SHIMADZU
Excellence in Science

Application News

GC/MS

No. M282

機械学習によるクロマトグラムデータからのマーカー探索

クロマトグラムデータから機械学習のアルゴリズムを用いてサンプルの判別を行う手法が注目されています。クロマトグラムデータから機械学習を行うには、ピークの強度情報をデータ表にする必要がありますが、ピーク同士の大きさに相関があったり、ピーク同士の大きさが極端に異なったりすると、判別モデルの精度が低下する場合があります。そのため、一般的にクロマトグラムデータを機械学習させる際には、ピーク情報をデータ表にした後、データの前処理が必要になります。

本稿では、食品サンプルのGC-MSスキャンデータから、Python 3.7を使って判別マーカーを探索する流れの一例をご紹介します。

T. Sakai

■データの取得

データは、アプリケーションニュースM273でもご紹介した牛肉のデータセットを用いました。牛肉のさまざまな部位の赤身に関して、適切に冷蔵保存されているもの（4℃サンプル）と、40℃環境に3時間暴露され劣化が進んだと考えられるもの（40℃サンプル）を準備しました（図1）。これらのサンプルに関して、20±3 mgを測定用バイアルに取ったものを、116本（ 4℃サンプル58本、 40℃サンプルを58本）準備し、200℃で加熱したときに発生するガス成分を、Solid Phase Micro Extraction法（SPME）を用いて分析しました。なお、分析には自動でのSPMEインジェクションが可能なオートインジェクターAOC-6000を用いました（図2）。

その結果得られたクロマトグラムの例を図3に示しました。サンプルの外観および分析結果クロマトグラムの外観からは、判別不可能であることがわかりました。

図 1 左：適切に冷蔵保存されているもの(4℃サンプル)右：40℃環境に3時間暴露されたもの(40℃サンプル)

図 2 GCMS-QP™2020 + AOC-6000の外観

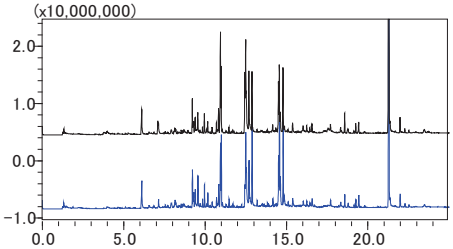

図 3 トータルイオンクロマトグラムの例
黒：4℃サンプル, 青：40℃サンプル

質量分析データの解析ソフトウェアであるMZmine 2 (Ver. 2.32)を使用して、クロマトグラムデータのデコンボリューション/ピークのピッキング/アラインメントを行いました。データは、波形処理の影響を受けづらいピーク高さを出力しました。

データセットは116データ x 9318ピークという非常に横長になったため、データの前処理は基本的に不要なピーク情報（特徴量）を削減していく方向で進めます。

■データの前処理とモデル作成

1. 欠損値の処理

クロマトグラムデータにおける欠損値は波形処理パラメータを調整することである程度救済することが可能ですが、完全に無くすことは難しい場合がほとんどです。その場合に単純な計算や別の回帰分析などで人工的に埋めることはあまり得策ではないと考えられるため、今回は一つでも欠損値を含む特徴量を、市販の表計算ソフトを用いてすべて削除しました。この結果、データセットは116データ x 200ピークまで特徴量を絞ることができました。

2. Pythonでのデータの読み込みとトレーニング/テストデータの分割

欠損値処理したデータセットを、同じく一般的な表計算ソフト内で特徴量名、データ名などを入力したのち、CSVファイルで保存し、Python コンソールに読み込みます。

```
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]
```

ここでは、各行にデータ名を(“Data001”, Data002”…)、”freshness”列にラベルデータ（ 4℃サンプルか40℃サンプルかを、それぞれ1と0で表現）を、その他の各列に特徴量データ（各ピークをRTと通し番号で名前付けしたもの）を配置しました。特徴量の値は各ピークの高さです。

アプリケーションニュースM273と同様に、このデータセットを、トレーニングセット92データ/テストセット24データに分割します。トレーニングセットでモデルを作成し、テストセットの予測を行います。分割はラベルデータの偏りが出ないように、Stratified Shuffle Splitを使用しました。

```
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. 特徴量同士の相関

特徴量同士に強い相関があると、多重共線性の問題が発生するため、相関が認められる特徴量同士は、一方を削除します。そのために、各特徴量同士の相関係数を確認します。

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

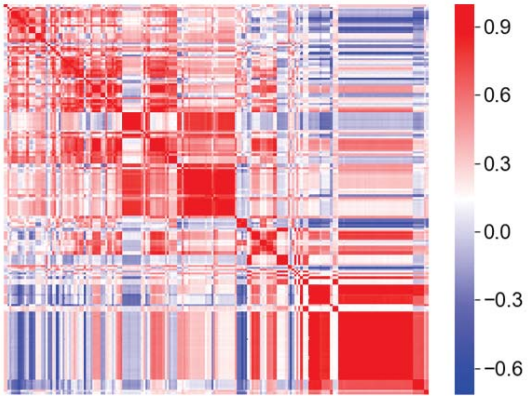

図 3 各特徴量同士の相関係数行列のヒートマップ
青や赤で示された部分が多く、複数の特徴量同士が相関していることがわかる。

```
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]
```

保持時間が近いピーク同士に、相関係数の高い組み合わせが多いことがわかります。ここでは、相関係数が0.8以上となる特徴量の組み合わせのうち、一方を削除しました。その結果、特徴量を30個まで絞り込むことができました。

4. 特徴量の絞り込み

特徴量の数が減ってきたら、各特徴量の内容について確認します。

4-1 . 値の分布

値のヒストグラムを確認し、なるべく外れ値がなく、分布が平均値を中心に左右に広がっている特徴量を選びます。

```
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)
```

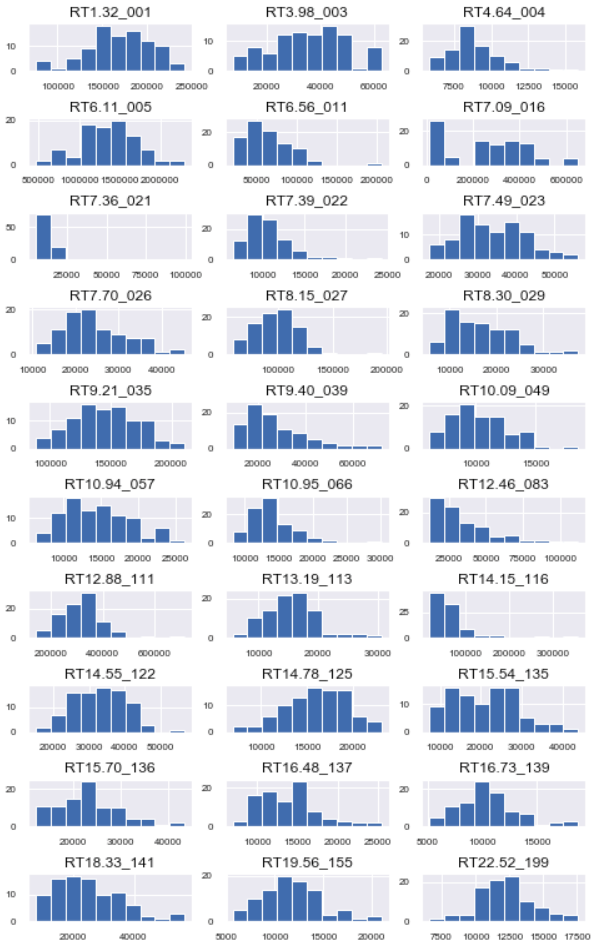

図 4 各特徴量の値のヒストグラム
外れ値や値の分布に偏りがある特徴量はマーカーとして機能しにくいことが多いので候補から外す。

4-2 . ラベルごとの特徴量値の分布

ラベルごとに特徴量値の分布に違いがあるものはマーカー候補の可能性が大きくなります。特徴量をスケーリングし、ボックスプロットなどで確認します。

```
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)
```

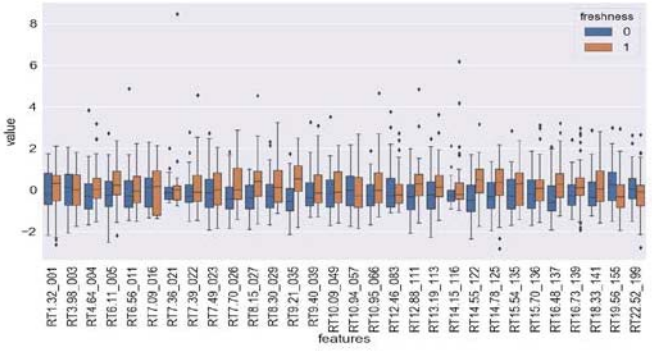

図5 ラベルごとの各ピークのボックスプロット

4-3 . モデルにおける寄与

現在の30個の特徴量でいったんモデルを作成し、各特徴量の寄与を計算します。今回は、4℃サンプルと40℃サンプルの二値判別モデルで、目的は判別に対して寄与の高いマーカー化合物を探索することなので、モデル式としてはロジスティック回帰を使います。ハイパーパラメータは"C"と"tol"をグリッドサーチにて最適化しました。

```
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
```

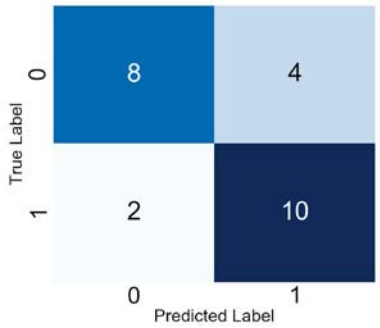

図6 予測されたラベルと実際のラベルのConfusion Matrix

現状でも精度75%で判別できていますが、ここからさらに寄与の高い特徴量を選択し、モデルの精度を上げます。

特徴量の寄与に関しては、ロジスティック回帰であれば各特徴量の係数の絶対値を寄与の大きさと考えることもできますが、ここではより汎用性の高いPermutation Importanceを使いました。

```
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)
```

サンプル数が多くないので、Permutation Importanceはtrainデータ内の分割の試行によってかなりばらつきが発生することが考えられます。ここでは分割のパターンをいくつか試して、その平均を取りました。予想通り標準偏差がかなり大きくなりましたが、全体的な傾向は捉えることができました。

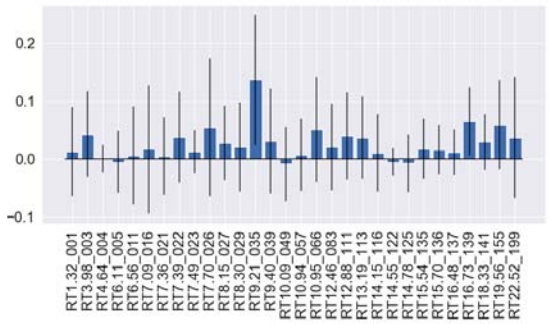

図 7　各特徴量のPermutation Importance の値

以上のような検討および試行錯誤から、今回は10個の特徴量を選択しました。

```
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]
```

今回のデータはGC-MSのスキャンデータなので、この時点でライブラリ検索や標準品を用いて、マーカー候補化合物の定性分析を行うこともできます（表1）。

表1 今回選択したピークと、そのライブラリ検索結果

ピーク名	ライブラリ検索結果における #1ヒット化合物	変数名
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. 特徴量エンジニアリング

ロジスティック回帰では、出力確率は特徴量の一次式の標準シグモイド関数で表されますが、データセットによっては特徴量の高次式モデルが有効になる場合もあります。ただし、一般的にはモデル式の高次化は、計算量の増大や過学習の要因となることが多いので、現状では極端な高次化は避けられる傾向にあります。

ここではデータがクロマトグラムであることを考慮し、「化合物ピーク同士の比」として、各々一回除算を行ったものを新しい特徴量としました。

```
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)
```

特徴量が多くなったので、さきほどと同様に特徴量値の分布や寄与から、特徴量を任意に選択します。

最終的な特徴量は以下の通りとしました。

```
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
```

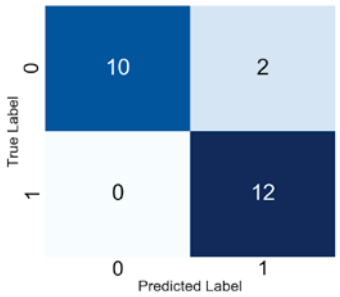

図 9　予測されたラベルと実際のラベルのConfusion Matrix

最終的に、このモデルで未知のサンプルを91.67%の精度で判別することができました。ラベルが1である確率Pを予測する式は以下の通りです。

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

判別は、 $P \geqq 0.5 \Rightarrow 1 \quad P < 0.5 \Rightarrow 0$ で行っています。

表2 バージョン情報一覧

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		

株式会社 島津製作所 分析計測事業部 グローバルアプリケーション開発センター

初版発行：2019 年 9月

島津コールセンター 0120-131691 (075) 813-1691