AnalyseHeath

January 15, 2023

1 Logistice Regression for health

1.1 Comprehension of caracteristiques

It will use real-world data that contains detailed nutritional information about foods for people with diabetes. The goal is to determine whether a diabetic patient should choose more often, less often, or in moderation for a specific food item based on the nutritional information in the dataset.

1.2 Principale objectif and mission

I would like to work for a hospital as a data scientist and I have no data in this field. So I took the data from the training because I am also a student for the moment. The goal is to determine whether a diabetic patient should choose more often, less often, or in moderation for a specific food item based on the nutritional information in the dataset. My analysis objective is mainly to retrieve data related to diabetes and to analyze : * Train and fine-tune logistic regression models * Interpret trained logistic regression models * Evaluate trained logistic regression models * The explanatory factors of the degradation and concentration of the diabetes rate in the blood * Make recommandation for next setps

1.2.1 Importation for librairies and preparation

[34]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split, learning_curve
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix, precision_recall_fscore_support, precision_score,␣
 ↪recall_score
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[35]:
```
dataset_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
 ↪cloud/IBM-ML241EN-SkillsNetwork/labs/datasets/food_items.csv"
food_df = pd.read_csv(dataset_url)
```

Observations of variables and data values

[36]:
```
food_df.head(10)
```

[36]:
```
   Calories  Total Fat  Saturated Fat  Monounsaturated Fat  \
0     149.0          0            0.0                  0.0
1     123.0          0            0.0                  0.0
2     150.0          0            0.0                  0.0
3     110.0          0            0.0                  0.0
4     143.0          0            0.0                  0.0
5     110.0          0            0.0                  0.0
6     142.0          0            0.0                  0.0
7     102.0          0            0.0                  0.0
8     145.0          0            0.0                  0.0
9     171.0          0            0.0                  0.0

   Polyunsaturated Fat  Trans Fat  Cholesterol  Sodium  Total Carbohydrate  \
0                  0.0        0.0            0     9.0                 9.8
1                  0.0        0.0            0     5.0                 6.6
2                  0.0        0.0            0     4.0                11.4
3                  0.0        0.0            0     6.0                 7.0
4                  0.0        0.0            0     7.0                13.1
5                  0.0        0.0            0     6.0                 7.0
6                  0.0        0.0            0    12.0                10.6
7                  0.0        0.0            0    13.0                 5.0
8                  0.0        0.0            0    17.0                11.0
9                  0.0        0.0            0     8.0                13.7

   Dietary Fiber  Sugars  Sugar Alcohol  Protein  Vitamin A  Vitamin C  \
0            0.0     0.0              0      1.3          0          0
1            0.0     0.0              0      0.8          0          0
2            0.0     0.0              0      1.3          0          0
3            0.0     0.0              0      0.8          0          0
4            0.0     0.0              0      1.0          0          0
5            0.0     0.0              0      0.8          0          0
6            0.0     0.0              0      1.2          0          0
7            0.0     0.0              0      0.7          0          0
8            0.0     0.0              0      1.2          0          0
9            0.0     0.0              0      2.5          0          0

   Calcium  Iron            class
0        0     0  'In Moderation'
1        0     0  'In Moderation'
2        0     0  'In Moderation'
3        0     0  'In Moderation'
4        0     0  'In Moderation'
5        0     0  'In Moderation'
6        0     0  'In Moderation'
7        0     0  'In Moderation'
```

```
8        0     0  'In Moderation'
9        0     0  'In Moderation'
```

1.2.2 Data mean

- Calories Calories of patients;
- Total Fat Total Fat;
- Saturated Fat Satured ;
- Monounsaturated Fat Monounsatured ;
- Polyunsaturated Fat Polyunsatured ;
- Trans Fat Trans Fat ;
- Cholesterol Presence of cholesterol ;
- Sodium Sodium quantity ;
- Total Carbohydrate Carbonnate quantity ;
- Dietary Fiber fievre ;
- Sugars Sugars quantity ;
- Sugar Alcohol Presence of Alcool ;
- Protein Proteine ;
- Vitamin A Vitamin ;
- Vitamin C Vitamin ;
- Calcium Calorie ;
- Iron Calorie ;
- class predictor and explain variable

1.2.3 We explain categories of class with this variables

- Calories Calories of patients;
- Total Fat Total Fat;
- Saturated Fat Satured ;
- Monounsaturated Fat Monounsatured ;
- Polyunsaturated Fat Polyunsatured ;
- Trans Fat Trans Fat ;
- Cholesterol Presence of cholesterol ;
- Sodium Sodium quantity ;
- Total Carbohydrate Carbonnate quantity ;
- Dietary Fiber fievre ;
- Sugars Sugars quantity ;
- Sugar Alcohol Presence of Alcool ;
- Protein Proteine ;
- Vitamin A Vitamin ;
- Vitamin C Vitamin ;
- Calcium Calorie ;
- Iron Calorie ;

1.2.4 Type of data

[24]:
```
food_df.dtypes
```

[24]:
```
Calories                  float64
Total Fat                   int64
Saturated Fat             float64
Monounsaturated Fat       float64
Polyunsaturated Fat       float64
Trans Fat                 float64
Cholesterol                 int64
Sodium                    float64
Total Carbohydrate        float64
Dietary Fiber             float64
Sugars                    float64
Sugar Alcohol               int64
Protein                   float64
Vitamin A                   int64
Vitamin C                   int64
Calcium                     int64
Iron                        int64
class                      object
dtype: object
```

1.2.5 Description statistics

[37]:
```
food_df.describe()
```

[37]:
```
           Calories     Total Fat  Saturated Fat  Monounsaturated Fat  \
count  13260.000000  13260.000000   13260.000000         13260.000000
mean     133.861086      4.475264       1.450617             0.338069
std       94.227650      5.386340       2.410318             1.345852
min        0.000000      0.000000       0.000000             0.000000
25%       70.000000      0.000000       0.000000             0.000000
50%      120.000000      3.000000       0.500000             0.000000
75%      180.000000      7.000000       2.000000             0.000000
max     2210.000000     43.000000      22.000000            40.000000

       Polyunsaturated Fat     Trans Fat   Cholesterol        Sodium  \
count         13260.000000  13260.000000  13260.000000  13260.000000
mean              0.254660      0.047459      8.857692    241.867142
std               2.230586      0.321402     20.976530    272.284363
min               0.000000      0.000000      0.000000      0.000000
25%               0.000000      0.000000      0.000000     40.000000
50%               0.000000      0.000000      0.000000    135.000000
75%               0.000000      0.000000     10.000000    370.000000
max             235.000000     11.000000    450.000000   2431.000000
```

```
       Total Carbohydrate  Dietary Fiber        Sugars  Sugar Alcohol  \
count        13260.000000   13260.000000  13260.000000   13260.000000
mean            18.232020       1.602971      6.645234       0.117949
std             14.786316       3.363879      8.328465       1.121529
min              0.000000       0.000000      0.000000       0.000000
25%              5.000000       0.000000      0.000000       0.000000
50%             17.000000       1.000000      3.000000       0.000000
75%             27.000000       2.000000     11.000000       0.000000
max            270.000000     305.000000    115.000000      31.000000

            Protein     Vitamin A     Vitamin C       Calcium          Iron
count  13260.000000  13260.000000  13260.000000  13260.000000  13260.000000
mean       4.661333      6.287632      6.741855      5.175264      5.235671
std        5.611143     18.374191     23.785100      8.779637      9.119459
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        1.000000      0.000000      0.000000      0.000000      0.000000
50%        3.000000      0.000000      0.000000      2.000000      2.000000
75%        7.000000      6.000000      2.000000      6.000000      8.000000
max       70.000000    622.000000   1000.000000    110.000000    170.000000
```

1.3 Variante models and classification choice

After presentation we are any missing data. As we can see from the above output, this dataset contains 17 nutrient categories about each food item. These categories include Calories, Total Fat, Protein, Sugar, etc., and are listed as numeric variables. As such, we only need to scale them for training our logistic regression model so that we can compare our feature coefficients directly.

We have three labels meaning our logistic regression model will be multinomial with three classes.

A multinomial logistic regression is a generalized logistic regression model which generates a probability distribution over all classes, based on the logits or exponentiated log-odds calculated for each class (usually more than two). We can try : * logistic regression * Decisions Tree * SVMs * KNNs * Ensemble learning like Random Forest

But we start with logistic regression. Also note that a multinomial logistic regression model is different from the `one-vs-rest` binary logistic regression. For `one-vs-rest` schema, you need to train an independent classifier for each class. For example, you need a `More Often` classifier to differentiate a food item between `More Often` and `Not More Often` (or, `In Moderation` and `Less Often`).

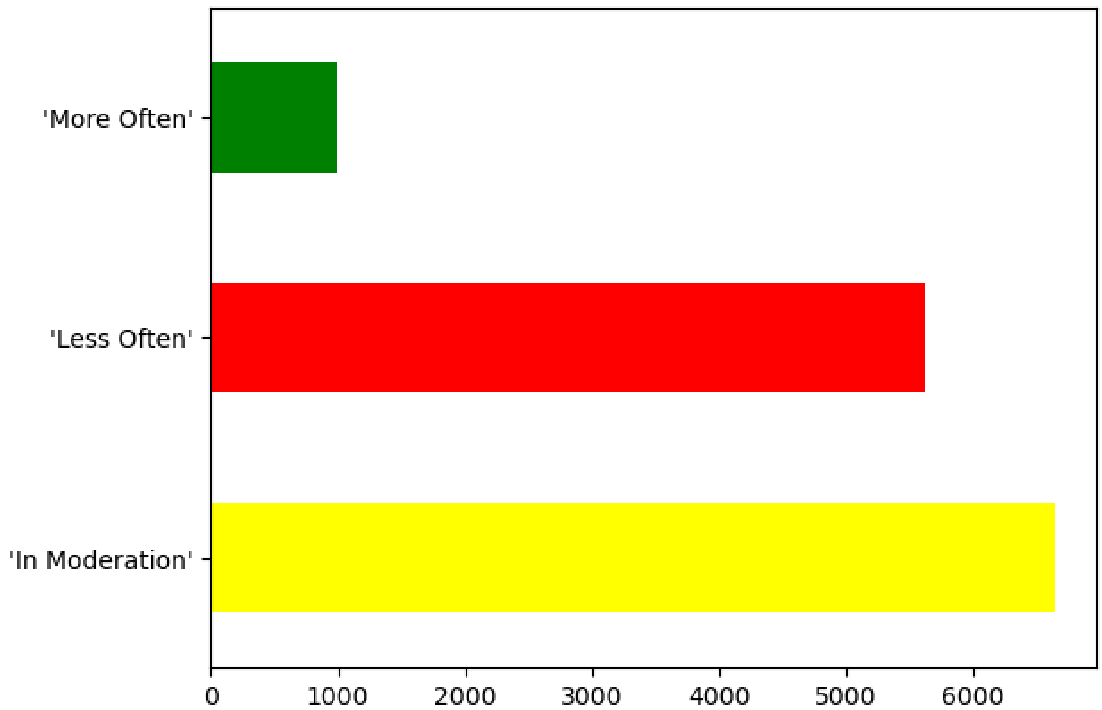
1.3.1 Predictor

[38]:
```
food_df['class'].value_counts(normalize=True)
```

[38]:
```
'In Moderation'    0.501433
'Less Often'       0.423906
'More Often'       0.074661
Name: class, dtype: float64
```

[39]:
```
food_df['class'].value_counts().plot.barh(color=['yellow', 'red', 'green'])
```

[39]: <AxesSubplot:>



We can see on the graph above, this data set has three classes: "In moderation", "Less often" and "More often". All three labels are unbalanced. For diabetic patients, most foods fall into the "In moderation" and "Less often" categories. This makes managing the diabetic diet very difficult. Therefore, we could build a machine learning model to help patients choose their foods.

We have three labels meaning our logistic regression model will be multinomial with three classes.

1.4 Feature engenneering

Fortunately, all feature columns are numeric so we just need to scale them. Here we use the `MinMaxScaler` provided by `sklearn` for scaling.

[40]:
```
#MinMaxScaler object
scaler = MinMaxScaler()
```

[41]:
```
#Application with data after identification
X_raw = food_df.iloc[:, :-1]
y_raw = food_df.iloc[:, -1:]
X = scaler.fit_transform(X_raw)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/preprocessing/data.py:323: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

For the target variable y, let's use the LabelEncoder provided by sklearn to encode its three class values.

[42]:
```python
# LabelEncoder object
label_encoder = LabelEncoder()
```

[43]:
```python
#Aplplication of data
y = label_encoder.fit_transform(y_raw.values.ravel())
```

The encoded target variable will only contain values 0=In Moderation, 1=Less Often, 2=More Often.

[44]:
```python
#Note of variable y 0
np.unique(y, return_counts=True)
```

[44]:
```
(array([0, 1, 2]), array([6649, 5621,  990]))
```

1.5 Preparation of modelisation

[45]:
```python
# First, let's split the training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪stratify=y, random_state = 10)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:1609: DeprecationWarning: `np.int` is
a deprecated alias for the builtin `int`. To silence this warning, use `int` by
itself. Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the
precision. If you wish to review your current use, check the release note link
for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  return floored.astype(np.int)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:1609: DeprecationWarning: `np.int` is
a deprecated alias for the builtin `int`. To silence this warning, use `int` by
itself. Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the
precision. If you wish to review your current use, check the release note link
for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  return floored.astype(np.int)
```

[46]:
```
# L2 penalty
penalty= 'l2'
# Type of classification problem is multinomial
multi_class = 'multinomial'
# Use lbfgs for L2 penalty and multinomial classes
solver = 'lbfgs'
# Max iteration = 1000
max_iter = 1000
```

[47]:
```
# Define a logistic regression model with above arguments
l2_model = LogisticRegression(random_state=10, penalty=penalty,␣
 ↪multi_class=multi_class, solver=solver, max_iter=max_iter)
```

[48]:
```
# Creation of mofel
l2_model.fit(X_train, y_train)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/utils/fixes.py:357: DeprecationWarning: distutils Version
classes are deprecated. Use packaging.version instead.
  if _joblib.__version__ >= LooseVersion('0.12'):
```

[48]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=1000, multi_class='multinomial',
          n_jobs=None, penalty='l2', random_state=10, solver='lbfgs',
          tol=0.0001, verbose=0, warm_start=False)
```

[49]:
```
l2_preds = l2_model.predict(X_test)
```

[50]:
```
l2_preds
```

[50]:
```
array([1, 0, 2, …, 0, 0, 2])
```

1.6 Evaluation

[51]:
```
# Function of evaluation
def evaluate_metrics(yt, yp):
    results_pos = {}
    results_pos['accuracy'] = accuracy_score(yt, yp)
    precision, recall, f_beta, _ = precision_recall_fscore_support(yt, yp)
    results_pos['recall'] = recall
    results_pos['precision'] = precision
    results_pos['f1score'] = f_beta
    return results_pos
```

[52]:
```
evaluate_metrics(y_test, l2_preds)
```

[52]:
```
{'accuracy': 0.7669683257918553,
 'recall': array([0.87067669, 0.73220641, 0.26767677]),
 'precision': array([0.72194514, 0.83047427, 0.92982456]),
 'f1score': array([0.78936605, 0.77825059, 0.41568627])}
```

As we can see from the above evaluation results, the logistic regression model has relatively good performance on this multinomial classification task. The overall accuracy is around `0.76` and the f1score is around `0.7`. Note that for **`recall`**, **`precision`**, and **`f1score`**, we output the values for each class to see how the model performs on an individual class. And, we can see from the results, the recall for `class=2` (More often) is not very good. This is actually a common problem called imbalanced classification challenge. We will introduce solution to this problem later in this course.

1.7 We can try again for next performance

[53]:
```
# L1 penalty
penalty= 'l1'
# Our classification problem is multinomial
multi_class = 'multinomial'
# Use saga for L1 penalty and multinomial classes
solver = 'saga'
# Max iteration = 1000
max_iter = 1000
```

[54]:
```
# Define a logistic regression model with above arguments
l2_model = LogisticRegression(random_state=120, penalty=penalty,␣
 ↪multi_class=multi_class, solver=solver, max_iter = 1000)
```

[55]:
```
l2_model.fit(X_train, y_train)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/utils/fixes.py:357: DeprecationWarning: distutils Version
classes are deprecated. Use packaging.version instead.
  if _joblib.__version__ >= LooseVersion('0.12'):
```

[55]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=1000, multi_class='multinomial',
          n_jobs=None, penalty='l1', random_state=120, solver='saga',
          tol=0.0001, verbose=0, warm_start=False)
```

[56]:
```
l2_preds = l2_model.predict(X_test)
```

[57]:
```
l2_preds
```

[57]:
```
array([1, 0, 2, …, 0, 0, 2])
```

1.8 Application for interpretation

[58]:
```
odd_ratios = l2_model.predict_proba(X_test[:1, :])[0]
odd_ratios
```

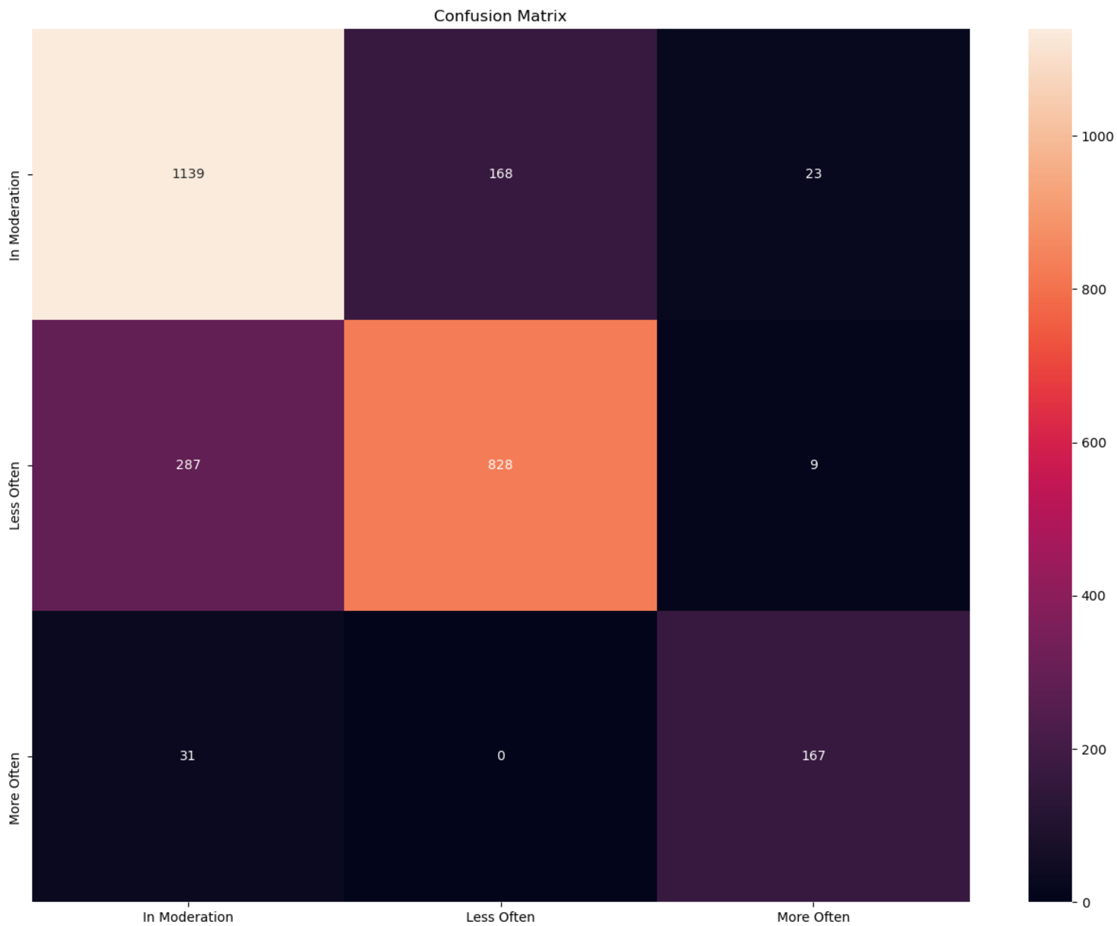
[58]:
```
array([1.78052047e-01, 8.21928521e-01, 1.94312960e-05])
```

[59]:
```
evaluate_metrics(y_test, l2_preds)
```

[59]:
```
{'accuracy': 0.8046757164404224,
 'recall': array([0.85639098, 0.7366548 , 0.84343434]),
 'precision': array([0.78174331, 0.8313253 , 0.83919598]),
 'f1score': array([0.81736634, 0.78113208, 0.84130982])}
```

1.9 We are amelioration

[60]:
```
## Make confusion matrix
cf = confusion_matrix(y_test, l2_preds)
plt.figure(figsize=(16, 12))
ax = sns.heatmap(cf, annot=True, fmt="d", xticklabels=["In Moderation", "Less␣
↪Often", "More Often"],
                 yticklabels=["In Moderation", "Less Often", "More Often"])
ax.set(title="Confusion Matrix");
```



1.9.1 Interpret logistic regression models

One way to interpret logistic regression models is by analyzing feature coefficients. Although it may not be as effective as the regular linear regression models because the logistic regression model has a sigmoid function, we can still get a sense for the importance or impact of each feature.

1.10 Presentation

We find that, unhealthy nutrients such as saturated fat, sugars, cholesterol, total fat, etc. have high positive coefficients. Foods containing unhealthy nutrients will have higher coefficients and will be more likely to be classified as "Less often".

[61]:
```
## Coefficients
l2_model.coef_
```

[61]:
```
array([[ 9.17517965,  0.        ,  0.        ,  5.01160431,
         0.        , -3.47379064,  0.        ,  0.65310136,
         0.        , 26.97955638,  0.        ,  0.        ,
         4.69404394,  0.21378175,  0.        ,  0.756085  ,
```

```
        0.        ],
      [ 0.        ,   4.27153943,  24.0845023 ,   0.        ,
        0.        ,   0.        ,   6.76348946,   0.        ,
        1.39732665,   0.        ,  15.06202754,   4.41802313,
        0.        ,  -2.19329857,   0.        ,  -1.04899691,
        0.        ],
      [-118.87271714, -29.35983135,   0.        ,   0.        ,
        0.        ,   0.        ,   0.        ,  -1.41583778,
      -44.43511175,   0.        ,   0.        ,   0.        ,
        0.        ,   0.        ,   0.        ,   0.        ,
        0.        ]])
```

[63]:

```python
# Extract and sort feature coefficients
def get_feature_coefs(regression_model, label_index, columns):
    coef_dict = {}
    for coef, feat in zip(regression_model.coef_[label_index, :], columns):
        if abs(coef) >= 0.01:
            coef_dict[feat] = coef
    # Sort coefficients
    coef_dict = {k: v for k, v in sorted(coef_dict.items(), key=lambda item:␣
↪item[1])}
    return coef_dict

# Generate bar colors based on if value is negative or positive
def get_bar_colors(values):
    color_vals = []
    for val in values:
        if val <= 0:
            color_vals.append('r')
        else:
            color_vals.append('g')
    return color_vals

# Visualize coefficients
def visualize_coefs(coef_dict):
    features = list(coef_dict.keys())
    values = list(coef_dict.values())
    y_pos = np.arange(len(features))
    color_vals = get_bar_colors(values)
    plt.rcdefaults()
    fig, ax = plt.subplots()
    ax.barh(y_pos, values, align='center', color=color_vals)
    ax.set_yticks(y_pos)
    ax.set_yticklabels(features)
    # labels read top-to-bottom
    ax.invert_yaxis()
    ax.set_xlabel('Feature Coefficients')
```

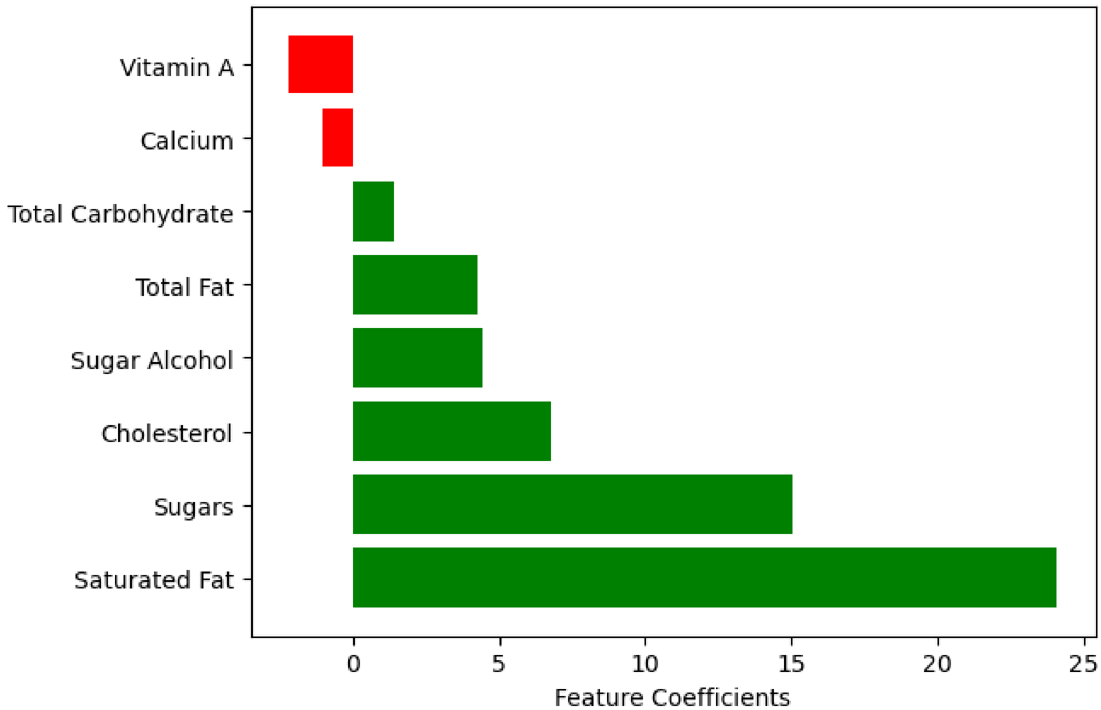
```
    ax.set_title('')
    plt.show()
```

[68]:
```
feature_cols = list(food_df.iloc[:, :-1].columns)
feature_cols
```

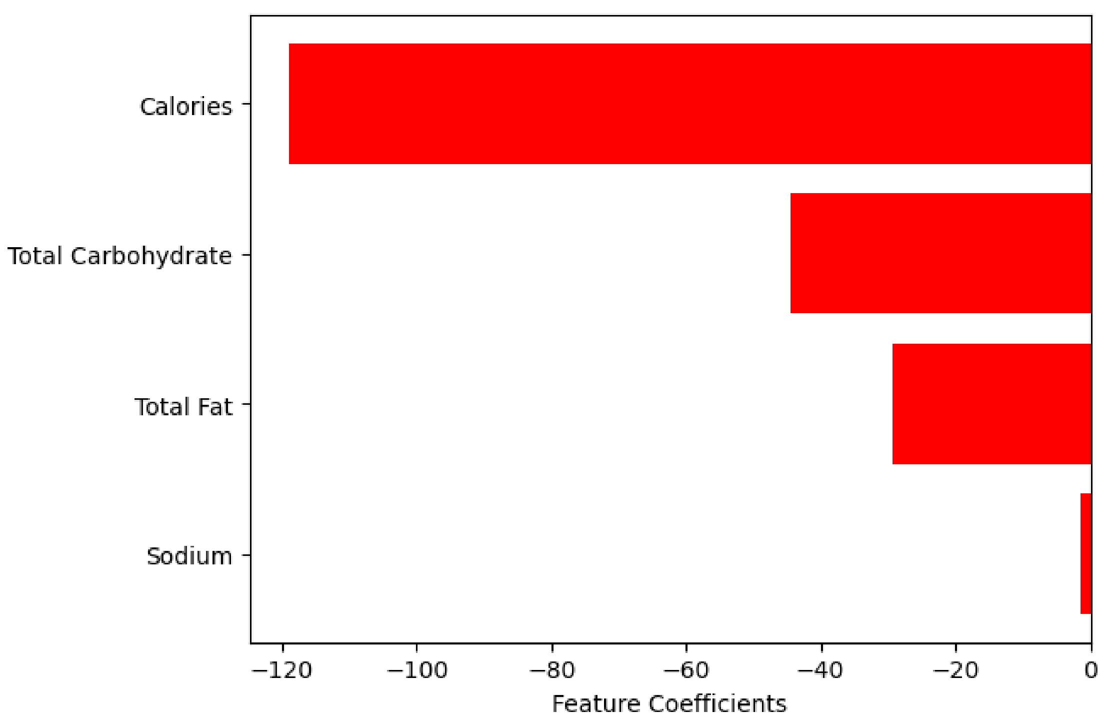
[68]:
```
['Calories',
 'Total Fat',
 'Saturated Fat',
 'Monounsaturated Fat',
 'Polyunsaturated Fat',
 'Trans Fat',
 'Cholesterol',
 'Sodium',
 'Total Carbohydrate',
 'Dietary Fiber',
 'Sugars',
 'Sugar Alcohol',
 'Protein',
 'Vitamin A',
 'Vitamin C',
 'Calcium',
 'Iron']
```

[71]:
```
# Get the coefficents for Class 1, Less Often
coef_dict = get_feature_coefs(l2_model, 1, feature_cols)
```

[72]:
```
visualize_coefs(coef_dict)
```



[70]:
```
# Coefficients for Class 2
coef_dict = get_feature_coefs(l2_model, 2, feature_cols)
visualize_coefs(coef_dict)
```



1.11 Conclusion

We find that, unhealthy nutrients such as saturated fat, sugars, cholesterol, total fat, etc. have high positive coefficients. Foods containing unhealthy nutrients will have higher coefficients and will be more likely to be classified as “Less often”.

For better explainability, we will try other models such as decision trees and K-nearest neighbors because overall the diabetes decision seems to be based on variables that have some relationship between them. The main weakness of this analysis is the fact that we have unbalanced classes and it will be necessary to think about the resampling in order to better apply the logistic regression

[ ]:

```

```