Felix_prototype_V0

October 27, 2018

1 Felix prototype

Version 0
Date 21/10/2018
Model used : **Random Forest** Classifier on features selected through **lasso**
Clustering method used : **Hierarchical clustering** using **ward metric** based on 6 **NOT variable**

```
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
        import itertools
        import pickle
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.model_selection import cross_val_score, GridSearchCV
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score
        from sklearn.model_selection import StratifiedKFold
        from sklearn.utils import resample
```

```
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 data
        file = path_data / Path("dataset.csv")
        with Path.open(file, 'rb') as fp:
            dataset = pd.read_csv(fp,  encoding='utf-8',low_memory=False, index_col = 0)
```

1.0.1 Features scope and selection strategy

Features are selected using lasso on the full scope of feature. The 50 more important features (logistic regression coef ranking) are kept regardless of their activability

In [12]:
```
# load feature sets
filename = path_dump / Path("dict_features_sets.sav")
with open(filename, 'rb') as fp:
     dict_features_sets = pickle.load(fp)

usual_common_scope_features = dict_features_sets['usual_common_scope_features']

cdv_actionable_individual_1_features = dict_features_sets.get('cdv_actionable_individua
cdv_actionable_individual_2_features = dict_features_sets.get('cdv_actionable_individua
cdv_actionable_admin_1_features = dict_features_sets.get('cdv_actionable_admin_1_featur
cdv_actionable_admin_2_features = dict_features_sets.get('cdv_actionable_admin_2_featur
insee_recreation_actionable_admin_1_features = dict_features_sets.get('insee_recreation
insee_recreation_actionable_admin_2_features = dict_features_sets.get('insee_recreation
insee_environment_actionable_admin_1_features = dict_features_sets.get('insee_environme
insee_environment_actionable_admin_2_features = dict_features_sets.get('insee_environme
insee_demographics_actionable_admin_1_features = dict_features_sets.get('insee_demograp
insee_demographics_actionable_admin_2_features = dict_features_sets.get('insee_demograp

RFE_LogisticRegression_10_features = dict_features_sets['RFE_LogisticRegression_10_feat
RFE_LogisticRegression_20_features = dict_features_sets['RFE_LogisticRegression_20_feat
RFE_LogisticRegression_50_features = dict_features_sets['RFE_LogisticRegression_50_feat
RFE_LogisticRegression_100_features = dict_features_sets['RFE_LogisticRegression_100_fe

RFE_LinearSVC_100_features = dict_features_sets['RFE_LinearSVC_100_features'],
RFE_LinearSVC_50_features =  dict_features_sets['RFE_LinearSVC_50_features'],
RFE_LinearSVC_20_features =  dict_features_sets['RFE_LinearSVC_20_features'],
RFE_LinearSVC_10_features =  dict_features_sets['RFE_LinearSVC_10_features'],

SelectFromModel_LinearSCV_features = dict_features_sets['SelectFromModel_LinearSCV_feat
SelectFromModel_LogisticRegression_features = dict_features_sets['SelectFromModel_Logis
```

In [13]:
```
RFE_LinearSVC_10_features = RFE_LinearSVC_10_features[0]
RFE_LinearSVC_10_features
```

Out[13]:
```
{'AGE',
 'AGE5',
 'CADVIE',
 'CDV5',
 'ETATSAN',
 'NB_D103',
 'NOT_AMIS',
 'NOT_FAMI',
 'SEXE_3_nan',
 'SOUFFDEP_Oui'}
```

In [14]:
```
print("The most important features obtained using lasso:")
print(list(SelectFromModel_LogisticRegression_features))
```

```
The most important features obtained using lasso:
```

```
['RE_LOG_nan', 'ETATSAN', 'RE_MEDI_Oui', 'HANDICAP_Oui', 'SOUFFDEP_Oui', 'CHOIXNUC_Sans avis', '
```

1.0.2 Clustering method - feature used

Hierarchical clustering is used using 6 common "NOT_" variable

```
In [15]: # loading clustering
         file = path_data / Path("clustTest3.csv")
         with Path.open(file, 'rb') as fp:
             clustTest1 = pd.read_csv(fp,  encoding='utf-8',low_memory=False, sep=";", index_col
```

1.0.3 Training set and test set preparation

```
In [16]: df = dataset.loc[:,:]
         # 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

         scope = ( RFE_LinearSVC_10_features  )  & set(dataset.columns)
         n_max = 2000

         df = df.loc[:,scope | {"HEUREUX_CLF"} ].dropna()
         features = df.loc[:,scope ].columns

         X = df.loc[:,scope]
         y = df["HEUREUX_CLF"]


         Xs, ys = resample(X, y, random_state=42)

         Xs = Xs.iloc[0:n_max,:]
         ys = ys.iloc[0:n_max]

         X_train, X_test, y_train, y_test = train_test_split(Xs, ys,
                                                             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: 10754
- training set: 1600
- test set: 400
Number of features: p=10
Number of class: 2
class 0 : 35.1%
class 1 : 64.9%
```

1.0.4 Learning and model performance evaluation on full dataset (before clustering)

In [17]:
```
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} %")
```

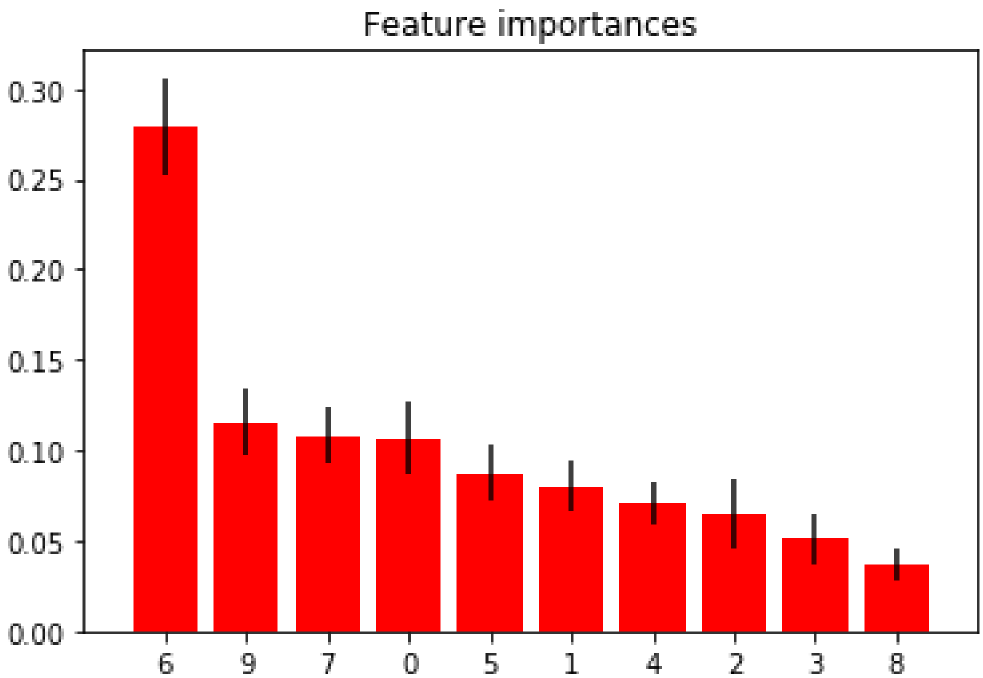
```
    res_full  = {
        'f1_score' : f1,
        'accuracy' : accuracy,
        'precision' : p,
        'recall' : r
    }
```

```
Determination of optimal hyperparameters in 41.2 s
Optimal values are {'max_depth': 32, 'n_estimators': 128}
Accuracy Score of cross valdation 73.12%
Random Forest, p=10
Model score
- Accuracy : 71.8 %
- Precision : 75.8 % (Happy # positive class)
- Recall : 81.0 %
- F1 score : 78.3 %
```

In [18]:
```
importances = clf.feature_importances_
std = np.std([tree.feature_importances_ for tree in clf.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]
features_name = np.array(features)
#features_name_sorted_rf = features_name[indices]
# Print the feature ranking
print("Feature ranking:")

max_features = 15

actionable_individual_1_features = cdv_actionable_individual_1_features
actionable_individual_2_features = cdv_actionable_individual_2_features
actionable_admin_1_features = cdv_actionable_admin_1_features | insee_recreation_action
actionable_admin_2_features = cdv_actionable_admin_2_features | insee_recreation_action


for f in range(min(X.shape[1],max_features)):
    print("%d. feature %d -%s- (%f)" % (f + 1, indices[f],features_name[indices[f]], im
    if features_name[indices[f]] in actionable_individual_1_features:
        print("\tActionable at individual level (1)")
    if features_name[indices[f]] in actionable_individual_2_features:
        print("\tActionable at individual level (2)")
    if features_name[indices[f]] in actionable_admin_1_features:
        print("\tActionable at administrative level (1)")
    if features_name[indices[f]] in actionable_admin_2_features:
        print("\tActionable at administrative level (2)")
```

```python
# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```

```
Feature ranking:
1. feature 6 -AGE- (0.279287)
2. feature 9 -NOT_AMIS- (0.115589)
        Actionable at individual level (1)
        Actionable at administrative level (2)
3. feature 7 -CDV5- (0.108272)
        Actionable at individual level (2)
        Actionable at administrative level (1)
4. feature 0 -ETATSAN- (0.106765)
        Actionable at individual level (1)
        Actionable at administrative level (1)
5. feature 5 -CADVIE- (0.087457)
        Actionable at individual level (1)
        Actionable at administrative level (1)
6. feature 1 -NOT_FAMI- (0.079764)
        Actionable at individual level (1)
        Actionable at administrative level (2)
7. feature 4 -NB_D103- (0.071202)
        Actionable at administrative level (1)
8. feature 2 -AGE5- (0.064716)
9. feature 3 -SOUFFDEP_Oui- (0.050810)
        Actionable at individual level (2)
        Actionable at administrative level (1)
10. feature 8 -SEXE_3_nan- (0.036137)
```



1.0.5 Learning and model performance evaluation on each clusters

In [19]:
```
n_estimators_range = [16,32,64,128]
max_depth_range = [2,4,8,16,32,64]
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'
         }
scope = ( RFE_LogisticRegression_50_features )  & set(dataset.columns)
features = df.loc[:,scope].columns
```

In [20]:
```
score_clustering_methods = []
clustering_methods = clustTest1.columns[2:3]

for method in clustering_methods:
    print("--------------------------------------------")
    print(f"\nAnalysis cluster method {method}")
    cluster_list = clustTest1[method].unique()
    print(f"liste of clusters : {cluster_list}")
    score_cluster = []
    for cluster in cluster_list:
        index_scope = clustTest1.loc[clustTest1[method]==cluster,:].index
        print(f"cluster {cluster} : {len(index_scope)} elements")
```

```
        Xc = X.loc[index_scope.intersection(X.index),:]
        yc = y[index_scope.intersection(X.index)]

        Xs, ys = resample(Xc, yc, random_state=42)

        Xs = Xs.iloc[0:n_max,:]
        ys = ys.iloc[0:n_max]

        X_train, X_test, y_train, y_test = train_test_split(Xs, ys,
                                                            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: {ys.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}%")


        startTime = time.time()
        clf = RandomForestClassifier(**params)
        grid = GridSearchCV(clf,
                            scoring='accuracy',
                            param_grid=param_grid)

        grid.fit(X_train, y_train)
        print(f"Optimal values are {grid.best_params_} \n\
cross validation score {100*grid.best_score_:0.2f}%")
        print()

        # Learning on full training set with optimals hyperparameters and score on test
        params_opt = {'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_opt).fit(X_train, y_train)


        y_test_pred = clf.predict(X_test)
        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)

                res  = {'f1_score' : f1,
                        'accuracy' : accuracy,
                        'precision' : p,
                        'recall' : r}

                cl = {'cluster' : cluster,
                      'size' : len(index_scope),
                      'model' : 'RandomForestClassifier',
                      'params' : params_opt,
                      'metrics' : res
                     }

                score_cluster.append(cl)

            d = {'clustering_method' : method,
                 'cluster_scores' : score_cluster
                }
            score_clustering_methods.append(d)
```

```
--------------------------------------------

Analysis cluster method clust3
liste of clusters : [2 4 6 1 3 5]
cluster 2 : 3053 elements
Number exemple: 2000
        - training set: 1600
        - test set: 400
Number of features: p=10
Number of class: 2
class 0 : 35.1%
class 1 : 64.9%
Optimal values are {'max_depth': 16, 'n_estimators': 64}
cross validation score 78.94%

cluster 4 : 2359 elements
Number exemple: 2000
        - training set: 1600
        - test set: 400
Number of features: p=10
Number of class: 2
class 0 : 35.1%
class 1 : 64.9%
Optimal values are {'max_depth': 32, 'n_estimators': 128}
cross validation score 79.06%
```

```
cluster 6 : 2313 elements
Number exemple: 2000
        - training set: 1600
        - test set: 400
Number of features: p=10
Number of class: 2
class 0 : 35.1%
class 1 : 64.9%
Optimal values are {'max_depth': 16, 'n_estimators': 128}
cross validation score 80.62%

cluster 1 : 528 elements
Number exemple: 510
        - training set: 408
        - test set: 102
Number of features: p=10
Number of class: 2
class 0 : 35.1%
class 1 : 64.9%
Optimal values are {'max_depth': 16, 'n_estimators': 128}
cross validation score 80.88%

cluster 3 : 1384 elements
Number exemple: 1361
        - training set: 1088
        - test set: 273
Number of features: p=10
Number of class: 2
class 0 : 35.1%
class 1 : 64.9%
Optimal values are {'max_depth': 16, 'n_estimators': 128}
cross validation score 84.01%

cluster 5 : 1494 elements
Number exemple: 1466
        - training set: 1172
        - test set: 294
Number of features: p=10
Number of class: 2
class 0 : 35.1%
class 1 : 64.9%
Optimal values are {'max_depth': 16, 'n_estimators': 128}
cross validation score 80.46%
```

1.0.6 Performance gain obtained using clustering

In [21]:
```
# F1 score
for score_method in score_clustering_methods:
    print(f"method {score_method['clustering_method']}:")
    average_score = 0
    total_size = 0
    for i, score_cluster in enumerate(score_method['cluster_scores']):
        print(f"cluster {score_cluster['cluster']} ({score_cluster['size']}), f1 macro
        average_score += score_cluster['metrics']['f1_score']*score_cluster['size']
        total_size += score_cluster['size']

    average_score = average_score / total_size
    print(f"average f1 on clusters {100*average_score:0.1f}% gain {100*(average_score-r
```

```
method clust3:
cluster 2 (3053), f1 macro 81.9%
cluster 4 (2359), f1 macro 86.8%
cluster 6 (2313), f1 macro 90.6%
cluster 1 (528), f1 macro 90.9%
cluster 3 (1384), f1 macro 93.5%
cluster 5 (1494), f1 macro 87.5%
average f1 on clusters 87.4% gain 9.1
```

In [22]:
```
# accuracy
for score_method in score_clustering_methods:
    print(f"method {score_method['clustering_method']}:")
    average_score = 0
    total_size = 0
    for i, score_cluster in enumerate(score_method['cluster_scores']):
        print(f"cluster {score_cluster['cluster']} ({score_cluster['size']}) , accuracy
        average_score = average_score + score_cluster['metrics']['accuracy']*score_clus
        total_size += score_cluster['size']
    average_score = average_score / total_size
    print(f"average accuracy on clusters {100*average_score:0.1f}% gain {100*(average_s
```

```
method clust3:
cluster 2 (3053) , accuracy 77.0%
cluster 4 (2359) , accuracy 81.8%
cluster 6 (2313) , accuracy 86.5%
cluster 1 (528) , accuracy 89.2%
cluster 3 (1384) , accuracy 90.8%
cluster 5 (1494) , accuracy 85.4%
average accuracy on clusters 83.4% gain 11.7
```

1.0.7 Feature importance of the models & actionable variables