cdv 2017-2018 impact of clustering

August 28, 2018

0.0.1 I) Loading data and preparing dataset - scope 2017-2018

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
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
```

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)
# 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)
```

In [4]:
```python
# load various variable set
filename = path_dump / Path("dict_var_groups.sav")
with open(filename, 'rb') as fp:
     dict_var_groups = pickle.load(fp)

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']
```

In [5]:
```python
exclusion = com_var | tech_var | bizz_var | text_var
scope_2017_2018_var_kept = scope_2017_2018_var - exclusion
cat_var_kept = cat_max9_var & scope_2017_2018_var_kept
scope_quant_var = (quant_var & scope_2017_2018_var_kept)
quant_null = np.sum(cdv_ssfmt.loc[:,scope_quant_var].isnull())
quant_var_kept = set(quant_null[quant_null < 200].index)

print(f"Out of {cdv.shape[1]} variable {len(scope_2017_2018_var)} \
are used in 2017 and 2018 ")
print(f"{len(scope_2017_2018_var & exclusion)} of 'technical' variable \
such as 'inseenum' are excluded ")
print(f"{len(scope_2017_2018_var_kept)} are remaining :")
print(f"\t{len(cat_var & scope_2017_2018_var_kept)} \
categorial variables : ")
print(f"\t\t{len(cat_max9_var & scope_2017_2018_var_kept)} \
with maximum 9 modalities  ")
print(f"\t\t{len(cat_min10_var & scope_2017_2018_var_kept)} \
with more modalities ... excluded")
print(f"\t{len(quant_var & scope_2017_2018_var_kept)} \
variables are quantitative ")
print(f"\t\t{len(quant_var_kept)} have less than 200 missing values")
print(f"\t\t{len(scope_quant_var)-len(quant_var_kept)} \
have more ... excluded")

scope = cat_var_kept | quant_var_kept
df = cdv_ssfmt.loc[cdv_ssfmt['ANNEEFUZ'].isin({39,40}),scope]
df.loc[:,cat_var_kept - {"HEUREUX"}] = cdv.loc[:,cat_var_kept - {"HEUREUX"}]
print(f"\nFinal number of variable kept : {df.shape[1]}")
```

```
Out of 354 variable 297 are used in 2017 and 2018
31 of 'technical' variable such as 'inseenum' are excluded
266 are remaining :
```

```
        180 categorial variables :
                165 with maximum 9 modalities
                15 with more modalities ... excluded
        86 variables are quantitative
                60 have less than 200 missing values
                26 have more ... excluded

Final number of variable kept : 225
```

```
In [6]: p = df.shape[1]
        print(f"{p} columns out of which {len(cat_var_kept)-1} \
        are corresponding to categorial features")
```

```
225 columns out of which 164 are corresponding to categorial features
```

```
In [7]: df = pd.get_dummies(df,
                            columns=cat_var_kept - {"HEUREUX"},
                            dummy_na = True,
                            drop_first=1)
```

```
In [8]: q = df.shape[1]
        print(f"{q} columns after encoding of {len(cat_var_kept)-1} categorial \
        variables in {len(cat_var_kept)-1+q-p} binary variables \
        (K-1 one hot encoding)")
```

```
645 columns after encoding of 164 categorial variables in 584 binary variables (K-1 one hot enco
```

```
In [9]: # encoding of "HEUREUX" '[nsp]'
        df.loc[df["HEUREUX"]==5,"HEUREUX"]= None

        # treating remaining missing values
        df_tmp = df.dropna()

        features = df_tmp.columns.drop(['HEUREUX'])
        X = df_tmp.loc[:,features]
        y = df_tmp["HEUREUX"]

        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)
```

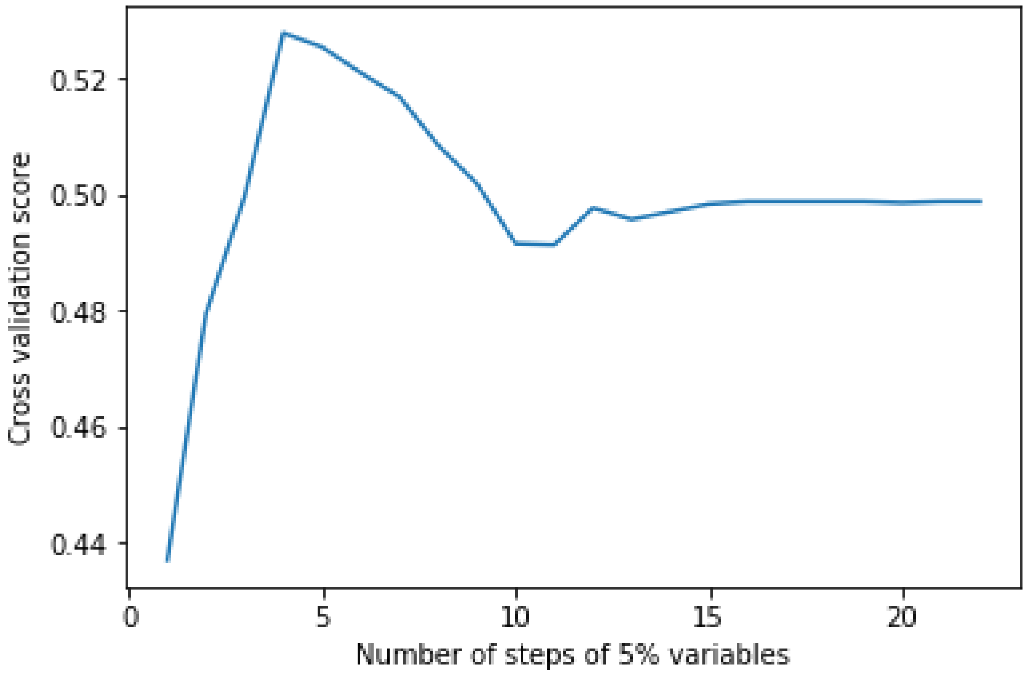
```
print(f"Number exemple: {y.shape[0]}\n\
- training set: {y_train.shape[0]}\n\
- test set: {y_test.shape[0]}")
print(f"Number of features: p={X_train.shape[1]}")
```

```
Number exemple: 5682
- training set: 4545
- test set: 1137
Number of features: p=644
```

0.0.2 II) Feature selection

In [12]:
```
startTime = time.time()

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

rfecv = RFECV(estimator=clf, step=0.05, cv=StratifiedKFold(2),
              scoring='accuracy')

rfecv.fit(X_train, y_train)

print("Optimal number of features : %d" % rfecv.n_features_)
#print(f"Correspondingf score {grid.best_score_:0.4f}")
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of steps of 5% variables")
plt.ylabel("Cross validation score")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()

print("Détermination des features optimales en %0.1f s" % (time.time() - startTime))
```

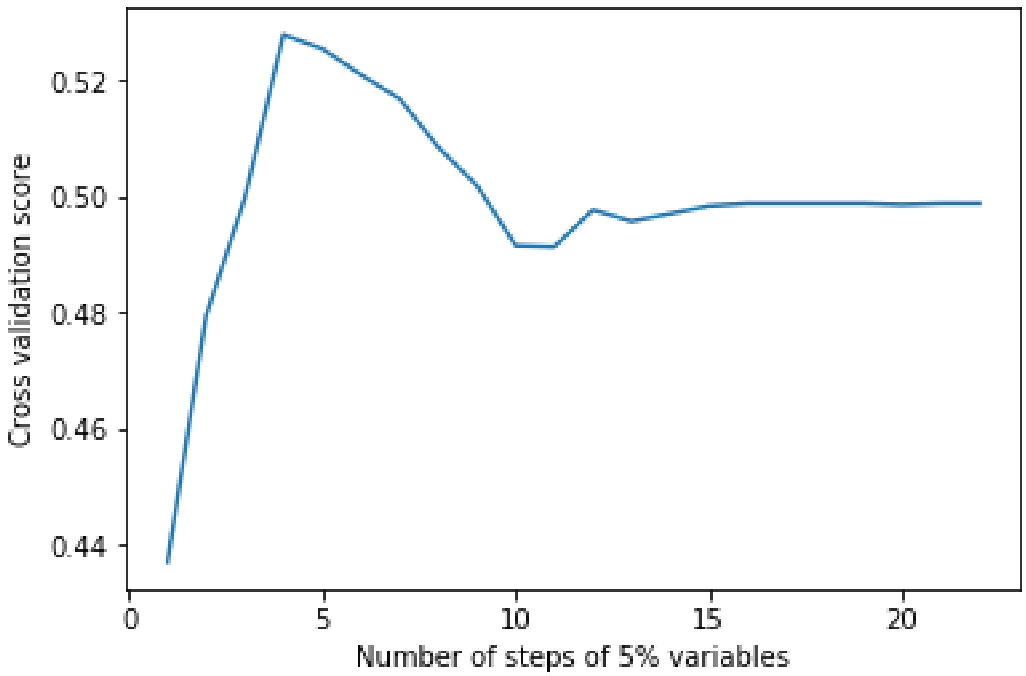
```
Optimal number of features : 68
```




```
Détermination des features optimales en 977.1 s
```

```
In [13]: startTime = time.time()

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

         rfecv = RFECV(estimator=clf, step=0.05, cv=StratifiedKFold(2),
                       scoring='f1_micro')

         rfecv.fit(X_train, y_train)

         print("Optimal number of features : %d" % rfecv.n_features_)
         #print(f"Correspondingf score {grid.best_score_:0.4f}")
         # Plot number of features VS. cross-validation scores
         plt.figure()
         plt.xlabel("Number of steps of 5% variables")
         plt.ylabel("Cross validation score")
         plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
         plt.show()

         print("Détermination des features optimales en %0.1f s" % (time.time() - startTime))
```

```
Optimal number of features : 68
```



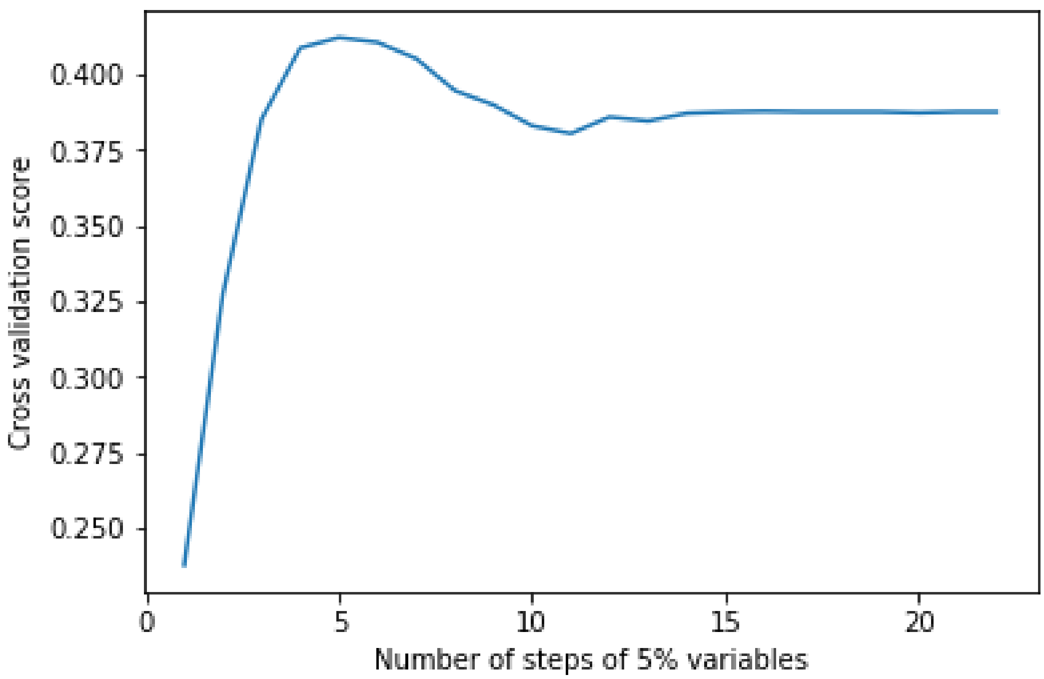

```
Détermination des features optimales en 988.2 s
```

In [14]:
```python
startTime = time.time()

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

rfecv = RFECV(estimator=clf, step=0.05, cv=StratifiedKFold(2),
              scoring='f1_macro')

rfecv.fit(X_train, y_train)

print("Optimal number of features : %d" % rfecv.n_features_)
#print(f"Correspondingf score {grid.best_score_:0.4f}")
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of steps of 5% variables")
plt.ylabel("Cross validation score")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
```

```
plt.show()

print("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 : 100
```



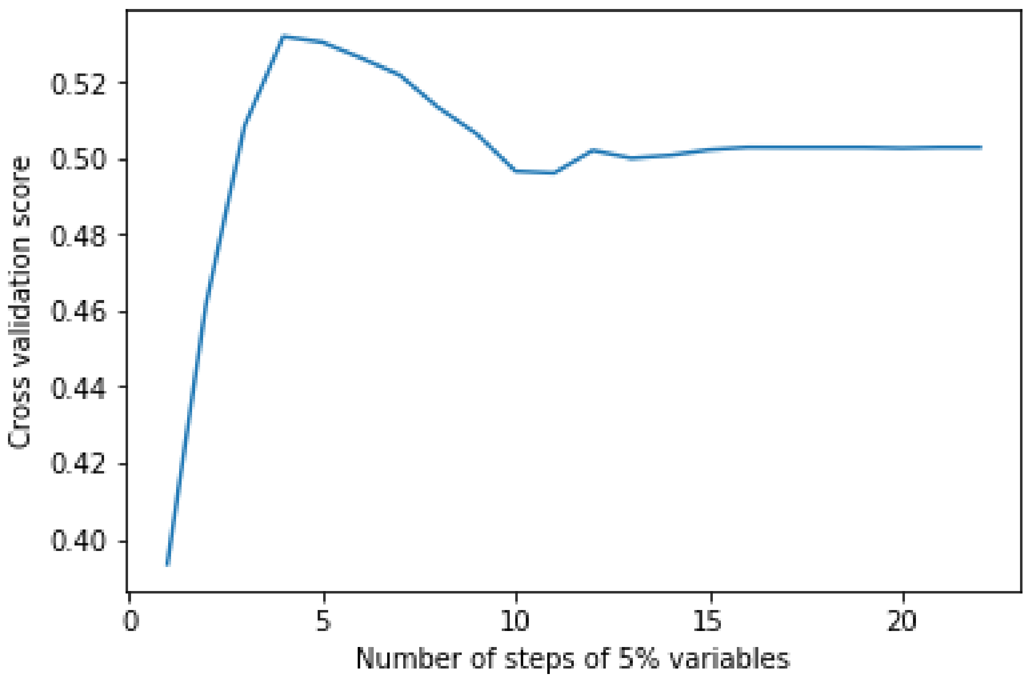

```
Détermination des features optimales en 1011.3 s
```

In [15]:
```
startTime = time.time()

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

rfecv = RFECV(estimator=clf, step=0.05, cv=StratifiedKFold(2),
              scoring='f1_weighted')
```

```python
rfecv.fit(X_train, y_train)

print("Optimal number of features : %d" % rfecv.n_features_)
#print(f"Correspondingf score {grid.best_score_:0.4f}")
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of steps of 5% variables")
plt.ylabel("Cross validation score")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()

print("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 : 68
```




```
Détermination des features optimales en 1018.0 s
```

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

In [ ]:
```python
startTime = time.time()
n_estimators_range = [16,32,64,128,256]
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='f1_weighted',
                    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\
F1 weighted 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)
accuracy = clf.score(X_test, y_test)
y_pred = clf.predict(X_test)
print(f"Random Forest, p={X_train.shape[1]}")
print(f"Accuracy: {accuracy*100:0.2f}%")
print(f"... done in {time.time() - startTime:0.1f}")
```

In [ ]:
```python
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 [ ]:
```
class_names = ["Jamais",
               "Occasionnellement",
               "Assez souvent",
               "Très souvent"]
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, y_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()
```

In [ ]:
```
len(y_test)
```