SHIMADZU
Excellence in Science

Application News
No. M282

Gas Chromatography Mass Spectrometry

Discovery of Markers from Chromatogram Data by Machine Learning

Techniques for identifying or classifying samples from chromatogram data by using machine learning algorithms have attracted considerable attention. In machine learning from chromatogram data, it is necessary to create a data table from peak intensity information, but the accuracy of the discriminant model may be reduced if a correlation exists between the sizes of the peaks or the sizes of the peaks are extremely different. For this reason, pretreatment of the data after creation of the data table of peak information is generally necessary in machine learning of chromatogram data.

This article introduces an example of the workflow in discovery of discriminant markers from GC-MS scan data of food samples by using Python 3.7.

T. Sakai

Data Acquisition

The data used here was the dataset for beef introduced in Application News No. M276. As shown in Fig. 1, two types of samples were prepared using red meat from various cuts of beef: properly refrigerated samples (4 °C samples) and samples which were expected to display deterioration due to exposure to a 40 °C environment for 3 h (40 °C samples). 20±3 mg of each sample was taken and placed in individual measurement vials. A total of 116 samples was prepared, comprising 58 of the 4 °C samples and 58 of the 40 °C samples, and the composition of the gas generated when the samples were heated to 200 °C was analyzed by the Solid Phase Micro Extraction (SPME) method. An AOC-6000 auto injector, which enabled automatic SPME injection, was used in this analysis (Fig. 2).

Fig. 3 shows an example of a chromatogram obtained as a result. Classification of the sample types was not possible from the sample appearance or the appearance of the chromatogram analysis results.

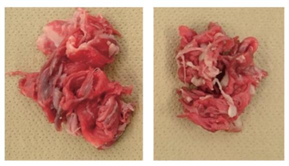

Fig. 1 Left: Properly Refrigerated Sample (4 °C Sample), Right: Sample Exposed to 40 °C Environment for 3 H (40 °C Sample)

Fig. 2 Appearance of GCMS-QP™2020 + AOC-6000

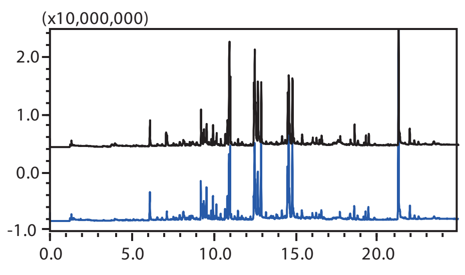

Fig. 3 Example of Total Ion Chromatograms Black: 4 °C Sample, Blue: 40 °C Sample

Picking/alignment of the deconvolution/peaks of the chromatogram data were done using the mass spectrometry data analysis software MZmine 2 (Ver. 2.32). The peak heights were output as data, as this data is relatively unaffected by waveform processing.

Because the dataset was extremely wide, comprising 9,318 peaks × 116 datafiles, we basically proceeded in the direction of reducing unnecessary peak data (features) for the data pretreatment.

Data Pretreatment and Model Creation

1. Treatment of missing values

Although imputation of missing values in chromatogram data is possible by adjusting the waveform processing parameters to some extent, it is almost always impossible to eliminate all missing values. In such cases, artificial substitution by a simple calculation, separate regression analysis, or the like is not considered advisable. Therefore, in this study, all features, including missing values, that occurred even once were deleted by using a commercial spreadsheet program. As a result, it was possible to narrow the features of the dataset to 200 peaks × 116 datafiles.

2. Data reading by Python and division into training and test sets

After deletion of the missing values, the feature names, data names, and similar information were input in the same general spreadsheet program. The dataset was saved in a CSV file, and was then imported to the Python console.

```
In [1]:
import pandas as pd
data =pd.read_csv("Data.csv", header=0, index_col=0)
data.head(5)
```

```
Out [1]:
          freshness   RT1.32_001   ...   RT22.52_199   RT23.48_200
Data001       0       154497      ...     10100          9510
Data002       0       154356      ...     12702         10054
Data003       0       179444      ...     15774         11863
Data004       0       211854      ...     11123          9546
Data005       0       129346      ...     12236         10996

[5 rows x 200 columns]
```

Here, the data names in each line ("Data001", "Data002" . . .), the label data in the "freshness" column ("1" or "0", indicating 4 °C Sample or 40 °C Sample, respectively), and the other feature data in each column (where the peaks are denoted by RT followed by a serial number) were arranged. The values of the features are the height of the respective peaks.

As described in Application News No. M276, this dataset was split into a training set of 92 datafiles and a test set of 24 datafiles. The model was created using the training set, and predictions were made using the test set. Division was done using Stratified Shuffle Split so as to avoid skewing of the label data.

```
In [2]:
def X_y_split(data):
    y = data.iloc[:, 0]
    X = data.iloc[:, 1:]
    return X, y

In[3]:
def stratified_shuffle_split(X, y, test_size=0.2):
    from sklearn.model_selection import StratifiedShuffleSplit
    sss = StratifiedShuffleSplit(test_size=test_size, random_state=0)
    for train_index, test_index in sss.split(X, y):
        train_X, test_X = X.iloc[train_index], X.iloc[test_index]
        train_y, test_y = y.iloc[train_index], y.iloc[test_index]
    return train_X, test_X, train_y, test_y

In[4]:
X, y = X_y_split(data)
train_X, test_X, train_y, test_y = stratified_shuffle_split(X,y,0.2)
```

3. Correlation between features

If a strong correlation exists between two features, one of the features in which this correlation is recognized is deleted due to occurrence of the problem of multicollinearity. For this reason, the correlation coefficient between each pair of features is checked.

```
In [5]:
import seaborn as sns
sns.heatmap(train_X.corr(), cmap="bwr", ¥
xticklabels=False, yticklabels=False)
```

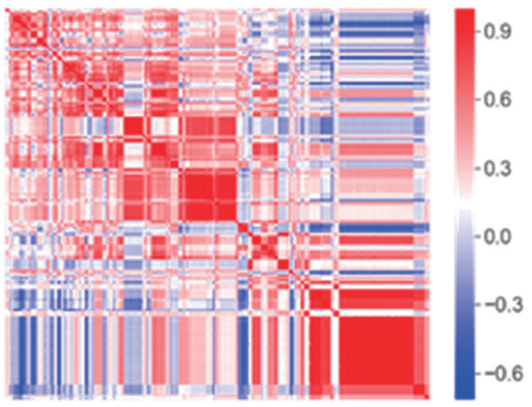

Fig. 4 Heatmap of Correlation Coefficient Matrix for Pairs of Features

Many parts are shown in blue and red, indicating correlation of multiple features.

```
In [6]:
def drop_correlating_columns (train_X, test_X, corr_value=0.8):
    import numpy as np
    corr1 = train_X.corr().abs()
    corr2 = np.triu(corr1.values,1)
    corr3 = np.asarray(np.where(corr2>corr_value))
    corr4 = np.unique(corr3[1])
    del_list = train_X.columns[corr4]
    train_X_ncorr = train_X.drop(del_list, axis=1)
    test_X_ncorr = test_X.drop(del_list, axis=1)
    return train_X_ncorr, test_X_ncorr

In [7]:
train_X_ncorr, test_X_ncorr = drop_correlating_columns¥
(train_X, test_X, corr_value=0.8)

In [8]:
train_X_ncorr.head(5)
```

```
Out[8]
        RT1.32_001   RT3.98_003   ...   RT19.56_155   RT22.52_199
Data050      135257       36010   ...         14516         13188
Data006      129535       34817   ...          7891         10228
Data032      219201       13716   ...         14183         12402
Data047      121548       42813   ...          9761         11610
Data055       90842       31363   ...         13666         13254

[5 rows x 30 columns]
```

It can be understood that peaks with similar retention times include many combinations with high correlation coefficients. Here, when a combination of features had a correlation coefficient of 0.8 or higher, one of the features was deleted. This made it possible to narrow the number of features to 30.

4. Narrowing of features

After the number of features decreases, the content of each feature is checked.

4-1. Distribution of values

The histograms of the values are checked, and select the features that include less outliers and the distribution is simple.

```
In [9]:
def hist_features(data, num):
    import matplotlib
    import matplotlib.pyplot as plt
    plt.figure(figsize=(8,16))
    plt.subplots_adjust(wspace=0.2, hspace=1)
    matplotlib.rc('xtick', labelsize=8)
    matplotlib.rc('ytick', labelsize=8)
    for i, col in enumerate(list(data.columns)[num:num + 60]):
        plt.subplot(10, 3, i + 1)
        plt.hist(data[col])
        plt.title(col, fontsize=12)
    return

In [10]:
hist_features (train_X_ncorr, 0)
```

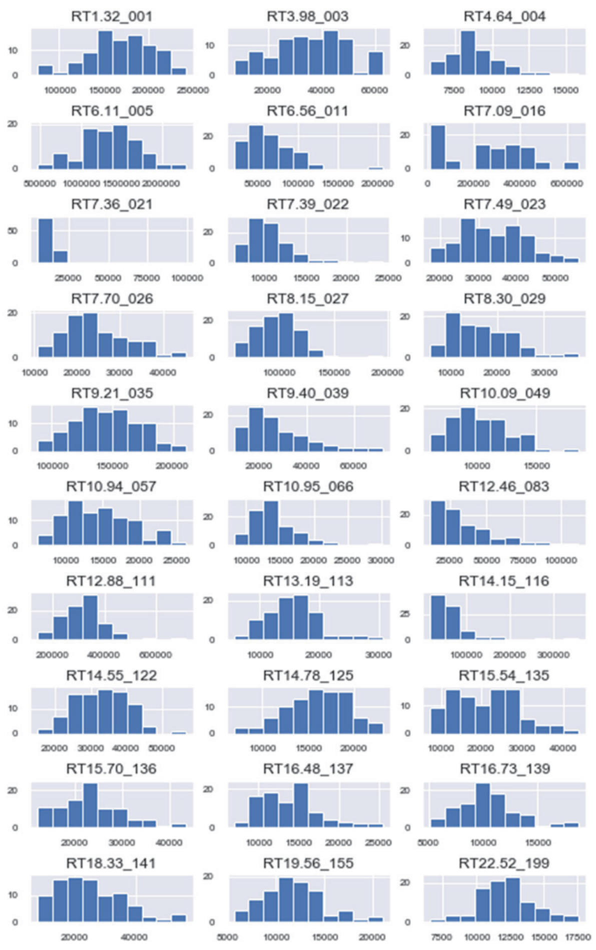

Fig. 5 Histograms of Features

Features that are outliers or have skewed distributions are excluded from candidates because functioning as a marker is generally difficult.

4-2. Distribution of features by label

Features whose distribution differs by label have a high possibility as marker candidates. The candidates are normalized with z-score and validated by a boxplot or other appropriate method.

```
In [11]:
def boxplot (train_X, train_y):
   import matplotlib.pyplot as plt
   import pandas as pd
   import seaborn as sns
   plt.figure(figsize=(15,10))
   for_sns = pd.concat([train_y, train_X], axis=1)
   for_sns = pd.melt(for_sns, id_vars="freshness", var_name="features",
            value_name="value")
   sns.boxplot(x="features", y="value", hue="freshness", data=for_sns)
   plt.xticks(rotation=90)
   plt.tight_layout()
   return

In [12]:
def standard_scaler_for_train_test (train_X, test_X):
   from sklearn.preprocessing import StandardScaler
   import pandas as pd
   ss = StandardScaler().fit(train_X)
   train_X_scaled = pd.DataFrame(ss.transform¥
   (X=train_X), columns=train_X.columns, index=train_X.index)
   test_X_scaled = pd.DataFrame(ss.transform¥
   (X=test_X), columns=test_X.columns, index=test_X.index)
   return train_X_scaled, test_X_scaled

In:[13]
train_X_ncorr_scaled,test_X_ncorr_scaled=¥
standard_scaler_for_train_test (train_X_ncorr, test_X_ncorr)
boxplot(train_X_ncorr_scaled, train_y)
```

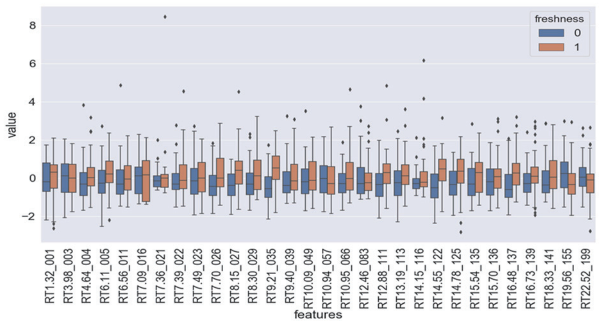

Fig. 6 Boxplot of Peaks by Label

4-3. Contribution to model

A temporal model is created here using the current 30 features, and the contributions of each feature are calculated. Since this study concerns the discovery of marker compounds with a high contribution to target classification by a binary discriminant model for the above-mentioned 4 °C and 40 °C samples, a logistic regression type model was used, and the hyperparameters “C” and “tol” were optimized by grid search.

```
In [14]:
def predict_with_logreg (train_X, test_X, train_y, test_y):
   from sklearn.linear_model import LogisticRegression as logreg
   from sklearn.model_selection import GridSearchCV
   from sklearn.metrics import accuracy_score as ac
   import pandas as pd
   import matplotlib.pyplot as plt
   from sklearn.metrics import confusion_matrix
   import seaborn as sns
   parameters_logreg = {"C": [100, 10, 1, 0.1, 0.01],
            "tol": [1e-3, 1e-4, 1e-5, 1e-6, 1e-7]}
   gscv = GridSearchCV(logreg(), parameters_logreg, cv=5)
   gscv.fit(train_X.values, train_y.values)
   best_params = gscv.best_params_
   model_01 = logreg(C=best_params["C"],
            tol=best_params["tol"], solver="lbfgs", random_state=4)
   model_01.fit (train_X.values, train_y.values)
   predict_y = pd.Series(model_01.predict(test_X.values),
            index=test_y.index)
   ac_score = ac(test_y, predict_y)
   cm = confusion_matrix(test_y, predict_y)
   plt.figure(figsize=(12,10))
   ax = plt.subplot()
sns.heatmap(cm, annot=True,cmap="Blues",fmt="d",cbar=False)
   sns.set(font_scale=2)
   ax.set_xlabel("Predicted Label", fontsize=20)
   ax.set_ylabel("True Label", fontsize=20)
   print("accuracy score is {0}".format(ac_score))
   return predict_y, model_01

In [15]:
   predict_y_01, model_01 = predict_with_logreg¥
   (train_X_ncorr_scaled, test_X_ncorr_scaled, train_y, test_y)
```

```
accuracy score is 0.75
```

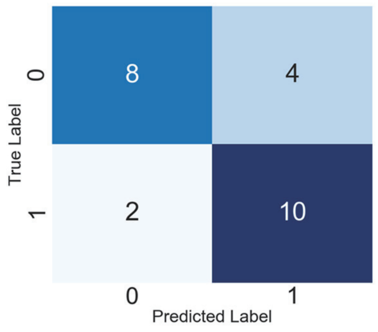

Fig. 7 Confusion Matrix of Predicted Label and True Label

Although the classification accuracy is 75% at this point, in further work, we improved the accuracy of the model by selecting features with higher contributions.

Although the coefficients in logistic regression can be considered as contributions of each features, here permutation importance algorithm is adopted in terms of its higher generality in various type of models.

```
In [16]:
def permutation_importances (train_X, train_y, iter=10):
   from sklearn.linear_model import LogisticRegression as logreg
   from sklearn.model_selection import GridSearchCV
   from sklearn.model_selection import train_test_split
   from eli5.sklearn import PermutationImportance
   import pandas as pd
   import matplotlib.pyplot as plt
   perm_imps = pd.DataFrame(columns = train_X.columns)
   for i in range(iter):
      tr_X, val_X, tr_y, val_y = train_test_split ¥
      (train_X, train_y, test_size=0.2, random_state=i)
      parameters_logreg = ¥
      {"C": [100, 10, 1, 0.1, 0.01], "tol": [1e-3, 1e-4, 1e-5, 1e-6, 1e-7]}
      gscv = GridSearchCV(logreg(), parameters_logreg, cv=5)
      gscv.fit(tr_X, tr_y)
      best_params = gscv.best_params_
      model_01 = logreg(C=best_params["C"], ¥
      tol=best_params["tol"], solver="lbfgs", random_state=0)
      model_01.fit (tr_X, tr_y)
      perm = PermutationImportance(model_01, ¥
      random_state=0).fit(val_X, val_y)
      perm_imps = perm_imps.append ¥
      (pd.Series(perm.feature_importances_, ¥
      index=train_X.columns), ignore_index=True)
   plt.figure(figsize=(12,8)).subplots_adjust(bottom=0.3)
   plt.bar(x=train_X.columns, height=perm_imps.mean(), ¥
   yerr=perm_imps.std())
   plt.xticks(rotation=90)
   plt.hlines(y=0, xmin=-0.5, xmax=train_X.shape[1]-0.5)
   return perm_imps

In [17]:
perm_imps = permutation_importances ¥
(train_X_ncorr_scaled, train_y)
```

Since the number of samples is relatively small, a single try of calculation may not reach true value. Therefore, several split patterns of training / test set were tried, and their average value was taken. Although the standard deviation was rather large, as expected, it was possible to grasp the overall trend.

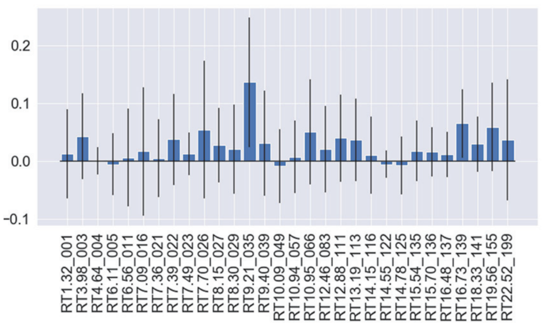

Fig. 8 Values of Permutation Importance of Each Feature

Based on the study and trial-and-error process described above, 10 features were selected at this time.

```
In [17]:
feature_list_01 = pd.Series(["RT1.32_001", "RT3.98_003",¥
"RT7.39_022", "RT9.21_035", "RT9.40_039", "RT12.46_083",¥
"RT12.88_111", "RT14.15_116", "RT16.73_139", "RT19.56_155"])

In [18]:
train_X_dropped = train_X_ncorr_scaled[feature_list_01]
test_X_dropped = test_X_ncorr_scaled[feature_list_01]
```

Because GC-MS scan data are used in this study, a qualitative analysis of the candidate marker compounds is possible by using library search and reference standards at this point in time (Table 1).

Table 1 Peaks Selected in Study and Their Library Search Results

Peak	#1 hit compound in library search results	Variable
RT1.32_001	Trimethylamine	*x1*
RT3.98_003	2-hydroxypropanamide	*x2*
RT7.39_022	gamma.-n-Amylbutyrolactone	*x3*
RT9.21_035	Uric acid	*x4*
RT9.40_039	Lauryl acetate	*x5*
RT12.46_083	9-Octadecen-1-ol	*x6*
RT12.88_111	Hexadecanamide	*x7*
RT14.15_116	Oleic Acid	*x8*
RT16.73_139	Benzyl icosanoate	*x9*
RT19.56_155	5-Cholestene	*x10*

5. Feature engineering

In the logistic regression method, the output probability is expressed by a first-order standard sigmoid function of the features. A higher-order model of the features may be more effective in some cases, depending on the dataset. However, there is currently a tendency to avoid extreme high-order models, which generally result in an increased calculation load and overfitting.

Here, the result of a single division operation for each case was used as a new feature "ratio of pairs of compound peaks," considering the fact that the data were chromatograms.

```
In [19]:
def features_ratio (train_X, test_X):
    import pandas as pd
    from sklearn.preprocessing import StandardScaler
    train_X_r = train_X.copy()
    for col1 in train_X.columns:
        for col2 in train_X.columns:
            train_X_r[col1+"_"+col2] = train_X[col1] / train_X[col2]
    test_X_r = test_X.copy()
    for col1 in test_X.columns:
        for col2 in test_X.columns:
            test_X_r[col1+"_"+col2] = test_X[col1] / test_X[col2]

    ss = StandardScaler().fit(train_X_r)

    train_X_r = pd.DataFrame(ss.transform(train_X_r),¥
columns=train_X_r.columns, index=train_X_r.index)
    test_X_r = pd.DataFrame (ss.transform(test_X_r), ¥
columns=test_X_r.columns, index=test_X_r.index
    return train_X_r, test_X_r

In [20]:
train_X_r, test_X_r = features_ratio (train_X_dropped, ¥
test_X_dropped)
```

Because the number of features became excessive, features values were selected arbitrarily based on the distribution and contribution of the features, in the same manner as above.

The final features were as follows:

```
In [21]:
feature_list_final = pd.Series(["RT9.21_035",¥
 "RT9.21_035_RT7.39_022", "RT3.98_003_RT9.21_035",¥
 "RT3.98_003_RT16.73_139", "RT9.40_039_RT19.56_155",¥
 "RT1.32_001_RT9.21_035", "RT16.73_139",¥
 "RT14.15_116_RT1.32_001", "RT9.21_035_RT19.56_155",¥
 "RT7.39_022_RT9.40_039"])
In [22]:
train_X_final = train_X_r[feature_list_final]
test_X_final = test_X_r[feature_list_final]
In [23]:
predict_y_final, model_final = predict_with_logreg¥
(train_X_final, test_X_final, train_y, test_y)
```

```
accuracy score is 0.9166666666666666
```

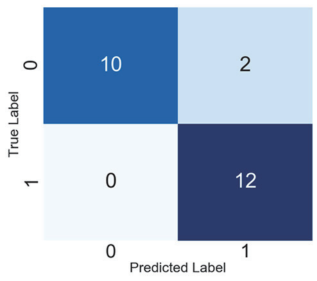

Fig. 9 Confusion Matrix of Predicted Label and True Label

Finally, it was possible to classify unknown samples with precision of 91.67% by using this model. The equation for predicting the probability *P* of label 1 is as follows.

$$P = (1 + \exp(1.19 * x4 + 1.22 * \frac{x4}{x3} - 0.21 * \frac{x2}{x4} - 1.32 * \frac{x2}{x9}$$
$$+ 1.72 * \frac{x5}{x10} - 0.50 * \frac{x1}{x4} + 0.18 * x9 + 1.25 * \frac{x8}{x1}$$
$$+ 0.16 * \frac{x4}{x10} - 0.07 * \frac{x3}{x5}))^{-1}$$

The classification process is based on $P \geq 0.5 \Rightarrow 1$, $P < 0.5 \Rightarrow 0$.

Table 2 Version Information

Python	3.7.3	seaborn	0.9.0
numpy	1.16.2	sklearn	0.20.3
pandas	0.24.2	eli5	0.8.2
matplotlib	3.0.3		

GCMS-QP is a trademark of Shimadzu Corporation in Japan and/or other countries.

First Edition: Oct. 2019

SHIMADZU

Shimadzu Corporation
www.shimadzu.com/an/

Related Products Some products may be updated to newer models.

GCMS-QP2020 NX

Gas Chromatograph Mass Spectrometer

Related Solutions

Metabolomics

Food Metabolomics

Life Science

Price Inquiry

Product Inquiry

Technical Service / Support Inquiry

Other Inquiry