sheet01

October 27, 2024

Names: Philipp Köhler, Alexander Bespalov

1 Sheet 1

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pyplot as plt
%matplotlib inline
```

1.1 1 Principal Component Analysis

1.1.1 (a)

[2]:
```
# TODO: implement PCA (fill in the blanks in the function below)

def pca(data, n_components=None):
    """
    Principal Component Analysis on a p x N data matrix.

    Parameters
    ----------
    data : np.ndarray
        Data matrix of shape (p, N).
    n_components : int, optional
        Number of requested components. By default returns all components.

    Returns
    -------
    np.ndarray, np.ndarray
        the pca components (shape (n_components, p)) and the projection (shape␣
↪(n_components, N))

    """
    # set n_components to p by default
    n_components = data.shape[0] if n_components is None else n_components
    assert n_components <= data.shape[0], f"Got n_components larger than␣
↪dimensionality of data!"
```

```python
    # center the data
    C = (np.eye(data.shape[1]) - np.ones((data.shape[1],1)) @ np.ones((data.
↪shape[1],1)).T/data.shape[1])
    x_centered = data @ C

    # compute X times X transpose
    xxt = x_centered @ x_centered.T

    # compute the eigenvectors and eigenvalues
    eigenvalues, eigenvectors = np.linalg.eig(xxt)

    # sort the eigenvectors by eigenvalue and take the n_components largest ones
    sorted_indices = np.argsort(eigenvalues)[::-1]
    components = eigenvectors[:, sorted_indices][:, :n_components].T #directly␣
↪use only first n_components and traspose for correct dimensions

    # compute X_projected, the projection of the data to the components
    X_projected = components @ x_centered

    return components, X_projected  # return the n_components first components␣
↪and the pca projection of the data
```

[3]:

```python
# Example data to test your implementation
# All the asserts on the bottom should go through if your implementation is␣
↪correct

data = np.array([
    [ 1,  0,  0, -1,  0,  0],
    [ 0,  3,  0,  0, -3,  0],
    [ 0,  0,  5,  0,  0, -5]
], dtype=np.float32)

# add a random offset to all samples. it should not affect the results
data += np.random.randn(data.shape[0], 1)

n_components = 2
components, projection = pca(data, n_components=n_components)  # apply your␣
↪implementation

# the correct results are known (up to some signs)
true_components = np.array([[0, 0, 1], [0, 1, 0]], dtype=np.float32)
true_projection = np.array([
    [ 0,  0,  5,  0,  0, -5],
    [ 0,  3,  0,  0, -3,  0]
], dtype=np.float32)
```

```
# check that components match, up to sign
assert isinstance(components, np.ndarray), f'Expected components to be numpy␣
↪array but got {type(components)}'
assert components.shape == true_components.shape, f'{components.shape}!
↪={true_components.shape}'
assert np.allclose(np.abs(components * true_components).sum(1), np.
↪ones(n_components)), f'Components not matching'

# check that projections agree, taking into account potentially flipped␣
↪components
assert isinstance(projection, np.ndarray), f'Expected projection to be numpy␣
↪array but got {type(projection)}'
assert projection.shape == (n_components, data.shape[1]), f'Incorrect shape of␣
↪projection: Expected {(n_components, data.shape[1])}, got {projection.shape}'
assert np.allclose(projection, true_projection * (components * true_components).
↪sum(1, keepdims=True), atol=1e-6), f'Projections not matching'

print('Test successful!')
```

```
Test successful!
```

1.1.2 (b)

Load the data (it is a subset of the data at https://opendata.cern.ch/record/4910#)

[4]:
```
features = np.load('data/dijet_features.npy')
labels = np.load('data/dijet_labels.npy')
label_names = ['b', 'c', 'q']  # bottom, charm or light quarks

print(f'{features.shape=}, {labels.shape=}')  # print the shapes

# TODO: print how many samples of each class are present in the data (hint:␣
↪numpy.unique)
unique_labels, counts = np.unique(labels, return_counts=True)

for label, count in zip(unique_labels, counts):
    print(f'Class {label_names[int(label)]}: {count} samples')
```

```
features.shape=(116, 2233), labels.shape=(2233,)
Class b: 999 samples
Class c: 864 samples
Class q: 370 samples
```

Normalize the data

[5]:
```
# TODO: report range of features and normalize the data to zero mean and unit␣
↪variance
```

```
feature_min = np.min(features, axis=1)
feature_max = np.max(features, axis=1)
feature_range = feature_max - feature_min

print("Feature ranges (min, max):")
for i in range(features.shape[0]):
    print(f"Feature {i}: min={feature_min[i]}, max={feature_max[i]},␣
↪range={feature_range[i]}")

mean = np.mean(features, axis=1,keepdims=True)
std_dev = np.std(features, axis=1,keepdims=True)

normalized_features = (features - mean) / std_dev

print("\nNormalized Features:")
print(f"Mean of normalized features: {np.mean(normalized_features, axis=1)}")
print(f"Standard deviation of normalized features: {np.std(normalized_features,␣
↪axis=1)}")
print(mean.shape)
```

```
Feature ranges (min, max):
Feature 0: min=0.5857568129500379, max=1.0563076922743733,
range=0.47055087932433537
Feature 1: min=-0.324, max=0.3001, range=0.6241
Feature 2: min=-139.62783040862374, max=145.8169283242588,
range=285.4447587328825
Feature 3: min=11.65252474327396, max=20713.95643348941,
range=20702.303908746137
Feature 4: min=99860.37217253156, max=100145.81693106721,
range=285.4447585356538
Feature 5: min=-0.9999999932105357, max=-0.9782917490360855,
range=0.021708244174450164
Feature 6: min=149256.83013332827, max=2725986.9412642987,
range=2576730.1111309705
Feature 7: min=94.02340841282924, max=231315.8411330428,
range=231221.81772462997
Feature 8: min=155058.36476195962, max=2728662.5177883604,
range=2573604.1530264006
Feature 9: min=-134437.84306900043, max=59292.72195058862,
range=193730.56501958903
Feature 10: min=-96619.36772098255, max=188237.84080264496,
range=284857.2085236275
Feature 11: min=149077.9934216972, max=2725850.600345601,
range=2576772.606923904
Feature 12: min=12928.3681490856, max=197132.54104026855,
range=184204.17289118294
Feature 13: min=12928.3681490856, max=197132.54104026855,
```

range=184204.17289118294
Feature 14: min=0.6189, max=0.9555, range=0.3366
Feature 15: min=-0.4623, max=0.3001, range=0.7624
Feature 16: min=-139.6418, max=141.8433, range=281.4851
Feature 17: min=0.5800954040409998, max=1.1115254680669437,
range=0.531430064025944
Feature 18: min=-0.443773666216847, max=0.302507038961965,
range=0.7462807051788121
Feature 19: min=-139.64419806197225, max=141.8362443627514,
range=281.4804424247236
Feature 20: min=-1.0, max=29.108757115711636, range=30.108757115711636
Feature 21: min=0.0, max=199.78329390377166, range=199.78329390377166
Feature 22: min=99860.35820293978, max=100141.84330330986,
range=281.4851003700751
Feature 23: min=0.9465362291424876, max=0.9998329246853725,
range=0.053296695542884964
Feature 24: min=17008.196317376653, max=253797.09047180956,
range=236788.8941544329
Feature 25: min=-155073.08929859565, max=50680.191555756384,
range=205753.28085435202
Feature 26: min=-57857.23599283395, max=200911.1746701653,
range=258768.41066299923
Feature 27: min=51083.047356840725, max=2462438.26731527,
range=2411355.219958429
Feature 28: min=54037.6894483653, max=2463166.2537098792,
range=2409128.5642615138
Feature 29: min=1.7974219811405918, max=4.6951916498513135,
range=2.8977696687107217
Feature 30: min=-3.1408206952421707, max=3.1406397690503645,
range=6.281460464292535
Feature 31: min=2166.4479443550063, max=18752.819663248687,
range=16586.37171889368
Feature 32: min=2166.4479443551163, max=18752.819663243805,
range=16586.371718888688
Feature 33: min=1079.4495219263306, max=240631.91904827222,
range=239552.4695263459
Feature 34: min=0.04829200017432982, max=0.948127177506002,
range=0.8998351773316722
Feature 35: min=0.1689743729829774, max=1.0, range=0.8310256270170227
Feature 36: min=2.0, max=27.0, range=25.0
Feature 37: min=810.9800170345983, max=16913.598170313464,
range=16102.618153278865
Feature 38: min=0.6189, max=0.9555, range=0.3366
Feature 39: min=-0.4623, max=0.3001, range=0.7624
Feature 40: min=-139.6418, max=141.8433, range=281.4851
Feature 41: min=-1.0, max=0.48710565762907054, range=1.4871056576290704
Feature 42: min=-1.0, max=105697.93163667158, range=105698.93163667158
Feature 43: min=-59620.990000000005, max=53280.71000000001,

```
range=112901.70000000001
Feature 44: min=-59901.05, max=96787.20999999999, range=156688.26
Feature 45: min=-1.0, max=2293534.37, range=2293535.37
Feature 46: min=-1.0, max=2294214.8532233015, range=2294215.8532233015
Feature 47: min=-1.0, max=4.87578563746712, range=5.87578563746712
Feature 48: min=-3.1415658912198636, max=3.140280472380842,
range=6.281846363600706
Feature 49: min=-1.0, max=19876.909167249592, range=19877.909167249592
Feature 50: min=-999.0, max=96599.07071007465, range=97598.07071007465
Feature 51: min=53968.27870894955, max=2463152.8394500166,
range=2409184.560741067
Feature 52: min=1.7974219811405918, max=4.6951916498513135,
range=2.8977696687107217
Feature 53: min=-3.1408206952421707, max=3.1406397690503645,
range=6.281460464292535
Feature 54: min=2166.4479443550063, max=18752.819663248687,
range=16586.37171889368
Feature 55: min=2166.4479443551163, max=18752.819663243805,
range=16586.371718888688
Feature 56: min=2.0, max=27.0, range=25.0
Feature 57: min=0.04829200017432982, max=0.948127177506002,
range=0.8998351773316722
Feature 58: min=0.8130122698360408, max=1.0, range=0.1869877301639592
Feature 59: min=0.1689743729829774, max=1.0, range=0.8310256270170227
Feature 60: min=810.9800170345983, max=16913.598170313464,
range=16102.618153278865
Feature 61: min=4.0, max=42.0, range=38.0
Feature 62: min=0.0, max=415590.21343112877, range=415590.21343112877
Feature 63: min=-14238.933261109574, max=20893.953677062935,
range=35132.88693817251
Feature 64: min=-15763.658295106707, max=14077.854629241412,
range=29841.51292434812
Feature 65: min=0.0, max=414952.9910297136, range=414952.9910297136
Feature 66: min=-99.0, max=23004.99014419313, range=23103.99014419313
Feature 67: min=-99.0, max=10.19875405869562, range=109.19875405869561
Feature 68: min=-1.0, max=0.9997606873512268, range=1.9997606873512268
Feature 69: min=-1.0, max=1.5778650366867029, range=2.5778650366867026
Feature 70: min=0.6189, max=0.9555, range=0.3366
Feature 71: min=-0.4623, max=0.3001, range=0.7624
Feature 72: min=-139.6418, max=141.8433, range=281.4851
Feature 73: min=0.6406772697526835, max=0.9320724208937139,
range=0.29139515114103043
Feature 74: min=-0.45013114549282307, max=0.2259, range=0.6760311454928231
Feature 75: min=-139.6259498422301, max=141.84220005099877,
range=281.4681498932289
Feature 76: min=-1.0, max=32.17404835910952, range=33.17404835910952
Feature 77: min=0.0, max=169.27122712076616, range=169.27122712076616
Feature 78: min=99860.35820293978, max=100141.84330330986,
```

range=281.4851003700751
Feature 79: min=0.9501617396913894, max=0.9998485019807651,
range=0.04968676228937574
Feature 80: min=17000.631265901116, max=142602.28704603694,
range=125601.65578013583
Feature 81: min=-95802.77105300459, max=65673.25265542942,
range=161476.02370843402
Feature 82: min=-105627.84353249811, max=53118.46547159311,
range=158746.30900409122
Feature 83: min=55591.29511756872, max=2110943.2850531084,
range=2055351.9899355397
Feature 84: min=58285.54228954092, max=2111684.495802592,
range=2053398.9535130512
Feature 85: min=1.833425554610007, max=4.74372706253242, range=2.910301507922413
Feature 86: min=-3.140454823153266, max=3.1386947383770276,
range=6.279149561530294
Feature 87: min=1792.527977990827, max=17332.61065648541,
range=15540.082678494582
Feature 88: min=1792.5279779567772, max=17332.61065648541,
range=15540.082678528632
Feature 89: min=1069.5623163565951, max=64763.93009149253,
range=63694.367775135936
Feature 90: min=0.06113612263625889, max=0.8125790285580228,
range=0.7514429059217639
Feature 91: min=0.20981383892222163, max=1.0000427631918138,
range=0.7902289242695922
Feature 92: min=2.0, max=27.0, range=25.0
Feature 93: min=1024.6749568765124, max=12294.218303200092,
range=11269.54334632358
Feature 94: min=0.6189, max=0.9555, range=0.3366
Feature 95: min=-0.4623, max=0.3001, range=0.7624
Feature 96: min=-139.6418, max=141.8433, range=281.4851
Feature 97: min=58208.66487941796, max=2111670.5381511813,
range=2053461.8732717633
Feature 98: min=1.833425554610007, max=4.74372706253242, range=2.910301507922413
Feature 99: min=-3.140454823153266, max=3.1386947383770276,
range=6.279149561530294
Feature 100: min=1792.527977990827, max=17332.61065648541,
range=15540.082678494582
Feature 101: min=1792.5279779567772, max=17332.61065648541,
range=15540.082678528632
Feature 102: min=2.0, max=27.0, range=25.0
Feature 103: min=0.06113612263625889, max=0.8125790285580228,
range=0.7514429059217639
Feature 104: min=0.8125394541188198, max=1.0, range=0.18746054588118022
Feature 105: min=0.20981383892222163, max=1.0000427631918138,
range=0.7902289242695922
Feature 106: min=1024.6749568765124, max=12294.218303200092,

```
range=11269.54334632358
Feature 107: min=4.0, max=40.0, range=36.0
Feature 108: min=0.0, max=831394.9967138312, range=831394.9967138312
Feature 109: min=-15417.909973212973, max=22330.617737374363,
range=37748.52771058734
Feature 110: min=-15962.145199447563, max=12408.96287347479,
range=28371.108072922354
Feature 111: min=0.0, max=831005.03566094, range=831005.03566094
Feature 112: min=-99.0, max=25460.95252067779, range=25559.95252067779
Feature 113: min=-99.0, max=10.263283563462764, range=109.26328356346276
Feature 114: min=-1.0, max=0.9999180436134338, range=1.9999180436134338
Feature 115: min=-1.0, max=1.1102195152776082, range=2.110219515277608

Normalized Features:
Mean of normalized features: [ 5.75560896e-15  3.24962727e-16  9.53111302e-17
1.59294382e-15
  4.08635592e-12  1.13008282e-12  8.75748695e-16 -2.17803583e-15
  1.31327501e-15 -2.30571396e-17 -2.76685675e-17  2.95002118e-15
  4.15873734e-15  4.15873734e-15 -4.41255233e-15  3.55490131e-16
 -7.43794736e-17 -3.98334895e-14 -5.07132774e-17  3.40077272e-17
  1.67739137e-16 -4.97636465e-16 -2.51342555e-12 -9.10711025e-15
 -2.32923100e-15 -1.89429007e-17  8.62436460e-17 -2.01465952e-15
 -5.19313905e-16  4.27090318e-15  5.04662367e-17 -5.56349513e-15
  6.82272570e-15 -5.31196722e-16  1.56544927e-15 -5.52754836e-15
  4.90228349e-17  5.90334867e-15 -4.41255233e-15  3.55490131e-16
 -7.43794736e-17  1.53333086e-15 -1.53830275e-15 -3.06268421e-17
 -3.01366448e-17  1.05632774e-15  1.04151151e-15 -4.25220888e-15
  1.17044504e-16  2.29810697e-15  5.63315131e-15 -2.15939124e-15
  4.27090318e-15  5.04662367e-17 -5.56349513e-15  6.82272570e-15
  4.90228349e-17  1.56544927e-15 -2.26112605e-14 -5.52754836e-15
  5.90334867e-15  1.39809545e-16 -5.28114150e-16 -8.94225481e-16
  2.37208869e-16 -3.00128137e-16 -3.09525008e-16  5.79448914e-16
 -1.02828628e-15  2.13418376e-16 -4.41255233e-15  3.55490131e-16
 -7.43794736e-17 -1.33940229e-14 -1.20995914e-15 -1.16739976e-16
 -4.96169757e-16 -8.22052283e-16 -2.51342555e-12  1.28791763e-13
 -3.01858354e-15  8.35277511e-18  1.24968454e-16 -3.91362317e-16
 -2.61750118e-15 -1.15535225e-14  1.87937440e-17  4.09733450e-15
  1.88076653e-15 -1.18400587e-15 -9.27456350e-16  4.12646978e-15
  6.86742299e-17 -5.05544256e-15 -4.41255233e-15  3.55490131e-16
 -7.43794736e-17  7.76982101e-16 -1.15535225e-14  1.87937440e-17
  4.09733450e-15  1.88076653e-15  6.86742299e-17 -9.27456350e-16
  2.10822055e-14  4.12646978e-15 -5.05544256e-15 -3.89199545e-16
 -3.55564709e-16 -2.97303482e-18  2.35868013e-16  6.27141768e-16
  1.74737072e-16  1.14815854e-15  5.68137864e-16  2.42901683e-16]
Standard deviation of normalized features: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
```

```
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
(116, 1)
```

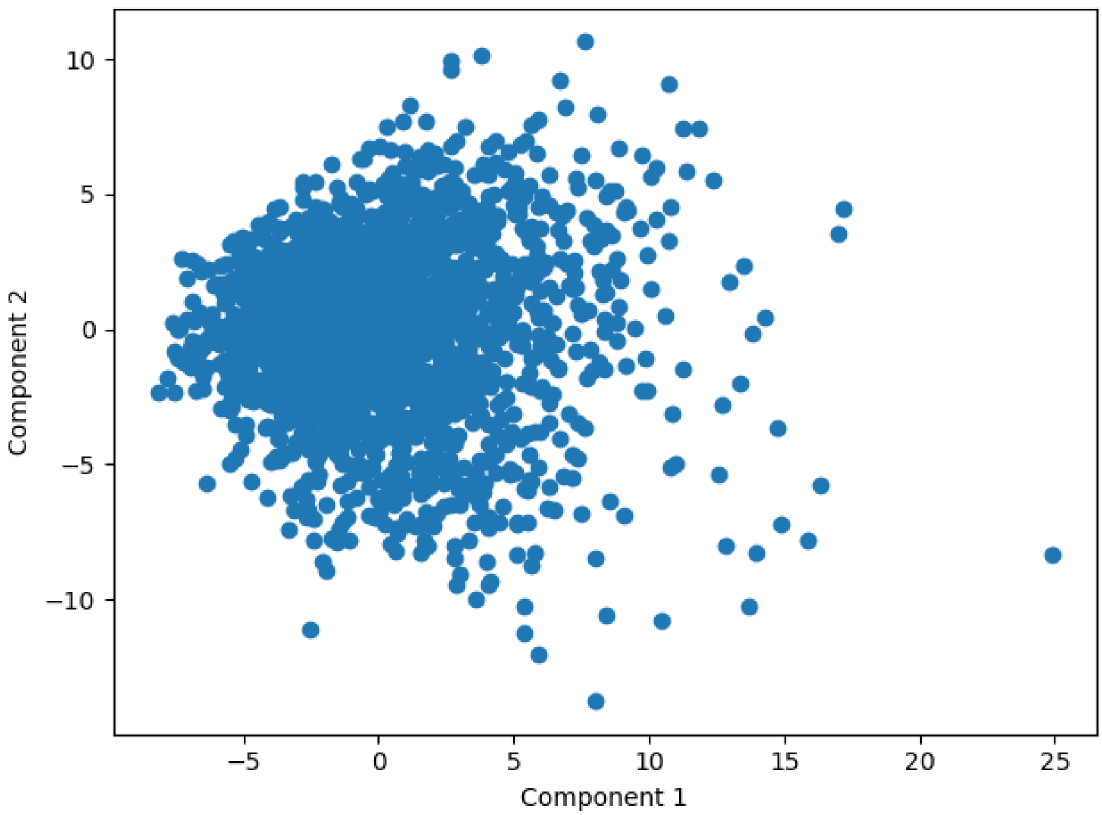
1.1.3 (c)

Compute a 2D PCA projection and make a scatterplot of the result, once without color, once coloring the dots by label. Interpret your results.

[6]:
```
# TODO: apply PCA as implemented in (a)
n_components = 2
components, projection = pca(normalized_features, n_components=n_components)
```

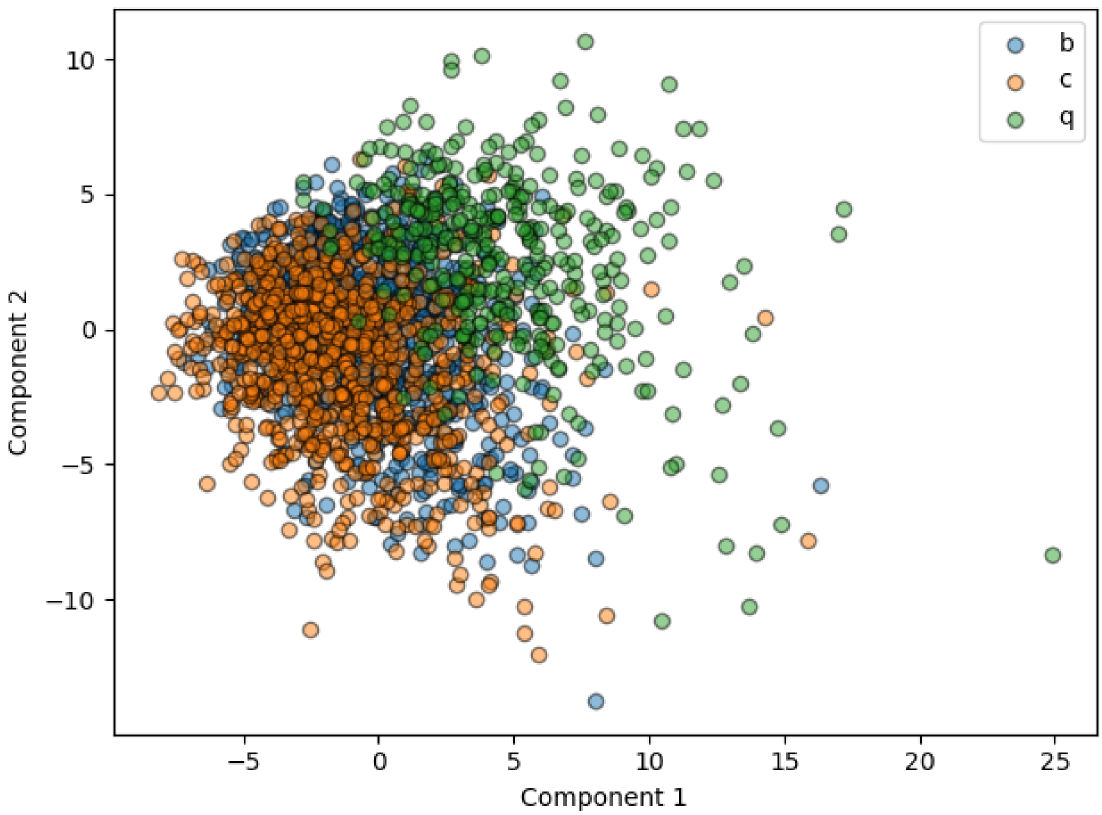
[7]:
```
# TODO: make a scatterplot of the PCA projection
plt.scatter(projection[0], projection[1])
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.tight_layout()
```

```
c:\Users\sasch\miniconda3\envs\mlph\lib\site-packages\matplotlib\cbook.py:1762:
ComplexWarning: Casting complex values to real discards the imaginary part
  return math.isfinite(val)
c:\Users\sasch\miniconda3\envs\mlph\lib\site-
packages\matplotlib\collections.py:197: ComplexWarning: Casting complex values
to real discards the imaginary part
  offsets = np.asanyarray(offsets, float)
```



No clusters of classes can be seen.

[8]:
```
# TODO: make a scatterplot, coloring the dots by their label and including a␣
↪legend with the label names
# (hint: one way is to call plt.scatter once for each of the three possible␣
↪labels. Why could it be problematic to scatter the data sorted by labels␣
↪though?)
for label in np.unique(labels):
    plt.scatter(
        projection[0][labels == label],
        projection[1][labels == label],
        label=label_names[int(label)],
        alpha=0.5,
        edgecolor='k'
    )
plt.legend()
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.tight_layout()
```



The classes b and c are well mixed. The class q is overlappig but more distinct.

1.2 2 Nonlinear Dimension Reduction

[9]:
```
import umap  # import umap-learn, see https://umap-learn.readthedocs.io/
```

[10]:
```
# if you have not done 1(b) yet, you can load the normalized features directly:
features = np.load('data/dijet_features_normalized.npy')
labels = np.load('data/dijet_labels.npy')
label_names = ['b', 'c', 'q']  # bottom, charm or light quarks
```

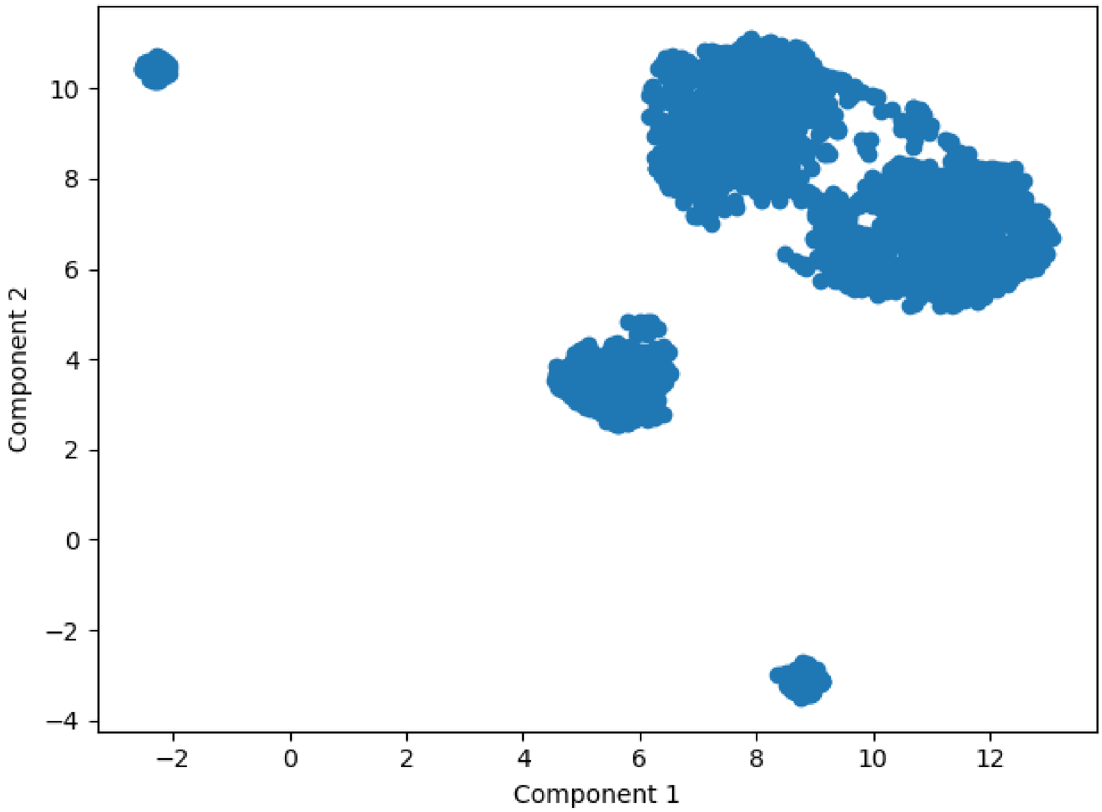
1.2.1 (a)

[11]:
```
# TODO: Apply umap on the normalized jet features from excercise 1. It will␣
↪take a couple of seconds.
# note: umap uses a different convention regarding the feature- and sample␣
↪dimension, N x p instead of p x N!

reducer = umap.UMAP()
umap_projection = reducer.fit_transform(normalized_features.T)
```

[12]:

```python
# TODO: make a scatterplot of the UMAP projection
plt.scatter(umap_projection[:,0], umap_projection[:,1])
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.tight_layout()
```

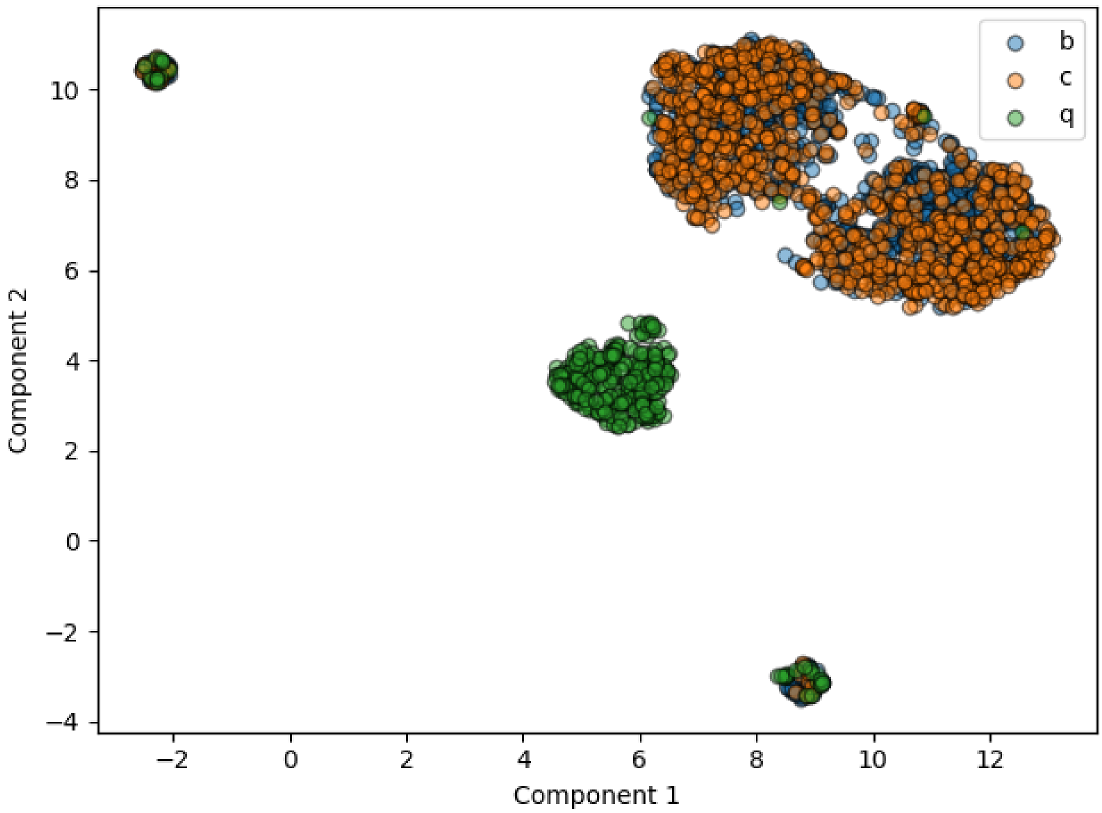


5 cluster groups are visible.

[13]:

```python
# TODO: make a scatterplot, coloring the dots by their label and including a␣
↪legend with the label names
for label in np.unique(labels):
    plt.scatter(
        umap_projection[labels == label, 0],
        umap_projection[labels == label, 1],
        label=label_names[int(label)],
        alpha=0.5,
        edgecolor='k'
    )
plt.legend()
plt.xlabel("Component 1")
plt.ylabel("Component 2")
```

```
plt.tight_layout()
```



At (6,2) a q cluster is clearly seen. At (4.5, -4) and (-2,6) are two small clusters with q and c dominating and b also contained. The two clusters around (10, 8) are dominated by c and b with a low amount of q.

1.2.2 (b)

[14]:
```
plt.figure(figsize=(15, 10))

n_neighbors_values = (2, 4, 8, 15, 30, 60, 100)

for i, n_neighbors in enumerate(n_neighbors_values):
    reducer = umap.UMAP(n_neighbors=n_neighbors)
    umap_projection = reducer.fit_transform(normalized_features.T)

    # Create a subplot for the current n_neighbors
    plt.subplot(3, 3, i + 1)  # 3 rows, 3 columns
    plt.title(f'UMAP with n_neighbors={n_neighbors}')

    for label in np.unique(labels):
```

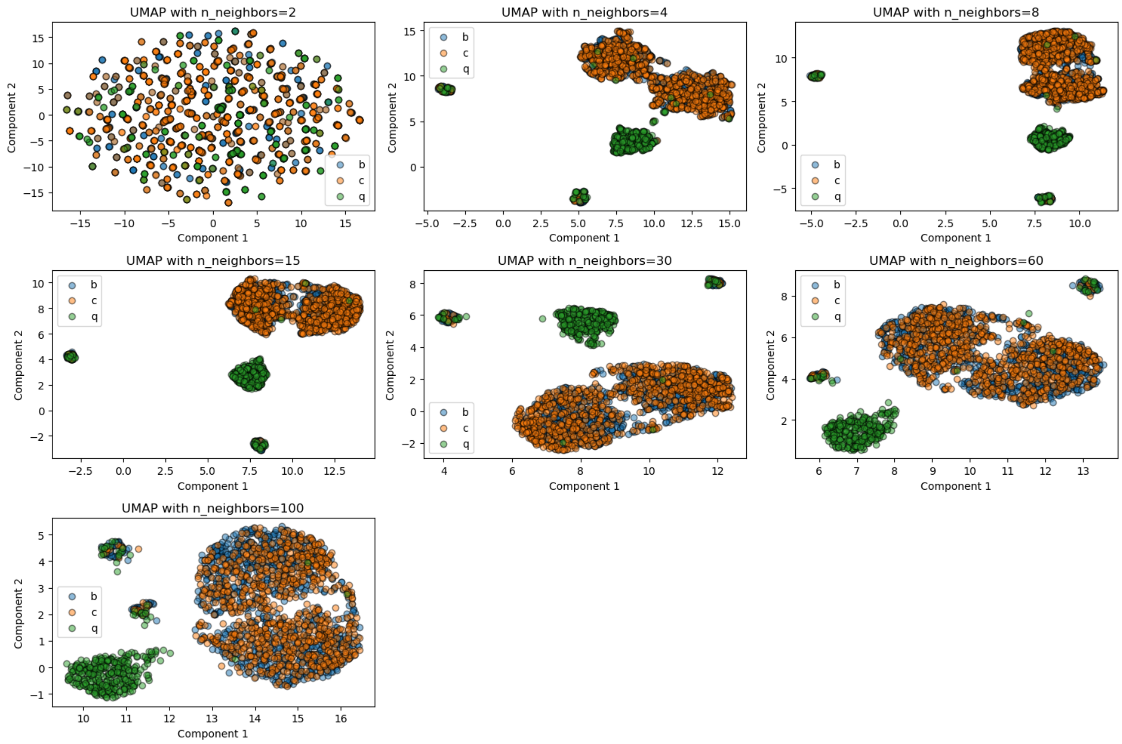
```
        plt.scatter(
            umap_projection[labels == label, 0],
            umap_projection[labels == label, 1],
            label=label_names[int(label)],
            alpha=0.5,
            edgecolor='k'
        )
    plt.legend()
    plt.xlabel("Component 1")
    plt.ylabel("Component 2")
    plt.tight_layout()


#plt.subplot(3, 3, len(n_neighbors_values))  # Select the last subplot for the␣
 ↪legend
#plt.legend(title='Label', loc='upper right', bbox_to_anchor=(1.2, 1))
plt.tight_layout()
plt.show()
```

```
c:\Users\sasch\miniconda3\envs\mlph\lib\site-
packages\sklearn\manifold\_spectral_embedding.py:455: UserWarning: Exited at
iteration 648 with accuracies
[1.38043604e-15 5.97961793e-06 1.68046949e-06 4.60862444e-06]
not reaching the requested tolerance 4.664063453674316e-06.
Use iteration 648 instead with accuracy
3.0671779640798114e-06.

  _, diffusion_map = lobpcg(
c:\Users\sasch\miniconda3\envs\mlph\lib\site-
packages\sklearn\manifold\_spectral_embedding.py:455: UserWarning: Exited
postprocessing with accuracies
[1.68076127e-15 5.97960806e-06 1.68052754e-06 4.60861606e-06]
not reaching the requested tolerance 4.664063453674316e-06.
  _, diffusion_map = lobpcg(
c:\Users\sasch\miniconda3\envs\mlph\lib\site-packages\umap\spectral.py:550:
UserWarning: Spectral initialisation failed! The eigenvector solver
failed. This is likely due to too small an eigengap. Consider
adding some noise or jitter to your data.

Falling back to random initialisation!
  warn(
```



For 2 neighbours the dataset is mixed. A value above 4 n_neighbours does not improve the clustering. It just changes the postitions of the clusters. Higher values for neighbours thend to spread the clusters.

1.3 3 RANSAC

Probability to sample m inliers:

$$p^m$$

Probabiltiy to not sample m inliers:

$$1 - p^m$$

Probability to sample r times not m inliers:

$$(1 - p^m)^r$$

Probability to not sample r times not m inliers:

$$1 - (1 - p^m)^r \stackrel{!}{=} 0.99$$

$$\Leftrightarrow r = \frac{log(1 - 0.99)}{log(1 - p^m)}$$

1.4 4 Bonus: PCA meets Random Matrix Theory

1.4.1 (a)

The directional distribution of all principal componets is isotropic gaussian, because they are random variables of an isotropic gaussian distributed random variable $\mathbf{X}$ with PCA applied wich conserves the property due to linearity.

1.4.2 (b)

Largest eigenvalues will grow with N. Middle eigenvalues are influenced by the ratio of p and N. Most eigenvalues will be zero.

1.4.3 (c)

According to Marchenko–Pastur the probability mass of the eigenvalues is in the reange of $(1 \pm \sqrt{\lambda})^2$.