In [1]:
```python
from os.path import join
import xlsxwriter

import pandas as pd
import numpy as np
import sklearn
from sklearn import linear_model
from sklearn.utils import shuffle
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import LabelEncoder

from sklearn import model_selection
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import ensemble
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier

from sklearn import svm

# Adding libraries needed for plotting the significance of variables in prediction of target
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import RFE
from matplotlib import pyplot as plt

# Adding libraries needed to select most significant properties in prediction model (Logist Regression)
from sklearn.feature_selection import RFECV

# importing one hot encoder from sklearn
from sklearn.preprocessing import OneHotEncoder

import random

import statsmodels.api as sm
```

```python
import graphviz
import pydotplus
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree.export import export_text

from inspect import getmembers
```

```
C:\Users\au548008\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmodels\compat\pandas.py:56: FutureWarning: T
he pandas.core.datetools module is deprecated and will be removed in a future version. Please use the pandas.tseries mo
dule instead.
  from pandas.core import datetools
C:\Users\au548008\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\six.py:31: FutureWarning: The m
odule is deprecated in version 0.21 and will be removed in version 0.23 since we've dropped support for Python 2.7. Ple
ase rely on the official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
C:\Users\au548008\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWarning:
The sklearn.tree.export module is  deprecated in version 0.22 and will be removed in version 0.24. The corresponding cl
asses / functions should instead be imported from sklearn.tree. Anything that cannot be imported from sklearn.tree is n
ow part of the private API.
  warnings.warn(message, FutureWarning)
```

In [5]:
```python
def select_10(df_train, df_test, component, logreg, title, top_n):
    # Plotting and sorting the first 10 significant features
    df_columns = pd.DataFrame(df_train.columns)
    df_scores = pd.DataFrame(abs(logreg.coef_[0]))
    df_sign = pd.DataFrame(np.sign(logreg.coef_[0]))
    df_coef = pd.DataFrame(logreg.coef_[0])
    featureScores = pd.concat([df_columns, df_scores, df_sign, df_coef], axis=1)
    featureScores.columns = ['Person property', 'Score', 'Sign', 'Coef']
    #featureScores.nlargest(top_n, 'Score').plot.barh(x='Person property', y='Score')
    #plt.show()

    columns = []; signs = []; coefs = [];
    for i in featureScores.nlargest(top_n, 'Score')['Person property']: columns.append(i)
    for i in featureScores.nlargest(top_n, 'Score')['Sign']: signs.append(int(i))
    for i in featureScores.nlargest(top_n, 'Score')['Coef']: coefs.append(i)
    # col_sign = []
    # for i in range(len(columns)): temp = []; temp.append(columns[i]); temp.append(signs[i]); col_sign.append(temp); de

    # print(col_sign)
    # print(top_n, 'most siginifant columns:', columns)

    # Testing the model (top 10)
    new_test = df_test[columns]
    new_train = df_train[columns]
    logreg.fit(new_train, y_train)
    result_10 = logreg.score(new_test, y_test)
    print(title + ' (accuracy rate) for predicting "' + component + '" is {0:.0%} (top '.format(result_10) + str(top_n) +
    return result_10, columns, coefs

def select_top(x_train, y_train, y_test, df_train, df_test, component, algorithm, title):

    # We want to add the most important features in a list (for eavery component and every algorithm)
    freq_list = []


    # Feature selection
    sel = SelectFromModel(algorithm)
    sel.fit(x_train, y_train)

    # print('df_train.columns', df_train.columns)
    # print('sel.get_support', sel.get_support())
```

```python
    # print('sel.estimator.coef', sel.estimator.coef_[0])

    # Plotting and sorting the first 10 significant features
    df_columns = pd.DataFrame(df_train.columns)
    if title == 'Logistic Regression' or title == 'SVM':
        df_scores = pd.DataFrame(abs(sel.estimator.coef_[0]))
        df_coef = pd.DataFrame(sel.estimator.coef_[0])
    else:
        df_scores = pd.DataFrame(sel.estimator.feature_importances_)
        df_coef = pd.DataFrame(sel.estimator.feature_importances_)
    df_sign = pd.DataFrame(sel.get_support())
    featureScores = pd.concat([df_columns, df_scores, df_sign, df_coef], axis=1)
    featureScores.columns = ['Person property', 'Score', 'Support', 'Coef']
    #featureScores.nlargest(10, 'Score').plot.barh(x='Person property', y='Score')
    #plt.show()
    featureScores = featureScores[featureScores.Support == True]

    columns = []; signs = []; coefs = [];
    selected_feat = df_train.columns[(sel.get_support())]
    for i in featureScores.nlargest(len(selected_feat), 'Score')['Person property']: columns.append(i)
    for i in featureScores.nlargest(len(selected_feat), 'Score')['Coef']: coefs.append(i)
    # print(columns, coefs)

    # print('most siginifant columns:', columns)
    new_train = df_train[selected_feat]; new_test = df_test[selected_feat]

    if title == 'Logistic Regression' or title == 'SVM':
        indices = np.argsort(abs(sel.estimator.coef_))[::-1][:len(new_train.columns)]
        initial_importances = sel.estimator.coef_
    else:
        indices = np.argsort(sel.estimator.feature_importances_)[::-1][:len(new_train.columns)]
        initial_importances = sel.estimator.feature_importances_

    algorithm = algorithm.fit(new_train, y_train)
    result_top = algorithm.score(new_test, y_test)
    print(title + ' performance for predicting "' + component + '" is {0:.0%} (selected features)'.format(result_top))
    if title == 'Logistic Regression' or title == 'SVM':
        return result_top, columns, coefs
    else:
        # Sorting selected features
        # algorithm.fit(x_train, y_train)
        # indices = np.argsort(algorithm.feature_importances_)[::-1][:len(new_train.columns)]
```

```
        selected_feat = df_train.columns[indices]
        # print(selected_feat, indices)

        columns = []
        for col in selected_feat: columns.append(col)
        """
        if title == 'Decision Tree' or title == 'Decision Tree (Entropy)':
            r = export_text(algorithm, feature_names=columns)
            # print(r)
            # print( getmembers( algorithm.tree_.children_left) )
            zip(df_train.columns[algorithm.tree_.feature], algorithm.tree_.threshold, algorithm.tree_.children_left, alg

            dot_data = StringIO()
            tree.export_graphviz(algorithm, out_file=dot_data, filled=True, rounded=True, special_characters=True, \
                                 feature_names = columns, class_names=['0','1'])
            graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
            graph.write_pdf(title + ' ' + component + ".pdf")
            Image(graph.create_png())
        """
        # return result_top, df_train.columns[indices], algorithm.feature_importances_[indices]
        sorted_importances = list(initial_importances)[:len(new_train.columns)]
        sorted_importances.sort(reverse = True)
        # print('shahab', df_train.columns, indices, initial_inportances, selected_feat, sorted_importances)
        return result_top, columns, sorted_importances

def preparing_data(input_file, component, component_list, int_list, init_list):

    # df = pd.read_csv(input_file, dtype={'country': 'category', 'gender': 'category', 'Money_available': 'category', 'd
    df = pd.read_csv(input_file, dtype = {'component_natural': 'category', 'component_calories': 'category', \
                                          'component_healthy': 'category', 'component_sugar': 'category', \
                                          'component_taste': 'category', 'component_heavy': 'category', \
                                          'component_fair': 'category', 'component_organic': 'category', \
                                          'component_artificial': 'category', 'component_vitamin': 'category', \
                                          'component_gluten': 'category', 'component_lactose': 'category', \
                                          'component_vegan': 'category', 'component_regional': 'category', \
                                          'component_fill': 'category', 'component_reward': 'category', \
                                          'component_look': 'category', 'component_fresh': 'category', \
                                          'component_energy': 'category', 'category_component_unhealthy': 'category', \
                                          'category_component_origin': 'category', 'category_component_healthy': 'catego
                                          'category_component_reward': 'category', 'category_component_taste': 'category
                                          'category_component_energy': 'category', 'category_component_specific': 'categ
```

```python
new_df = df[init_list]

# removing other "components' categories
for c in component_list:
    if c != component:
        new_df = new_df.drop(c, axis = 1)

pos = new_df.loc[(new_df[component] == '1')]
neg = new_df.loc[(new_df[component] == '0')]

# Balancing data
random.seed(0)
if len(pos) <= len(neg):
    a = random.sample(range(len(neg)), len(pos)) # non duplicate indices
    final = neg.iloc[a, :]
    final = pd.concat([pos, final])
    # print(len(pos), len(neg), len(final))
    # print(final[:10][:3])
else:
    a = random.sample(range(len(pos)), len(neg)) # non duplicate indices
    final = pos.iloc[a, :]
    final = pd.concat([neg, final])
    # print(len(pos), len(neg), len(final))
    # print(final[:10][:3])
"""
# random.seed(0)
if len(pos) <= len(neg):
    # a = random.sample(range(len(neg)), len(pos)) # non duplicate indices
    # final = neg.iloc[a, :]
    final = neg.iloc[:len(pos), :]
    final = pd.concat([pos, final])
    print(len(pos), len(neg), len(final))
else:
    # a = random.sample(range(len(pos)), len(neg)) # non duplicate indices
    # final = pos.iloc[a, :]
    final = pos.iloc[:len(neg), :]
    final = pd.concat([neg, final])
    print(len(pos), len(neg), len(final))
"""

# print('columns', final_new.columns)
```

```
    x = np.array(final.drop([component], 1))
    y = np.array(final[component])

    # Split the data in train and test data
    np.random.seed(0) # the "train_test_split" function uses "np.random.seed" to split the data
    x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(x, y, test_size = 0.2)

    x_df_train = pd.DataFrame({'gender': x_train[:, 0], 'age': x_train[:, 1], 'FS_SC': x_train[:, 2], 'relationship': x_
    x_df_test = pd.DataFrame({'gender': x_test[:, 0], 'age': x_test[:, 1], 'FS_SC': x_test[:, 2], 'relationship': x_test

    return x_train, x_test, y_train, y_test, x, y, final.drop([component], 1), x_df_train, x_df_test

def predict_regression(x_train, x_test, y_train, y_test, component, df, df_train, df_test):
    # Creating the model
    # Logistic Regression
    # Using "LogisticRegression" function from "scikit-learn" library from Python. We have set below parameters for:
    # multi_class='ovr' : means we have select our training dataset based on One vs Rest
    # solver = "liblinear", because our dataset is small

    # Scale your data
    scaler = StandardScaler(); scaler.fit(df_train); X_scaled = pd.DataFrame(scaler.transform(df_train), columns = df_tr
    scaler_test = StandardScaler(); scaler_test.fit(df_test); X_scaled_test = pd.DataFrame(scaler_test.transform(df_test

    # Results for all columns
    logreg = LogisticRegression(C=1e5, solver='liblinear', multi_class='ovr', max_iter = 1000)
    logreg.fit(X_scaled, y_train)
    result_all = logreg.score(X_scaled_test, y_test)
    print('Logistic Regression (accuracy rate) for predicting "' + component + '" is {0:.0%} all columns'.format(result_

    #result_10, col_10, sign_10 = select_10(df_train, df_test, component, logreg, 'Logistic Regression', 5)

    logreg_top = LogisticRegression(C=1e5, solver='liblinear', multi_class='ovr', max_iter = 1000)
    logreg_top.fit(X_scaled, y_train)

    result_top, col_top, sign_top = select_top(x_train, y_train, y_test, df_train, df_test, component, logreg_top, 'Logi
    #if result_10 > result_top:
    #    return result_all, max(result_10, result_top), col_10, sign_10
    #else:
    #    return result_all, max(result_10, result_top), col_top, sign_top
    return result_all, result_top, col_top, sign_top

def predict_svm(x_train, x_test, y_train, y_test, component, df, df_train, df_test):
```

```python
    # SVM
    svc = svm.SVC(kernel='linear')
    svc.fit(df_train, y_train)
    """
    #######################
    coef = svc.coef_.ravel()
    top_positive_coefficients = np.argsort(coef)[-10:]
    top_negative_coefficients = np.argsort(coef)[:10]
    top_coefficients = np.hstack([top_negative_coefficients, top_positive_coefficients])
    plt.figure(figsize=(15, 5))
    colors = ['red' if c < 0 else 'blue' for c in coef[top_coefficients]]
    plt.bar(np.arange(2 * 10), coef[top_coefficients], color=colors)
    feature_names = np.array(df_train.columns)
    plt.xticks(np.arange(0, 1 + 2 * 10), feature_names[top_coefficients], rotation=60, ha='right')
    plt.show()
    #######################
    """
    # Results for all columns
    result_all = svc.score(df_test, y_test)
    print('SVM performance for predicting "' + component + '" is {0:.0%} (all columns)'.format(result_all))

    #result_10, col_10, coef_10 = select_10(df_train, df_test, component, svc, 'SVM', 5)
    result_top, col_top, coef_top = select_top(x_train, y_train, y_test, df_train, df_test, component, svc, 'SVM')
    #if result_10 > result_top:
    #    return result_all, max(result_10, result_top), col_10, coef_10
    #else:
    #    return result_all, max(result_10, result_top), col_top, coef_top
    return result_all, result_top, col_top, coef_top

def predict_svm_poly(x_train, x_test, y_train, y_test, component, df, df_train, df_test):
    # SVM Poly
    svc = svm.SVC(kernel='poly', degree = 3)
    svc.fit(df_train, y_train)
    result_all = svc.score(df_test, y_test)
    print('SVM_Polynomial for predicting "' + component + '" is {0:.0%} (all columns)'.format(result_all))
    return result_all

def predict_svm_rbf(x_train, x_test, y_train, y_test, component, df, df_train, df_test):
    # SVM RBF
    svc = svm.SVC(kernel='rbf')
    svc.fit(x_train, y_train)
    result_all = svc.score(df_test, y_test)
```

```
    print('SVM_RBF for predicting "' + component + '" is {0:.0%} (all columns)'.format(result_all))
    return result_all

def predict_svm_sigmoid(x_train, x_test, y_train, y_test, component, df, df_train, df_test):
    # SVM Sigmoid
    svc = svm.SVC(kernel='sigmoid')
    svc.fit(x_train, y_train)
    result_all = svc.score(df_test, y_test)
    print('SVM_Sigmoid for predicting "' + component + '" is {0:.0%} (all columns)'.format(result_all))
    return result_all

def predict_dt(x_train, x_test, y_train, y_test, component, df, df_train, df_test):

    # Decision Tree
    clf = DecisionTreeClassifier(random_state=0)
    clf = clf.fit(x_train, y_train)

    # tree.plot_tree(clf)
    """
    dot_data = tree.export_graphviz(clf, out_file=None, feature_names=df_train.columns, class_names=component, \
                                    filled=True, rounded=True, special_characters=True)
    graph = graphviz.Source(dot_data)
    graph.render()

    """

    # print(clf.predict(x_test))
    result_all = clf.score(x_test, y_test)
    print('Decision Tree performance for predicting "' + component + '" is {0:.0%} (all columns)'.format(result_all))

    # result_top, col_top, coef = select_top(x_train, y_train, y_test, df_train, df_test, component, clf, 'Decision Tree
    result_top, col_top, coef = select_top(x_train, y_train, y_test, df_train, df_test, component, clf, 'Decision Tree')
    # print(col_top)
    # print(coef)

    """
    #########################
    feature_select = ExtraTreesClassifier().fit(x_train, y_train)
    model = SelectFromModel(feature_select, prefit=True)
    x_new = model.transform(x_train)
    nb_features = x_new.shape[1]
    print('%i features were selected as being important:' % nb_features)
```

```
    indices = np.argsort(feature_select.feature_importances_)[::-1][:nb_features]
    col_width = len(max(df_train.columns[2+indices[f]] for f in range(nb_features))) + 5
    for f in range(nb_features):
        number = f+1
        feature_name = ''.join(df_train.columns[2+indices[f]].ljust(col_width))
        feature_importance = feature_select.feature_importances_[indices[f]]
        print('   %d.\t%s %f%%' % (number, feature_name, (feature_importance * 100)))
    features = []
    for f in sorted(np.argsort(feature_select.feature_importances_)[::-1][:nb_features]):
        features.append(df_train.columns[2+f])
    #########################
    """
    return result_all, result_top, col_top, coef

def predict_dt_ent(x_train, x_test, y_train, y_test, component, df, df_train, df_test):
    # Decision Tree Entropy
    clf = DecisionTreeClassifier(random_state=0, criterion="entropy", max_depth=3)
    clf = clf.fit(x_train, y_train)
    result_all = clf.score(x_test, y_test)
    print('Decision Tree (Entropy) performance for predicting "' + component + '" is {0:.0%} (all columns)'.format(resul
    result_top, col_top, coef = select_top(x_train, y_train, y_test, df_train, df_test, component, clf, 'Decision Tree (
    return result_all, result_top, col_top, coef

def predict_rf(x_train, x_test, y_train, y_test, component, df, df_train, df_test):
    # Random Forest
    clf = RandomForestClassifier(n_estimators = 100)
    clf = clf.fit(x_train, y_train)
    # print(clf.predict(x_test))
    result_all = clf.score(x_test, y_test)
    print('Random Forest performance for predicting "' + component + '" is {0:.0%} (all columns)'.format(result_all))
    result_top, col_top, coef = select_top(x_train, y_train, y_test, df_train, df_test, component, clf, 'Random Forest')
    return result_all, result_top, col_top, coef

def predict_et(x_train, x_test, y_train, y_test, component, df, df_train, df_test):
    # Extra Tree
    clf = ExtraTreesClassifier(random_state=0)
    clf = clf.fit(x_train, y_train)
    # print(clf.predict(x_test))
    result_all = clf.score(x_test, y_test)
    print('Extra Trees performance for predicting "' + component + '" is {0:.0%} (all columns)'.format(result_all))
    result_top, col_top, coef = select_top(x_train, y_train, y_test, df_train, df_test, component, clf, 'ExtraTrees')
    return result_all, result_top, col_top, coef
```

```python
def write_list(output_file, info_list, header):
    wb = xlsxwriter.Workbook(output_file); ws = wb.add_worksheet()
    for i in range(len(header)): ws.write(0, i, header[i]);
    for i in range(len(info_list)):
        for j in range(len(info_list[i])): ws.write(i + 1, j , info_list[i][j])
    wb.close()
    print('finished')
```

In [6]:
```
my_path = r'C:\Aarhus\Internship\Morten\17\Data'
input_file = 'predict_reason_ML_with_component_categories.csv'
predict_file = 'predict_component_results_reason_context_person.xlsx'
header = ["component's category", 'method', 'Accuracy Rate (all)', 'Accuracy Rate (top)', 'Significant Features']

freq_file = 'freq_person_context_reason.csv'
freq_header = ["component's category", 'method', 'Feature']
freq_list = []

component = ['category_component_unhealthy', 'category_component_origin', 'category_component_healthy', \
             'category_component_reward', 'category_component_taste', 'category_component_energy', \
             'category_component_specific']

# component = ['category_component_unhealthy']

int_list = ['age', 'FS_SC', 'Sports', 'Last_meal']

init_list = ['gender', 'age', 'FS_SC', 'relationship', 'people_in_HH', 'Money_available', 'Sports', 'diet', \
             'timing_morning', 'timing_afternoon', 'timing_evening', 'timing_night', 'situation_home', 'situation_work',
             'situation_go', 'situation_FnF', 'situation_PoP', 'situation_leisure', 'situation_sports', 'situation_learn
             'situation_read', 'situation_TV', 'situation_PC', 'withc_alone', 'withc_partner', 'withc_friends', \
             'withc_colleagues', 'withc_family', 'withc_acquian', 'withc_children', 'country', \
             'reason_nutrition', 'reason_energy', 'reason_hunger', 'reason_break', 'reason_pastime', 'reason_mood', \
             'reason_stress', 'reason_diet', 'reason_enjoy', 'reason_relax', 'reason_reward', 'reason_habit', 'reason_fe
             'category_component_unhealthy', 'category_component_origin', 'category_component_healthy', \
             'category_component_reward', 'category_component_taste', 'category_component_energy', \
             'category_component_specific']

predict_list = []

for c in component:
    x_train, x_test, y_train, y_test, x, y, df, df_train, df_test = preparing_data(join(my_path, input_file), c, compone

    # Logistic Regression
    acc, acc_top, cols, coef = predict_regression(x_train, x_test, y_train, y_test, c, df, df_train, df_test)
    row = []; row.append(c); row.append('Logistic Regression'); row.append(acc); row.append(acc_top)
    col_score = []; col_score_str = ''
    for i in range(len(cols)):
        temp = []; temp.append(cols[i]); col_score_str += cols[i] + ': '; temp.append(coef[i])
        col_score_str += str(round(coef[i] * 100, 2)).format(1.0/3.0) + ', '; col_score.append(temp); del temp
        temp = []; temp.append(c); temp.append('Logistic Regression'); temp.append(cols[i]); freq_list.append(temp); del
```

```
    row.append(col_score_str[:-2]); predict_list.append(row); del row

    # SVM
    acc, acc_top, cols, coef = predict_svm(x_train, x_test, y_train, y_test, c, df, df_train, df_test)
    row = []; row.append(c); row.append('SVM'); row.append(acc); row.append(acc_top)
    col_score = []; col_score_str = ''
    for i in range(len(cols)):
        temp = []; temp.append(cols[i]); col_score_str += cols[i] + ': '; temp.append(coef[i])
        col_score_str += str(round(coef[i] * 100, 2)).format(1.0/3.0) + ', '; col_score.append(temp); del temp
        temp = []; temp.append(c); temp.append('SVM'); temp.append(cols[i]); freq_list.append(temp); del temp
    row.append(col_score_str[:-2]); predict_list.append(row); del row

    """
    # SVM Poly
    acc = predict_svm_poly(x_train, x_test, y_train, y_test, c, df, df_train, df_test)
    row = []; row.append(c); row.append('SVM (Polynomial)'); row.append(acc); row.append(''); predict_list.append(row); d

    # SVM RBF
    acc = predict_svm_rbf(x_train, x_test, y_train, y_test, c, df, df_train, df_test)
    row = []; row.append(c); row.append('SVM (RBF)'); row.append(acc); row.append(''); predict_list.append(row); del row

    # SVM Sigmoid
    acc = predict_svm_sigmoid(x_train, x_test, y_train, y_test, c, df, df_train, df_test)
    row = []; row.append(c); row.append('SVM (Sigmoid)'); row.append(acc); row.append(''); predict_list.append(row); del
    """

    # Decision Tree
    acc, acc_top, cols, coef = predict_dt(x_train, x_test, y_train, y_test, c, df, df_train, df_test)
    row = []; row = []; row.append(c); row.append('Decision Tree'); row.append(acc); row.append(acc_top)
    col_score = []; col_score_str = ''
    for i in range(len(cols)):
        temp = []; temp.append(cols[i]); col_score_str += cols[i] + ': '; temp.append(coef[i])
        col_score_str += str(round(coef[i] * 100, 2)).format(1.0/3.0) + '%, '; col_score.append(temp); del temp
        temp = []; temp.append(c); temp.append('Decision Tree'); temp.append(cols[i]); freq_list.append(temp); del temp
    row.append(col_score_str[:-2]); predict_list.append(row); del row

    # Decison Tree Entropy
    acc, acc_top, cols, coef = predict_dt_ent(x_train, x_test, y_train, y_test, c, df, df_train, df_test)
    row = []; row = []; row.append(c); row.append('Decision Tree (Entropy)'); row.append(acc); row.append(acc_top)
    col_score = []; col_score_str = ''
    for i in range(len(cols)):
        temp = []; temp.append(cols[i]); col_score_str += cols[i] + ': '; temp.append(coef[i])
```

```
        col_score_str += str(round(coef[i] * 100, 2)).format(1.0/3.0) + '%, '; col_score.append(temp); del temp
        temp = []; temp.append(c); temp.append('Decision Tree (Entropy)'); temp.append(cols[i]); freq_list.append(temp);
    row.append(col_score_str[:-2]); predict_list.append(row); del row

    # Random Forest
    acc, acc_top, cols, coef = predict_rf(x_train, x_test, y_train, y_test, c, df, df_train, df_test);
    row = []; row.append(c); row.append('Random Forest'); row.append(acc); row.append(acc_top)
    col_score = []; col_score_str = ''
    for i in range(len(cols)):
        temp = []; temp.append(cols[i]); col_score_str += cols[i] + ': '; temp.append(coef[i])
        col_score_str += str(round(coef[i] * 100, 2)).format(1.0/3.0) + '%, '; col_score.append(temp); del temp
        temp = []; temp.append(c); temp.append('Random Forest'); temp.append(cols[i]); freq_list.append(temp); del temp
    row.append(col_score_str); predict_list.append(row); del row

    # Extra Tree
    acc, acc_top, cols, coef = predict_et(x_train, x_test, y_train, y_test, c, df, df_train, df_test);
    row = []; row.append(c); row.append('Extra Tree'); row.append(acc); row.append(acc_top)
    col_score = []; col_score_str = ''
    for i in range(len(cols)):
        temp = []; temp.append(cols[i]); col_score_str += cols[i] + ': '; temp.append(coef[i])
        col_score_str += str(round(coef[i] * 100, 2)).format(1.0/3.0) + '%, '; col_score.append(temp); del temp
        temp = []; temp.append(c); temp.append('Extra Tree'); temp.append(cols[i]); freq_list.append(temp); del temp
    row.append(col_score_str); predict_list.append(row); del row

    temp_list = ['']
    predict_list.append(temp_list)
```

```
Logistic Regression (accuracy rate) for predicting "category_component_unhealthy" is 60% all columns
Logistic Regression performance for predicting "category_component_unhealthy" is 55% (selected features)
SVM performance for predicting "category_component_unhealthy" is 60% (all columns)
SVM performance for predicting "category_component_unhealthy" is 57% (selected features)
Decision Tree performance for predicting "category_component_unhealthy" is 62% (all columns)
Decision Tree performance for predicting "category_component_unhealthy" is 57% (selected features)
Decision Tree (Entropy) performance for predicting "category_component_unhealthy" is 62% (all columns)
Decision Tree (Entropy) performance for predicting "category_component_unhealthy" is 53% (selected features)
Random Forest performance for predicting "category_component_unhealthy" is 67% (all columns)
Random Forest performance for predicting "category_component_unhealthy" is 57% (selected features)
Extra Trees performance for predicting "category_component_unhealthy" is 67% (all columns)
ExtraTrees performance for predicting "category_component_unhealthy" is 65% (selected features)
```

```
Logistic Regression (accuracy rate) for predicting "category_component_origin" is 59% all columns
Logistic Regression performance for predicting "category_component_origin" is 54% (selected features)
SVM performance for predicting "category_component_origin" is 54% (all columns)
SVM performance for predicting "category_component_origin" is 57% (selected features)
Decision Tree performance for predicting "category_component_origin" is 48% (all columns)
Decision Tree performance for predicting "category_component_origin" is 57% (selected features)
Decision Tree (Entropy) performance for predicting "category_component_origin" is 43% (all columns)
Decision Tree (Entropy) performance for predicting "category_component_origin" is 59% (selected features)
Random Forest performance for predicting "category_component_origin" is 56% (all columns)
Random Forest performance for predicting "category_component_origin" is 67% (selected features)
Extra Trees performance for predicting "category_component_origin" is 56% (all columns)
ExtraTrees performance for predicting "category_component_origin" is 56% (selected features)
Logistic Regression (accuracy rate) for predicting "category_component_healthy" is 62% all columns
Logistic Regression performance for predicting "category_component_healthy" is 59% (selected features)
SVM performance for predicting "category_component_healthy" is 63% (all columns)
SVM performance for predicting "category_component_healthy" is 60% (selected features)
Decision Tree performance for predicting "category_component_healthy" is 59% (all columns)
Decision Tree performance for predicting "category_component_healthy" is 58% (selected features)
Decision Tree (Entropy) performance for predicting "category_component_healthy" is 60% (all columns)
Decision Tree (Entropy) performance for predicting "category_component_healthy" is 56% (selected features)
Random Forest performance for predicting "category_component_healthy" is 62% (all columns)
Random Forest performance for predicting "category_component_healthy" is 61% (selected features)
Extra Trees performance for predicting "category_component_healthy" is 64% (all columns)
ExtraTrees performance for predicting "category_component_healthy" is 60% (selected features)
Logistic Regression (accuracy rate) for predicting "category_component_reward" is 71% all columns
Logistic Regression performance for predicting "category_component_reward" is 67% (selected features)
SVM performance for predicting "category_component_reward" is 68% (all columns)
SVM performance for predicting "category_component_reward" is 71% (selected features)
Decision Tree performance for predicting "category_component_reward" is 69% (all columns)
Decision Tree performance for predicting "category_component_reward" is 63% (selected features)
Decision Tree (Entropy) performance for predicting "category_component_reward" is 65% (all columns)
Decision Tree (Entropy) performance for predicting "category_component_reward" is 54% (selected features)
Random Forest performance for predicting "category_component_reward" is 71% (all columns)
Random Forest performance for predicting "category_component_reward" is 63% (selected features)
Extra Trees performance for predicting "category_component_reward" is 69% (all columns)
ExtraTrees performance for predicting "category_component_reward" is 70% (selected features)
Logistic Regression (accuracy rate) for predicting "category_component_taste" is 61% all columns
Logistic Regression performance for predicting "category_component_taste" is 55% (selected features)
SVM performance for predicting "category_component_taste" is 60% (all columns)
SVM performance for predicting "category_component_taste" is 53% (selected features)
Decision Tree performance for predicting "category_component_taste" is 58% (all columns)
Decision Tree performance for predicting "category_component_taste" is 60% (selected features)
```

```
Decision Tree (Entropy) performance for predicting "category_component_taste" is 58% (all columns)
Decision Tree (Entropy) performance for predicting "category_component_taste" is 53% (selected features)
Random Forest performance for predicting "category_component_taste" is 64% (all columns)
Random Forest performance for predicting "category_component_taste" is 62% (selected features)
Extra Trees performance for predicting "category_component_taste" is 63% (all columns)
ExtraTrees performance for predicting "category_component_taste" is 61% (selected features)
Logistic Regression (accuracy rate) for predicting "category_component_energy" is 59% all columns
Logistic Regression performance for predicting "category_component_energy" is 55% (selected features)
SVM performance for predicting "category_component_energy" is 58% (all columns)
SVM performance for predicting "category_component_energy" is 55% (selected features)
Decision Tree performance for predicting "category_component_energy" is 56% (all columns)
Decision Tree performance for predicting "category_component_energy" is 55% (selected features)
Decision Tree (Entropy) performance for predicting "category_component_energy" is 56% (all columns)
Decision Tree (Entropy) performance for predicting "category_component_energy" is 55% (selected features)
Random Forest performance for predicting "category_component_energy" is 60% (all columns)
Random Forest performance for predicting "category_component_energy" is 59% (selected features)
Extra Trees performance for predicting "category_component_energy" is 60% (all columns)
ExtraTrees performance for predicting "category_component_energy" is 60% (selected features)
Logistic Regression (accuracy rate) for predicting "category_component_specific" is 55% all columns
Logistic Regression performance for predicting "category_component_specific" is 48% (selected features)
SVM performance for predicting "category_component_specific" is 55% (all columns)
SVM performance for predicting "category_component_specific" is 56% (selected features)
Decision Tree performance for predicting "category_component_specific" is 61% (all columns)
Decision Tree performance for predicting "category_component_specific" is 65% (selected features)
Decision Tree (Entropy) performance for predicting "category_component_specific" is 54% (all columns)
Decision Tree (Entropy) performance for predicting "category_component_specific" is 57% (selected features)
Random Forest performance for predicting "category_component_specific" is 64% (all columns)
Random Forest performance for predicting "category_component_specific" is 68% (selected features)
Extra Trees performance for predicting "category_component_specific" is 67% (all columns)
ExtraTrees performance for predicting "category_component_specific" is 64% (selected features)
```

In [5]:

```python
# print(predict_list)

write_list(join(my_path, predict_file), predict_list, header)
```

```
finished
```

In [8]:

```python
write_list(join(my_path, freq_file), freq_list, freq_header)
```

```
finished
```

In [177]:
```
np.random.seed(0)
#some random function
a = np.random.randint(10699, size=5)
print (a)
```

```
[2732 9845 3264 4859 9225]
```

In [48]:
```
random.seed(0)
random.sample(range(15), 15)
```

Out[48]:
```
[13, 6, 12, 14, 0, 4, 8, 7, 3, 2, 11, 5, 9, 10, 1]
```

In [52]:
```
temp = ''
for i in range(1, 14):
    temp  += "'" + str(i) + "':" + str(i)  + ', '
print(temp)
```

```
'1':1, '2':2, '3':3, '4':4, '5':5, '6':6, '7':7, '8':8, '9':9, '10':10, '11':11, '12':12, '13':13,
```

In [11]:
```python
after_list = ['gender', 'age', 'FS_SC', 'relationship', 'people_in_HH', 'Money_available', 'Sports', 'diet', \
              'timing_morning', 'timing_afternoon', 'timing_evening', 'timing_night', 'situation_home', 'situation_work',
              'situation_go', 'situation_FnF', 'situation_PoP', 'situation_leisure', 'situation_sports', 'situation_learn
              'situation_read', 'situation_TV', 'situation_PC', 'withc_alone', 'withc_partner', 'withc_friends', \
              'withc_colleagues', 'withc_family', 'withc_acquian', 'withc_children', 'country', \
              'reason_nutrition', 'reason_energy', 'reason_hunger', 'reason_break', 'reason_pastime', 'reason_mood', \
              'reason_stress', 'reason_diet', 'reason_enjoy', 'reason_relax', 'reason_reward', 'reason_habit', 'reason_fe
temp = ''; temp1 = ''
for i in range(len(after_list)):
    temp += "'" + after_list[i] + "': x_train[:, " + str(i) + "], "
    temp1 += "'" + after_list[i] + "': x_test[:, " + str(i) + "], "
print(temp)
print(temp1)
```

```
'gender': x_train[:, 0], 'age': x_train[:, 1], 'FS_SC': x_train[:, 2], 'relationship': x_train[:, 3], 'people_in_HH': x
_train[:, 4], 'Money_available': x_train[:, 5], 'Sports': x_train[:, 6], 'diet': x_train[:, 7], 'timing_morning': x_tra
in[:, 8], 'timing_afternoon': x_train[:, 9], 'timing_evening': x_train[:, 10], 'timing_night': x_train[:, 11], 'situati
on_home': x_train[:, 12], 'situation_work': x_train[:, 13], 'situation_go': x_train[:, 14], 'situation_FnF': x_train[:,
15], 'situation_PoP': x_train[:, 16], 'situation_leisure': x_train[:, 17], 'situation_sports': x_train[:, 18], 'situati
on_learn': x_train[:, 19], 'situation_read': x_train[:, 20], 'situation_TV': x_train[:, 21], 'situation_PC': x_train[:,
22], 'withc_alone': x_train[:, 23], 'withc_partner': x_train[:, 24], 'withc_friends': x_train[:, 25], 'withc_colleague
s': x_train[:, 26], 'withc_family': x_train[:, 27], 'withc_acquian': x_train[:, 28], 'withc_children': x_train[:, 29],
'country': x_train[:, 30], 'reason_nutrition': x_train[:, 31], 'reason_energy': x_train[:, 32], 'reason_hunger': x_trai
n[:, 33], 'reason_break': x_train[:, 34], 'reason_pastime': x_train[:, 35], 'reason_mood': x_train[:, 36], 'reason_stre
ss': x_train[:, 37], 'reason_diet': x_train[:, 38], 'reason_enjoy': x_train[:, 39], 'reason_relax': x_train[:, 40], 're
ason_reward': x_train[:, 41], 'reason_habit': x_train[:, 42], 'reason_feeling': x_train[:, 43],
'gender': x_test[:, 0], 'age': x_test[:, 1], 'FS_SC': x_test[:, 2], 'relationship': x_test[:, 3], 'people_in_HH': x_tes
t[:, 4], 'Money_available': x_test[:, 5], 'Sports': x_test[:, 6], 'diet': x_test[:, 7], 'timing_morning': x_test[:, 8],
'timing_afternoon': x_test[:, 9], 'timing_evening': x_test[:, 10], 'timing_night': x_test[:, 11], 'situation_home': x_t
est[:, 12], 'situation_work': x_test[:, 13], 'situation_go': x_test[:, 14], 'situation_FnF': x_test[:, 15], 'situation_
PoP': x_test[:, 16], 'situation_leisure': x_test[:, 17], 'situation_sports': x_test[:, 18], 'situation_learn': x_test
[:, 19], 'situation_read': x_test[:, 20], 'situation_TV': x_test[:, 21], 'situation_PC': x_test[:, 22], 'withc_alone':
x_test[:, 23], 'withc_partner': x_test[:, 24], 'withc_friends': x_test[:, 25], 'withc_colleagues': x_test[:, 26], 'with
c_family': x_test[:, 27], 'withc_acquian': x_test[:, 28], 'withc_children': x_test[:, 29], 'country': x_test[:, 30], 'r
eason_nutrition': x_test[:, 31], 'reason_energy': x_test[:, 32], 'reason_hunger': x_test[:, 33], 'reason_break': x_test
[:, 34], 'reason_pastime': x_test[:, 35], 'reason_mood': x_test[:, 36], 'reason_stress': x_test[:, 37], 'reason_diet':
x_test[:, 38], 'reason_enjoy': x_test[:, 39], 'reason_relax': x_test[:, 40], 'reason_reward': x_test[:, 41], 'reason_ha
bit': x_test[:, 42], 'reason_feeling': x_test[:, 43],
```

In [ ]: