Case_study3: mouse skin lipids dataset

Yinyue Zhu

Table of contents

1 Overview

In this case study, we performed processing and annotation on a mouse skin lipid dataset. Annotation is based on m/z and CCS matching against CCS Compendium.

2 Downstream Analysis – feature annotation

2.0.1 Introduction

In this case study, we process a MALDI-TIMS-MS1 lipids dataset of mouse skin tissue, extract features with m/z and ion mobility, then convert ion mobility into collision cross section(CCS), and annotate features by looking up m/z and CCS in CCS compendium.

```
import timsimaging

# enable visualization in the Jupyter notebook
from bokeh.io import show, output_notebook
output_notebook()
# disable FutureWarning
import warnings
warnings.filterwarnings('ignore', category=FutureWarning)
```

```
Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_lo
```

```
bruker_d_folder_name = r"D:\dataset\Melanie\MF402upper.d"
dataset = timsimaging.spectrum.MSIDataset(bruker_d_folder_name)
dataset
```

```
100%|                                          | 65052/65052 [00:13<00:00, 4783.46it/s]

MSIDataset with 65052 pixels
        mz range: 299.999-1350.004
        mobility range: 0.700-1.960
```

```
dataset.image()
```

```
Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_ex
```

2.0.2 Peak processing

First we run the processing workflow to get the feature list. There are 3 columns in the result: m/z, ion mobility and intensity. The ion mobility is in $1/K_0$ (inverse reduced mobility), of which the unit is $V \cdot s \cdot cm^{-2}$

```
results = dataset.process(sampling_ratio=0.1, frequency_threshold=0.02, tolerance=3, adaptive
peak_list = results["peak_list"]
peak_list
```

```
Computing mean spectrum...
Traversing graph...
Finding local maxima...
Summarizing...

100%|                                                | 930/930 [02:41<00:00,  5.77it/s]
```

	mz_values	mobility_values	total_intensity
1	302.058416	0.906070	28.807071
2	303.065771	0.907390	273.366334
3	303.065934	1.076893	73.819985
4	303.065516	0.948773	8.603689
5	304.069069	0.906924	3.887010
...	...	...	...
926	1253.899905	1.844601	14.027978
927	1255.913279	1.853024	26.725288
928	1281.929607	1.870141	25.601230
929	1283.943538	1.877011	2.597694
930	1298.554342	1.759565	7.082859

2.0.3 Converting ion mobility into CCS

However, $1/K_0$ depends on the ion mobility technique and instrument. The data was collected by trapped ion mobility spectrometry(TIMS) while most ion mobility databases are from drift tube(DTIMS), to allow cross-platform comparison, we need to convert ion mobility into CCS, a property only depends on the ion itself.

TIMSImaging converts $1/K_0$ into CCS by a linear model fitted with reference ion mobilities of calibrants. The metadata of calibration is stored in the raw data:

```
dataset.cali_info
```

KeyName	KeyPolarity	Value
CalibrationDateTime	+	2024-07-28T12:37:35+02:00
CalibrationUser	+	unknown
CalibrationSoftware	+	timsTOF
CalibrationSoftwareVersion	+	5.1.8
MzCalibrationMode	+	4
MzStandardDeviationPPM	+	0.209581
ReferenceMassList	+	RedP__0__to__2000 07-03-2024 New List
MzCalibrationSpectrumDescription	+	<unknown>
ReferenceMassPeakNames	+	b'p3\x00p5\x00p9\x00p17\x00p21\x00p25\x00p29\x...
ReferencePeakMasses	+	b'\x17\xd9\xce\xf7S”\x8a@\xc5 \xb0r\xe8\x11\x8...
MeasuredTimesOfFlight	+	b'!\xe8\xa9\xb9\x14\x0c\xf2@e\xaeo\xd5\x7f\xb3...
MeasuredMassPeakIntensities	+	b'\x00\x00\x00\x00\x80<\xc6@\x00\x00\x00\x00\x..
MassesPreviousCalibration	+	b'\xe0\xf4\xb4cY”\x8a@\xd79uE\xec\x11\x8c@\x7f...
MassesCorrectedCalibration	+	b'\x962\xe9BT”\x8a@B\x8e\xd8\xe1\xe7\x11\x8c@\.
MobilityCalibrationDateTime	+	2024-07-27T13:42:36+02:00
MobilityCalibrationUser	+	unknown
MobilityStandardDeviationPercent	+	0.175188
ReferenceMobilityList	+	Tuning Mix ES-TOF CCS compendium (ESI)
CalibrationMobilogramDescription	+	<unknown>
ReferenceMobilityPeakNames	+	b'C6H19N3O6P3\x00C12H19F12N3O6P3\x00C18H19F
ReferencePeakMobilities	+	b'\xbe\xa3\x80$\x12\x90\xe7?\x00\xa0\xefq\x07\...
MeasuredTimsVoltages	+	b':\xe9\xb9o{uY@\x9d\xa9\x10\x8a\x9ada@\x8a:0,...
MeasuredMobilityPeakIntensities	+	b'\x0eJ\x96\x80\x13\xe1CA\xdce\xc5&\xb0\x9feA\..
MobilitiesPreviousCalibration	+	b'\x8a^\x97]\xd1\x9e\xe7?\xfd\xdap2s\xd5\xef?\...
MobilitiesCorrectedCalibration	+	b'\xb5\x1aw\xe9\xb5\x8c\xe7?”d\xaaG\x9a\xc6\xe...
MobilitiyReferencePressure	+	2.195872
MobilitiyPressureCompensationFactor	+	50.000000

Compute CCS. We also have an option to call interal CCS conversion function from Bruker's TDFSDK API. Here we compute CCS in both methods.

```
ccs_linear_model = dataset.ccs_calibrator(method="linear")
peak_list["CCS_linear"] = ccs_linear_model.transform(peak_list["mz_values"], peak_list["mobi
ccs_bruker_model = dataset.ccs_calibrator(method="internal")
peak_list["CCS_bruker"] = ccs_bruker_model.transform(peak_list["mz_values"], peak_list["mobi
peak_list["Delta"] = (peak_list["CCS_bruker"]-peak_list["CCS_linear"])/peak_list["CCS_bruker"
```

```
peak_list
```

	mz_values	mobility_values	total_intensity	CCS_linear	CCS_bruker	Delta
1	302.058416	0.906070	28.807071	189.238376	189.666498	0.002257
2	303.065771	0.907390	273.366334	189.489590	189.916152	0.002246
3	303.065934	1.076893	73.819985	225.188273	225.393028	0.000908
4	303.065516	0.948773	8.603689	198.205047	198.577456	0.001875
5	304.069069	0.906924	3.887010	189.364654	189.791992	0.002252
...	...	...	...	...	...	...
926	1253.899905	1.844601	14.027978	374.204010	373.482918	-0.001931
927	1255.913279	1.853024	26.725288	375.913545	375.181832	-0.001950
928	1281.929607	1.870141	25.601230	379.316539	378.563683	-0.001989
929	1283.943538	1.877011	2.597694	380.709363	379.947853	-0.002004
930	1298.554342	1.759565	7.082859	356.744077	356.131468	-0.001720

The results from TIMSImaging are almost the same with those from Bruker's method.

2.0.4 Query features against CCS database

Now we can search m/z and CCS in a database to obtain putative feature annotation. Here we use CCS compendium, which could be downloaded from https://mcleanresearch-group.shinyapps.io/CCS-Compendium

```
import pandas as pd
def query_feature(mz, ccs, i, library, ppm_tol=20, ccs_tol=5):

    columns = ['Compound', 'Neutral.Formula', 'CAS',
        'Theoretical.mz', 'Ion.Species', 'Charge',
        'CCS', 'Super.Class',
        'Class', 'Subclass']
    # ppm window
    mz_tol = mz * ppm_tol * 1e-6
    mz_min, mz_max = mz - mz_tol, mz + mz_tol

    ccs_min, ccs_max = ccs - ccs_tol, ccs + ccs_tol
    hit_index =  (library['mz'].between(mz_min, mz_max)) & (library['CCS'].between(ccs_min, c
    # candidate subset by adduct and mz
    hits = library.loc[hit_index, columns].copy()
    hits["feature_id"] = i
```

```
    hits["measured_mz"] = mz
    hits["measured_CCS"] = ccs
    hits["ppm_error"] = (hits["Theoretical.mz"] - mz) / mz * 1e6
    hits["ccs_dev_%"] = ((hits["CCS"] - ccs) / ccs).abs() * 100

    if hits.empty:
        return pd.DataFrame()
    return hits
```

To save time, we search against a subset of the database: lipids with +1 charge state:

```
library = pd.read_csv(r"D:\dataset\UnifiedCCSCompendium_FullDataSet_2025-07-28.csv")
lipids_lib = library.loc[
    (library["Super.Class"]=="Lipids and lipid-like molecules")&
    (library["Charge"]==1)
    ]
lipids_lib
```

	Compound	Neutral.Formula	CAS	InChi
1045	5-iPF2alpha-VI	C20H34O5	180469-63-0	InChI=1S/C20H34O5/c1-2-3-4-5
1047	8-iso-15(R)-Prostaglandin F2alpha	C20H34O5	214748-65-9	InChI=1S/C20H34O5/c1-2-3-6-9
1049	8-iso-Prostaglandin F2alpha	C20H34O5	27415-26-5	InChI=1S/C20H34O5/c1-2-3-6-9
1051	15(R)-Prostaglandin F2alpha	C20H34O5	37658-84-7	InChI=1S/C20H34O5/c1-2-3-6-9
1053	11-beta-Prostaglandin F2alpha	C20H34O5	38432-87-0	InChI=1S/C20H34O5/c1-2-3-6-9
...	...	...	...	...
3666	Progesterone	C21H30O2	57-83-0	InChI=1S/C21H30O2/c1-13(22)
3668	17-alpha-Hydroxyprogesterone	C21H30O3	68-96-2	InChI=1S/C21H30O3/c1-13(22)
3669	17-alpha-Hydroxyprogesterone	C21H30O3	68-96-2	InChI=1S/C21H30O3/c1-13(22)
3670	d7-Cholesterol Ester (18:1)	C45H71D7O2	1416275-35-8	InChI=1S/C45H78O2/c1-7-8-9-
3671	Cholesteryl Palmitate	C43H76O2	601-34-3	InChI=1S/C43H76O2/c1-7-8-9-

Query features with m/z tolerance=20 ppm and CCS tolerance=5 $\mathring{A}^2$:

```
all_results = []
for i, feat in peak_list.iterrows():
    all_results.append(query_feature(feat['mz_values'], feat['CCS_linear'], i, lipids_lib, pp
```

```
final_results = pd.concat([r for r in all_results if not r.empty],
                          ignore_index=True) if any(not r.empty for r in all_results) els
final_results
```

	Compound	Neutral.Formula	CAS	Theoretical.mz	I
0	LysoPC (13:0)	C21H44NO7P	20559-17-5	454.2933	[
1	Lithocholyltaurine	C26H45NO5S	6042-32-6	506.2916	[
2	LysoPC (16:0)	C24H50NO7P	17364-16-8	518.3223	[
3	LysoPC (18:1)	C26H52NO7P	19420-56-5	522.3559	[
4	1,2-Didecanoyl-sn-glycero-3-phosphoethanolamine	C25H50NO8P	253685-27-7	524.3352	[
5	LysoPC (18:0)	C26H54NO7P	19420-57-6	524.3716	[
6	LysoPC (2-18:0)	C26H54NO7P	27098-24-4	524.3716	[
7	Platelet-activating Factor	C26H54NO7P	74389-68-7	524.3716	[
8	LysoPC (19:0)	C27H56NO7P	108273-88-7	538.3872	[
9	PC (17:0/02:0)	C27H56NO7P	93037-84-4	538.3872	[
10	LysoPC (18:1)	C26H52NO7P	19420-56-5	544.3379	[
11	LysoPC (18:0)	C26H54NO7P	19420-57-6	546.3536	[
12	LysoPC (2-18:0)	C26H54NO7P	27098-24-4	546.3536	[
13	Platelet-activating Factor	C26H54NO7P	74389-68-7	546.3536	[
14	PC (16:0E/2:0)	C26H54NO7P	74389-68-8	546.3536	[
15	LysoPC (20:0)	C28H58NO7P	108341-80-6	552.4029	[
16	PC (O-18:0/2:0)	C28H58NO7P	74389-69-8	552.4029	[
17	1,2-Dilauroyl-sn-glycero-3-phosphocholine	C32H64NO8P	18194-25-7	622.4448	[
18	Cer 42:02	C42H81NO3	NaN	630.6189	[
19	Cer 42:01	C42H83NO3	NaN	632.6346	[
20	1,2-Dilauroyl-sn-glycero-3-phosphocholine	C32H64NO8P	18194-25-7	644.4268	[
21	PE (O-32:01)	C37H74NO7P	NaN	720.4920	[
22	PC (16:1/16:1) (del9-cis)	C40H76NO8P	4724-96-3	730.5387	[
23	1,2-Dipalmitoyl-sn-glycero-3-phosphocholine	C40H80NO8P	63-89-8	734.5700	[
24	SM 36:01	C41H83N2O6P	NaN	735.5781	[
25	1-Palmitoyl-2-oleoyl-sn-glycero-3-phosphoethan...	C39H76NO8P	26662-94-2	740.5207	[
26	PE 34:01	C39H76NO8P	NaN	740.5207	[
27	1,2-Diheptadecanoyl-sn-glycero-3-phosphoethano...	C39H78NO8P	140219-78-9	742.5363	[
28	PE 34:01	C39H76NO8P	NaN	756.4946	[
29	1,2-Dipalmitoyl-sn-glycero-3-phosphocholine	C40H80NO8P	63-89-8	756.5520	[
30	PC (18:1(9Z)/16:0)	C42H82NO8P	59491-62-2	760.5856	[
31	Di(2-ethylhexyl) adipate	C22H42O4	103-23-1	763.6064	[
32	PE 36:03	C41H76NO8P	NaN	764.5207	[
33	PE (18:1/18:1)(del9-cis)	C41H78NO8P	NaN	766.5363	[
34	PE 36:02	C41H78NO8P	NaN	766.5363	[
35	PE (18:1/18:1)(del9-cis)	C41H78NO8P	NaN	766.5363	[
36	GlcCer 36:01 OH	C42H81NO9	NaN	766.5809	[
37	PE 36:01	C41H80NO8P	NaN	768.5520	[
38	SM 36:01	C41H83N2O6P	NaN	769.5626	[
39	1,2-Distearoyl-sn-glycero-3-phosphoethanolamine	C41H82NO8P	1069-79-0	770.5676	[
40	PC 34:03	C42H78NO8P	NaN	778.5363	[

	Compound	Neutral.Formula	CAS	Theoretical.mz	I
41	PC (18:1(9Z)/16:0)	C42H82NO8P	59491-62-2	782.5676	[
42	PC (18:1/18:1) (del9-trans)	C44H84NO8P	56782-46-8	786.6013	[
43	PE 36:01	C41H80NO8P	NaN	790.5339	[
44	1,2-Distearoyl-sn-glycero-3-phosphocholine	C44H88NO8P	816-94-4	790.6326	[
45	GlcCer 38:01 OH	C44H85NO9	NaN	794.6122	[
46	PC 34:02	C42H80NO8P	NaN	796.5259	[
47	PE (O-38:05)	C43H78NO7P	NaN	796.5233	[
48	PC (18:1/18:1) (del9-trans)	C44H84NO8P	56782-46-8	808.5833	[
49	1,2-Distearoyl-sn-glycero-3-phosphocholine	C44H88NO8P	816-94-4	812.6146	[
50	d7-TG (15:0-18:1)	C51H89D7O6	2097561-17-4	812.7724	[
51	SM 42:02	C47H93N2O6P	NaN	817.6563	[
52	SM 42:00	C47H97N2O6P	NaN	817.7162	[
53	PS 36:04	C42H74NO10P	NaN	828.4767	[
54	PE 39:02	C44H84NO8P	NaN	830.5652	[
55	PS 37:04	C43H76NO10P	NaN	842.4924	[
56	1,2-Diarachidoyl-sn-glycero-3-phosphocholine	C48H96NO8P	61596-53-0	846.6952	[
57	1,2-Diarachidoyl-sn-glycero-3-phosphocholine	C48H96NO8P	61596-53-0	868.6772	[
58	Diisononyl hexahydrophthalate	C26H48O4	166412-78-8	871.7003	[

We got some matches and we can view their ion images:

```
image, _  = timsimaging.plotting.image(dataset, i=539, results=results)
show(image)
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): application/javascript, application/vnd.bokehjs_ex
```

2.0.5 Compare with annotation results from METASPACE

Here we compare using LIPID MAPS

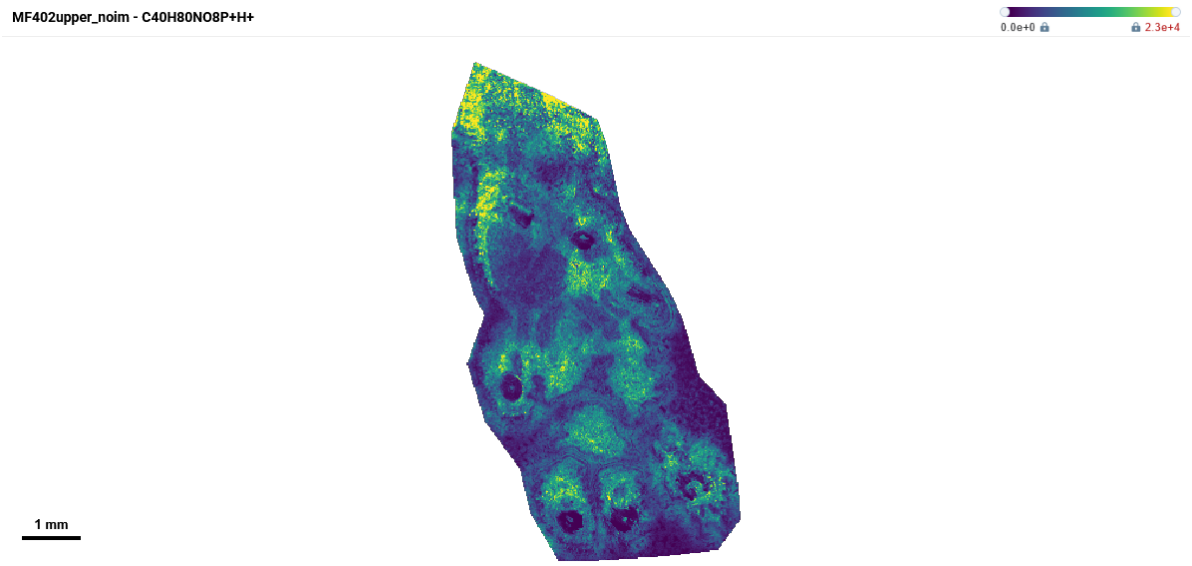
```
metaspace_results = pd.read_csv(r"D:\dataset\Melanie\metaspace_annotations.csv", header=2)
metaspace_results
```

	group	datasetName	datasetId	formula	adduct	chemMod	ion
0	NaN	MF402upper_noim	2026-01-07_13h02m29s	C40H80NO8P	M+H	NaN	C40H80NO

	group	datasetName	datasetId	formula	adduct	chemMod	ion
1	NaN	MF402upper_noim	2026-01-07_13h02m29s	C38H77N2O7P	M+H	NaN	C38H77N2
2	NaN	MF402upper_noim	2026-01-07_13h02m29s	C41H83N2O6P	M+H	NaN	C41H83N2
3	NaN	MF402upper_noim	2026-01-07_13h02m29s	C45H91N2O6P	M+H	NaN	C45H91N2
4	NaN	MF402upper_noim	2026-01-07_13h02m29s	C33H64NO9P	M+H	NaN	C33H64NO
...	...	...	...	...	...	...	...
91	NaN	MF402upper_noim	2026-01-07_13h02m29s	C35H64NO10P	M+H	NaN	C35H64NO
92	NaN	MF402upper_noim	2026-01-07_13h02m29s	C36H68NO10P	M+H	NaN	C36H68NO
93	NaN	MF402upper_noim	2026-01-07_13h02m29s	C40H81N2O6P	M+H	NaN	C40H81N2
94	NaN	MF402upper_noim	2026-01-07_13h02m29s	C39H74NO8P	M+H	NaN	C39H74NO
95	NaN	MF402upper_noim	2026-01-07_13h02m29s	C46H80NO10P	M+H	NaN	C46H80NO

```
metaspace_ions = metaspace_results[["formula", "adduct"]].copy()
metaspace_ions["adduct"] = metaspace_ions["adduct"].map(lambda x: f"[{x}]".format(x))
timsimaging_ions = final_results[["Neutral.Formula", "Ion.Species"]].copy()
timsimaging_ions.columns = ["formula", "adduct"]
intersection = timsimaging_ions.merge(
    metaspace_ions,
    on=["formula", "adduct"],
    how="inner"
)
intersection
```

	formula	adduct
0	C40H76NO8P	[M+H]
1	C40H80NO8P	[M+H]
2	C39H78NO8P	[M+Na]
3	C40H80NO8P	[M+Na]
4	C41H76NO8P	[M+Na]
5	C41H80NO8P	[M+Na]
6	C41H82NO8P	[M+Na]
7	C42H78NO8P	[M+Na]
8	C42H82NO8P	[M+Na]
9	C48H96NO8P	[M+Na]



{#ion image from metaspace width="60%"}