test_pred)
         p = precision_score(y_test, y_test_pred)
         r = recall_score(y_test, y_test_pred)
         print(f"- Precision : {p*100:0.1f} % (Happy # positive class)")
         print(f"- Recall : {r*100:0.1f} %")
         print(f"- F1 score : {f1*100:0.1f} %")
```

```
Model score
- Accuracy : 68.0 %
- Precision : 74.5 % (Happy # positive class)
- Recall : 76.8 %
- F1 score : 75.6 %
```

```
In [35]: # Use regression coefficients to rank features
         clf = LogisticRegression(C=C_log[np.argmin(mean_score_l2)],
                                  penalty='l2',
```

```
                                random_state=42,
                                class_weight='balanced')


        clf.fit(X_train,y_train)
        coef_l2 = abs(clf.coef_)
        coef_sorted_l2 = -np.sort(-coef_l2).reshape(-1)
        print(coef_sorted_l2)
        features_sorded_l2 = np.argsort(-coef_l2).reshape(-1)
        print(features_sorded_l2)
        features_name = np.array(features)
        features_name_sorted_l2 = features_name[features_sorded_l2]

        clf = LogisticRegression(C=C_log[np.argmin(mean_score_l1)],
                                penalty='l2',
                                random_state=42,
                                class_weight='balanced')

        clf.fit(X_train,y_train)
        coef_l1 = abs(clf.coef_)
        coef_sorted_l1 = -np.sort(-coef_l1).reshape(-1)
        features_sorded_l1 = np.argsort(-coef_l1).reshape(-1)
        features_name_sorted_l1 = features_name[features_sorded_l1]

        nf = X_train.shape[1]

        fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
        ind = np.arange(nf)    # the x locations for the groups

        plt.subplot(1, 2, 1)
        p1 = plt.bar(ind, coef_sorted_l2[0:nf], 1, color='b',alpha=0.5)
        plt.ylabel('Feature importance')
        plt.title(u'Top %i features\nLogistic regression pénalisation l2 C = 0.4' % nf)
        plt.xticks(ind + 0.35/2.0, features_name_sorted_l2[0:nf], rotation = 90)

        plt.subplot(1, 2, 2)
        p1 = plt.bar(ind, coef_sorted_l1[0:nf], 1, color='b',alpha=0.5)
        plt.ylabel('Feature importance')
        plt.title(u'Top %i features\nLogistic regression pénalisation l1 C = 7.2' % nf)
        plt.xticks(ind + 0.35/2.0, features_name_sorted_l1[0:nf], rotation = 90)

        plt.show()
```

```
[ 0.4549099   0.42221413  0.17622814  0.15161577  0.14338918  0.10525766
  0.08990527  0.08769821  0.08126144  0.04908717  0.04774626  0.03318206
  0.03061024  0.02029375  0.01800904  0.01711365  0.01062747  0.00864857
  0.00407274  0.00242724]
[19 16 11  4  2  7 14  5  3 17 18  8  6  9 10  0 15 12  1 13]
```



Random Forest

```
In [36]: startTime = time.time()
         n_estimators_range = [32,64,128,256,512]
         max_depth_range = [4,8,16,32,64]
         param_grid = dict(n_estimators=n_estimators_range, max_depth = max_depth_range)

         params = {'max_features' :'sqrt', 'random_state' : 32,
                   'min_samples_split' : 2, 'class_weight' : 'balanced'}
         clf = RandomForestClassifier(**params)

         grid = GridSearchCV(clf, scoring='accuracy', param_grid=param_grid)
         grid.fit(X_train, y_train)
         print(f"Determination of optimal hyperparameters in {time.time() - startTime:0.1f} s")
         print(f"Optimal values are {grid.best_params_} \n\
         Accuracy Score of cross valdation {100*grid.best_score_:0.2f}%")

         # Learning on full training set with optimals hyperparameters and score on test set
         params = {'max_features' :'sqrt', 'random_state' : 32,
                   'min_samples_split' : 2, 'class_weight' : 'balanced',
                   'n_estimators' : grid.best_params_['n_estimators'],
                   'max_depth' : grid.best_params_['max_depth']}
         clf = RandomForestClassifier(**params).fit(X_train, y_train)
         clf.fit(X_train, y_train)
```

```
        y_test_pred = clf.predict(X_test)

        print(f"Random Forest, p={X_train.shape[1]}")
        accuracy = clf.score(X_test, y_test)
        f1 = f1_score(y_test, y_test_pred)
        p = precision_score(y_test, y_test_pred)
        r = recall_score(y_test, y_test_pred)
        print(f"Model score\n- Accuracy : {accuracy*100:0.1f} %")
        print(f"- Precision : {p*100:0.1f} % (Happy # positive class)")
        print(f"- Recall : {r*100:0.1f} %")
        print(f"- F1 score : {f1*100:0.1f} %")
```

```
Determination of optimal hyperparameters in 127.1 s
Optimal values are {'max_depth': 4, 'n_estimators': 256}
Accuracy Score of cross valdation 69.17%
Random Forest, p=20
Model score
- Accuracy : 67.8 %
- Precision : 74.7 % (Happy # positive class)
- Recall : 76.1 %
- F1 score : 75.4 %
```

1.5 5) Multi class regression

1.5.1 a) Training and test set preparation

```
In [37]: features = df.columns.drop(['HEUREUX',"HEUREUX_CLF", "HEUREUX_REG"])
         pred = "HEUREUX"

         # treating remaining missing values
         df_tmp = df.loc[:,set(features) | {pred} ]
         df_tmp.loc[df_tmp["HEUREUX"]==5,"HEUREUX"]=None
         df_tmp = df_tmp.dropna()

         X = df_tmp.loc[:,features]
         y = df_tmp[pred]

         X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                             test_size=0.2,
                                                             random_state=42)

         scaler = StandardScaler().fit(X_train)
         X_train = scaler.transform(X_train)
         X_test = scaler.transform(X_test)

         print(f"Number exemple: {y.shape[0]}\n- training set: \
         {y_train.shape[0]}\n- test set: {y_test.shape[0]}")
         print(f"Number of features: p={X_train.shape[1]}")
```

```
        print(f"Number of class: {len(np.unique(y))}")
        for c in np.unique(y):
            print(f"class {c:0.0f} : {100*np.sum(y==c)/len(y):0.1f}%")
```

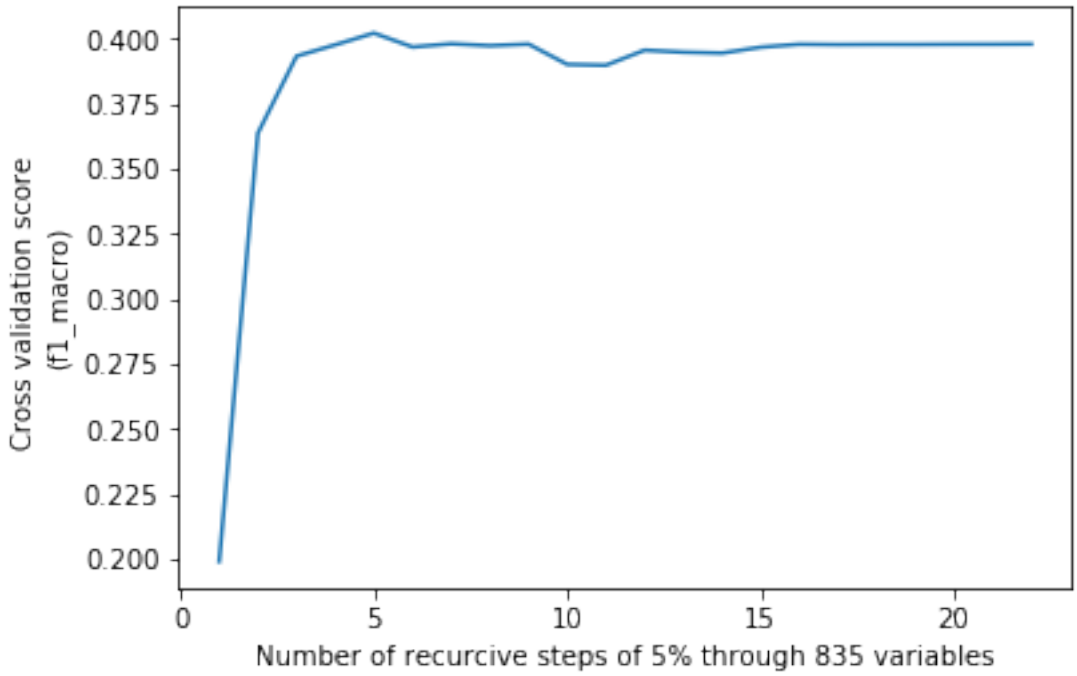
```
Number exemple: 10445
- training set: 8356
- test set: 2089
Number of features: p=835
Number of class: 4
class 1 : 1.8%
class 2 : 33.3%
class 3 : 49.0%
class 4 : 15.9%
```

1.5.2 b) Feature selection

In [38]:
```
startTime = time.time()

scoring='f1_macro'
step = 0.05

clf = LogisticRegression(C=1,
                         penalty='l1',
                         class_weight='balanced',
                         random_state=42)

rfecv = RFECV(estimator=clf, step=step, cv=StratifiedKFold(2),
              scoring=scoring)

rfecv.fit(X_train, y_train)

print("Optimal number of features : %d" % rfecv.n_features_)

# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel(f"Number of recurcive steps of {100*step:0.0f}% through {X_train.shape[1]} v
plt.ylabel(f"Cross validation score \n({scoring})")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
print(f"Détermination des features optimales en %0.1f s" % (time.time() - startTime))
```

```
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
```

```
Optimal number of features : 138
```




```
Détermination des features optimales en 8071.4 s
```

```
In [39]: nclf_mask = rfecv.support_.copy()
         X_train = X_train[:,nclf_mask]
         X_test = X_test[:,nclf_mask]
         print(f"Number of features: p={X_train.shape[1]}")
```

```
Number of features: p=138
```

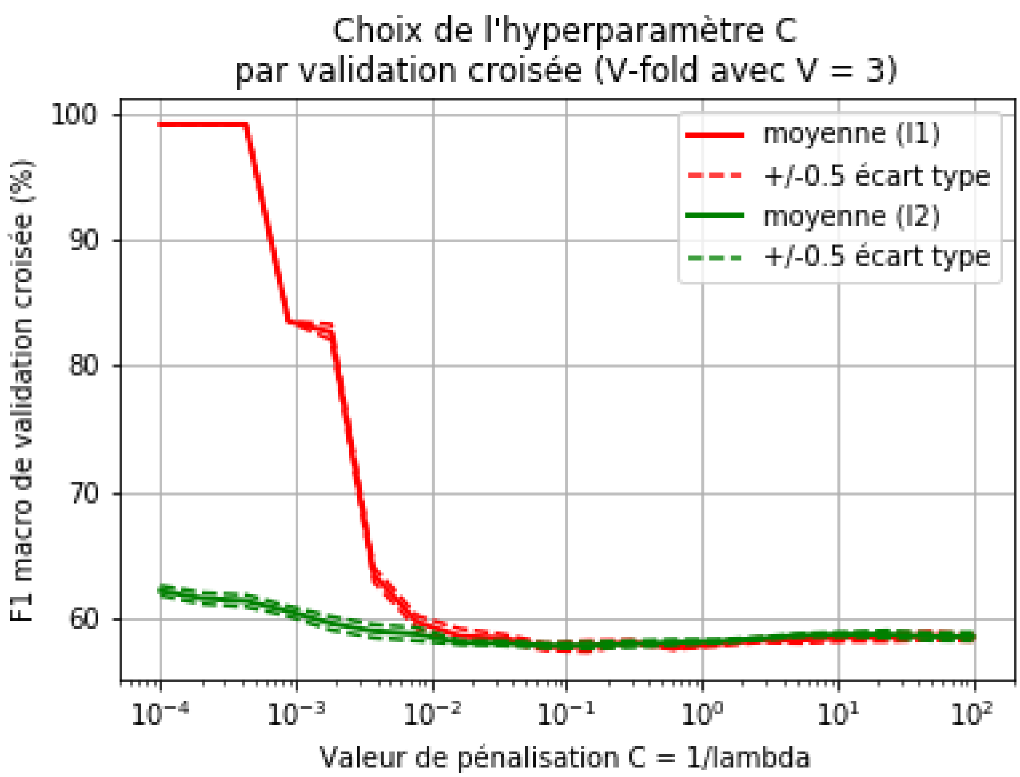
```
In [40]: nb_value = 20 # Nombre de valeurs testées pour l'hyperparamètre
         mean_score_l1 = np.zeros(nb_value)
         mean_score_l2 = np.zeros(nb_value)
         C_log = np.logspace(-4,2,nb_value)
         cv = 3 # V-fold, nombre de fold

         mean_score_l1 = np.empty(nb_value)
         std_scores_l1 = np.empty(nb_value)

         mean_score_l2 = np.empty(nb_value)
         std_scores_l2 = np.empty(nb_value)
```

```
np.random.seed(seed=42)

startTime = time.time()

for i, C in enumerate(C_log):
    clf = LogisticRegression(C=C, penalty='l1', tol=0.01, random_state=42, class_weight
    mean_score_l1[i] = 100*np.mean(1-cross_val_score(clf,
                                                     X_train,
                                                     y_train,
                                                     cv=cv,
                                                     scoring='f1_macro'))
    std_scores_l1[i] = 100*np.std(1-cross_val_score(clf,
                                                    X_train,
                                                    y_train,
                                                    cv=cv,
                                                    scoring='f1_macro'))


for i, C in enumerate(C_log):
    clf = LogisticRegression(C=C, penalty='l2', tol=0.01, random_state=42, class_weight
    mean_score_l2[i] = 100*np.mean(1-cross_val_score(clf,
                                                     X_train,
                                                     y_train,
                                                     cv=cv,
                                                     scoring='f1_macro'))
    std_scores_l2[i] = 100*np.std(1-cross_val_score(clf,
                                                    X_train,
                                                    y_train,
                                                    cv=cv,
                                                    scoring='f1_macro'))

plt.figure()
plt.semilogx(C_log,mean_score_l1[:],'r',linewidth=2,label='moyenne (l1)')
plt.semilogx(C_log,mean_score_l1[:]-0.5*std_scores_l1[:],
             'r--', label=u'+/-0.5 écart type')
plt.semilogx(C_log,mean_score_l1[:]+0.5*std_scores_l1[:],'r--')

plt.semilogx(C_log,mean_score_l2[:],'g',linewidth=2,label='moyenne (l2)')
plt.semilogx(C_log,mean_score_l2[:]-0.5*std_scores_l2[:], 'g--',
             label=u'+/-0.5 écart type')
plt.semilogx(C_log,mean_score_l2[:]+0.5*std_scores_l2[:],'g--')

plt.xlabel("Valeur de pénalisation C = 1/lambda")
plt.ylabel("F1 macro de validation croisée (%)")
plt.title(u"Choix de l'hyperparamètre C\npar validation croisée \
(V-fold avec V = %s)" % (cv))
plt.legend(bbox_to_anchor=(1, 1))
```

```
    plt.grid()
    plt.show()
    print(f"Détermination des paramètres optimaux en \
    {time.time() - startTime:0.1f} s")
    print(f"Pénalisation l1, valeur optimale : \
    C = {C_log[np.argmax(mean_score_l1)]:0.4f}")
    print(f"Pénalisation l2, valeur optimale : \
    C = {C_log[np.argmax(mean_score_l2)]:0.4f}")
```

```
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
```

```
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
//anaconda/envs/py36/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: Undefin
  'precision', 'predicted', average, warn_for)
```



```
Détermination des paramètres optimaux en 983.7 s
Pénalisation l1, valeur optimale : C = 0.0001
Pénalisation l2, valeur optimale : C = 0.0001
```

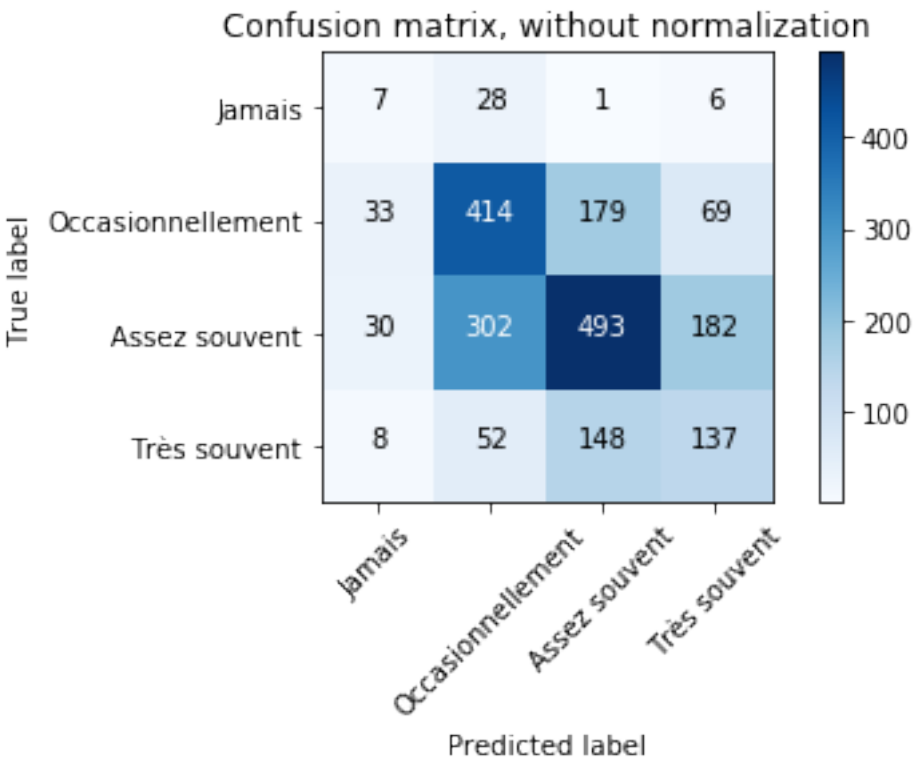
In [41]:
```
# Learning on full training set with optimals hyperparameters and score on test set
clf = LogisticRegression(C=C_log[np.argmax(mean_score_l2)],
                         penalty='l2',
                         tol=0.01,
                         random_state=42,
                         class_weight='balanced')
clf.fit(X_train, y_train)
y_test_pred = clf.predict(X_test)
```

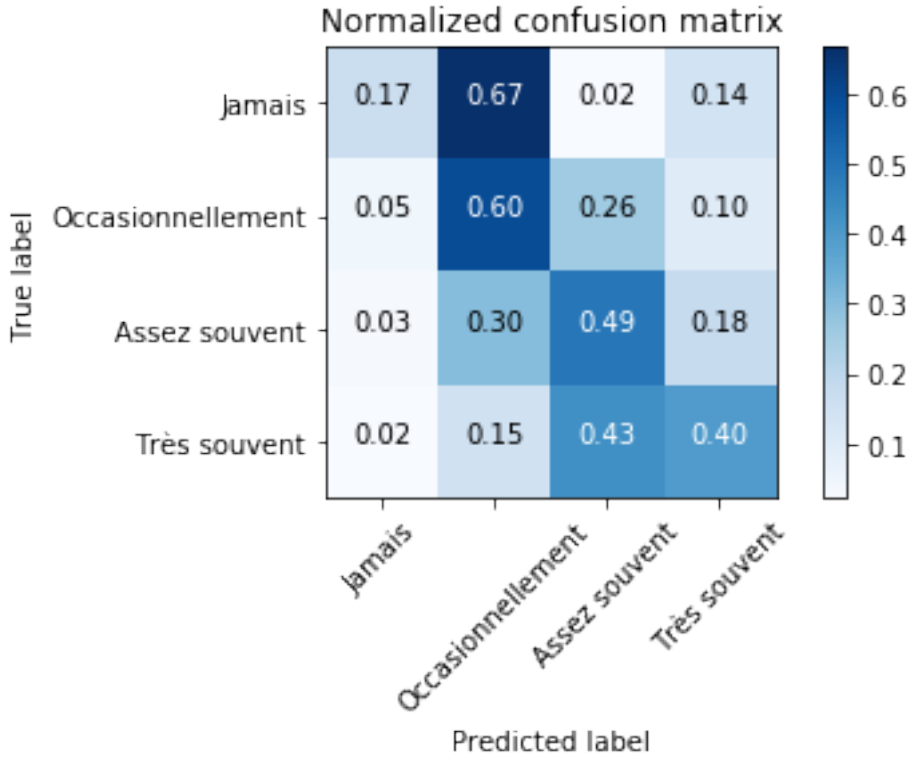
In [42]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [43]:
```
# Model evaluation
```

```
        class_names = ["Jamais",
                       "Occasionnellement",
                       "Assez souvent",
                       "Très souvent" ]
        f1_scores = f1_score(y_test, y_test_pred, labels = [1,2,3,4], average=None)
        for i,c in enumerate(class_names):
            print(f"f1 score class '{c}' : {100*f1_scores[i]:0.1f}%")
        acurracy = clf.score(X_test, y_test)
        f1_macro = f1_score(y_test, y_test_pred, average='macro')
        f1_weighted = f1_score(y_test, y_test_pred, average='weighted')
        print(f"Average scores :\nf1 macro : {f1_macro*100:0.4f} %\n\
        f1 weighted : {f1_weighted*100:0.4f} %\nacurracy : {acurracy*100:0.4f} %")

        # Compute confusion matrix
        cnf_matrix = confusion_matrix(y_test, y_test_pred)
        np.set_printoptions(precision=2)

        # Plot non-normalized confusion matrix
        plt.figure()
        plot_confusion_matrix(cnf_matrix, classes=class_names,
                              title='Confusion matrix, without normalization')

        # Plot normalized confusion matrix
        plt.figure()
        plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                              title='Normalized confusion matrix')

        plt.show()
```

```
f1 score class 'Jamais' : 11.7%
f1 score class 'Occasionnellement' : 55.5%
f1 score class 'Assez souvent' : 53.9%
f1 score class 'Très souvent' : 37.1%
Average scores :
f1 macro : 39.5539 %
f1 weighted : 50.8346 %
acurracy : 50.3112 %
Confusion matrix, without normalization
[[  7  28   1   6]
 [ 33 414 179  69]
 [ 30 302 493 182]
 [  8  52 148 137]]
Normalized confusion matrix
[[ 0.17  0.67  0.02  0.14]
 [ 0.05  0.6   0.26  0.1 ]
 [ 0.03  0.3   0.49  0.18]
 [ 0.02  0.15  0.43  0.4 ]]
```


Confusion matrix, without normalization
Jamais
Occasionnellement
Assez souvent
Très souvent
True label
7
28
1
6
33
414
179
69
30
302
493
182
8
52
148
137
400
300
200
100
Jamais
Occasionnellement
Assez souvent
Très souvent
Predicted label


Normalized confusion matrix
Jamais
Occasionnellement
Assez souvent
Très souvent
True label
0.17
0.67
0.02
0.14
0.05
0.60
0.26
0.10
0.03
0.30
0.49
0.18
0.02
0.15
0.43
0.40
0.6
0.5
0.4
0.3
0.2
0.1
Jamais
Occasionnellement
Assez souvent
Très souvent
Predicted label