Assignment 3

Olatomiwa Akinlaja

Kudzai Sibanda

1 Astronomy Data

Performing the following manifold methods and comparing the differences:

- UMAP
- Modified LLE
- Spectral Embedding (sklearn's implementation of Laplacian Eigenmaps)
- ISOMAP

Investigating the effect of:

- Dataset size
- Number of clusters
- Number of neighbours
- The impact of the above on the 2 or 3D embedding (ie visualisation) - does it show a visual difference etween classes?

- choose a suitable baseline for each, and then iterate. For all sections
- show 2/3d plots of the embedding to motivate your answer.

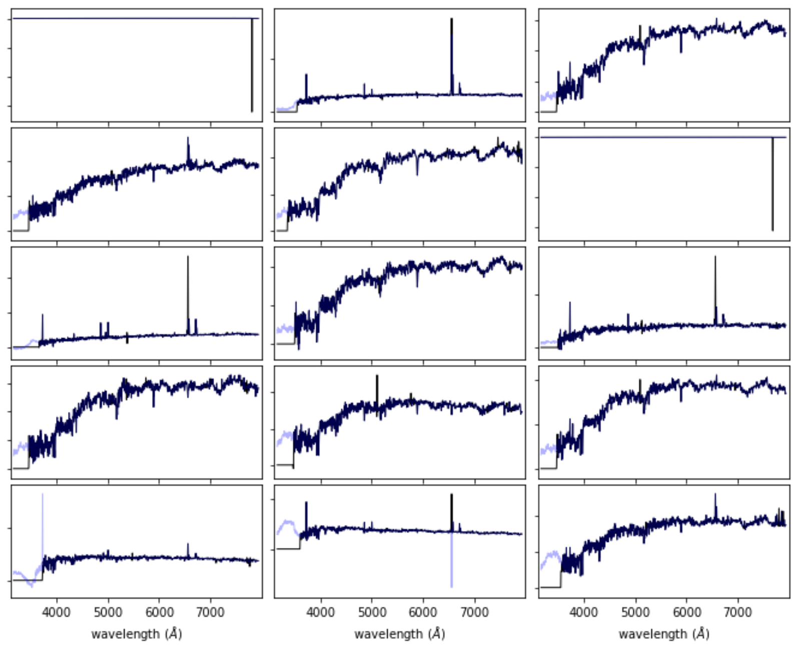
In [1]:
```python
import numpy as np
from matplotlib import pyplot as plt

from astroML.datasets import sdss_corrected_spectra
import seaborn as sns

from sklearn.manifold import LocallyLinearEmbedding, Isomap, SpectralEmbedding

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import kneighbors_graph
from sklearn.metrics.cluster import adjusted_mutual_info_score
from sklearn import manifold, neighbors

from scipy.sparse.csgraph import laplacian as csgraph_laplacian
import scipy.spatial as spt
from scipy.sparse.linalg import eigsh, eigs

from umap import UMAP
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler

random_state = 25
np.random.seed(random_state)
#from sklearn.utils import resample
from plotly.subplots import make_subplots
import plotly.graph_objects as go
```

In [2]:
```python
#Loading the data
data = sdss_corrected_spectra.fetch_sdss_corrected_spectra()
spectra = sdss_corrected_spectra.reconstruct_spectra(data)
y = data['lineindex_cln']

# lookup for classes, 0-6
cdict = ['unknown', 'star', 'absorption galaxy',
         'galaxy', 'emission galaxy',
         'narrow-line QSO', 'broad-line QSO']
```

In [3]:
```python
# Normalise the data
# sc = MinMaxScaler(feature_range = (0, 1))
# spectra_normalized = sc.fit_transform(spectra)
```

In [4]:
```python
#----------------------------------------------------------------------
# Use pre-computed PCA to reconstruct spectra
spectra_raw = data['spectra']
spectra_corr = sdss_corrected_spectra.reconstruct_spectra(data) # spectra_normalized
wavelengths = sdss_corrected_spectra.compute_wavelengths(data)

#------------------------------------------------------------
# select random spectra
nrows = 5
ncols = 3
ind = np.random.randint(spectra_corr.shape[0], size=nrows * ncols)
spec_sample_raw = spectra_raw[ind]
spec_sample_corr = spectra_corr[ind]
```

In [5]:
```python
fig = plt.figure(figsize=(10, 8))

fig.subplots_adjust(left=0.05, right=0.95, wspace=0.05,
                    bottom=0.1, top=0.95, hspace=0.05)

for i in range(ncols):
    for j in range(nrows):
        ax = fig.add_subplot(nrows, ncols, ncols * j + 1 + i)
        ax.plot(wavelengths, spec_sample_raw[ncols * j + i], '-k', lw=1)
        ax.plot(wavelengths, spec_sample_corr[ncols * j + i], '-b', lw=1, alpha=0.3) # , c='blue')
        ax.set_xlim(3100, 7999)

        ax.yaxis.set_major_formatter(plt.NullFormatter())
        ax.xaxis.set_major_locator(plt.MultipleLocator(1000))
        if j < nrows - 1:
            ax.xaxis.set_major_formatter(plt.NullFormatter())
        else:
            plt.xlabel(r'wavelength $(\AA)$')

        ylim = ax.get_ylim()
        dy = 0.05 * (ylim[1] - ylim[0])
        ax.set_ylim(ylim[0] - dy, ylim[1] + dy)

plt.show()
```



In [6]:
```
# Compute PCA components
#  because the spectra have been reconstructed from masked values,
#  we'll use the values computed
#  in the file compute_sdss_pca.py
evals = data['evals'] ** 2
evals_cs = evals.cumsum()
evals_cs /= evals_cs[-1]
evecs = data['evecs']
spec_mean = spectra_corr.mean(0)
```

In [7]:
```
def plot_2d(X, labels, title):
    plt.figure()
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap=plt.cm.jet)
    plt.xlabel('coefficient 1')
    plt.ylabel('coefficient 2')
    plt.title(title)

def plot_3d(X, labels, title):
    fig = px.scatter_3d(
    X, x=0, y=1, z=2,
    color=labels, labels=labels,title=title )
    fig.update_traces(marker_size=8)
    fig.show()


def mask_outliers(X):
    BT = neighbors.BallTree(X)
    dist, ind = BT.query(X, 10)
    dist_to_n = dist[:, -1]
    dist_to_n -= dist_to_n.mean()
    std = np.std(dist_to_n)
    flag = (dist_to_n > 0.25 * std)
    print(" - removing {0} outliers for plot".format(flag.sum()))
    return flag
```

In [8]:
```
# Plot Subplot Functions
def subp_2d(X):
    values_1 = X[0]
    title_1 = X[1]
    values_2 = X[2]
    title_2 = X[3]
    values_3 = X[4]
    title_3 = X[5]
    values_4 = X[6]
    title_4 = X[7]
    label = X[8]
    label2 = X[9] # removed flag

    fig, axs = plt.subplots(2, 2, figsize=(12,10))

    axs[0, 0].scatter(values_1[:, 0], values_1[:, 1], c=label, cmap=plt.cm.jet)
    axs[0, 0].title.set_text(title_1)


    axs[0, 1].scatter(values_2[:, 0], values_2[:, 1], c=label, cmap=plt.cm.jet)
    axs[0, 1].title.set_text(title_2)


    axs[1, 0].scatter(values_3[:, 0], values_3[:, 1], c=label, cmap=plt.cm.jet)
    axs[1, 0].title.set_text(title_3)


    axs[1, 1].scatter(values_4[:, 0], values_4[:, 1], c=label2, cmap=plt.cm.jet)
    axs[1, 1].title.set_text(title_4)

def subp_3d(X):
    values_1 = X[0]
    title_1 = X[1]
    values_2 = X[2]
    title_2 = X[3]
    values_3 = X[4]
    title_3 = X[5]
    values_4 = X[6]
    title_4 = X[7]
    label = X[8]
    label2 = X[9] # removed flag

    fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=("Isomap", "Umap", "Spectral Embedding", "LLE"),
    specs=[[{'type': 'scatter3d'}, {'type': 'scatter3d'}], # 'scatter3d'
           [{'type': 'scatter3d'}, {'type': 'scatter3d'}]])
    fig.add_trace(
        go.Scatter3d(x=values_1[:,0], y=values_1[:,1], z=values_1[:,2], mode = 'markers',marker = dict(size = 4, color =label)),
        row=1, col=1)

    fig.add_trace(
        go.Scatter3d(x=values_2[:,0], y=values_2[:,1], z=values_2[:,2], mode = 'markers',marker = dict(size = 4, color =label)),
        row=1, col=2)

    fig.add_trace(
        go.Scatter3d(x=values_3[:,0], y=values_3[:,1], z=values_3[:,2], mode = 'markers',marker = dict(size = 4, color =label)),
        row=2, col=1)

    fig.add_trace(
        go.Scatter3d(x=values_4[:,0], y=values_4[:,1], z=values_4[:,2], mode = 'markers',marker = dict(size = 4, color =label2)),
        row=2, col=2)

    fig.update_layout(height=800, width=800,
                  title_text="Subplots of Manifold Methods")

    fig.show()
```

1.1 Dataset size

- Defaults:
- n_components = 4
- n_neighbors = 6

In [9]:
```
# Change Dataset Size

spectra_50 = spectra[0: int(len(spectra) * 0.5)]
spectra_75 = spectra[0: int(len(spectra) * 0.75)]
spectra_100 = spectra

y_50 = y[0: int(len(spectra) * 0.5)]
y_75 = y[0: int(len(spectra) * 0.75)]
y_100 = y
```

50% Data Size

In [10]:
```
iso = Isomap(n_components=4, n_neighbors=6)
ISOMAP_PROJECTION = iso.fit_transform(spectra_50)
```

In [11]:
```
umap_obj = UMAP(
    n_components=4,
    metric="euclidean",
    n_neighbors=6,
    min_dist=0.1, #0.5?
    random_state=random_state
)


UMAP_PROJECTION = umap_obj.fit_transform(spectra_50)
```

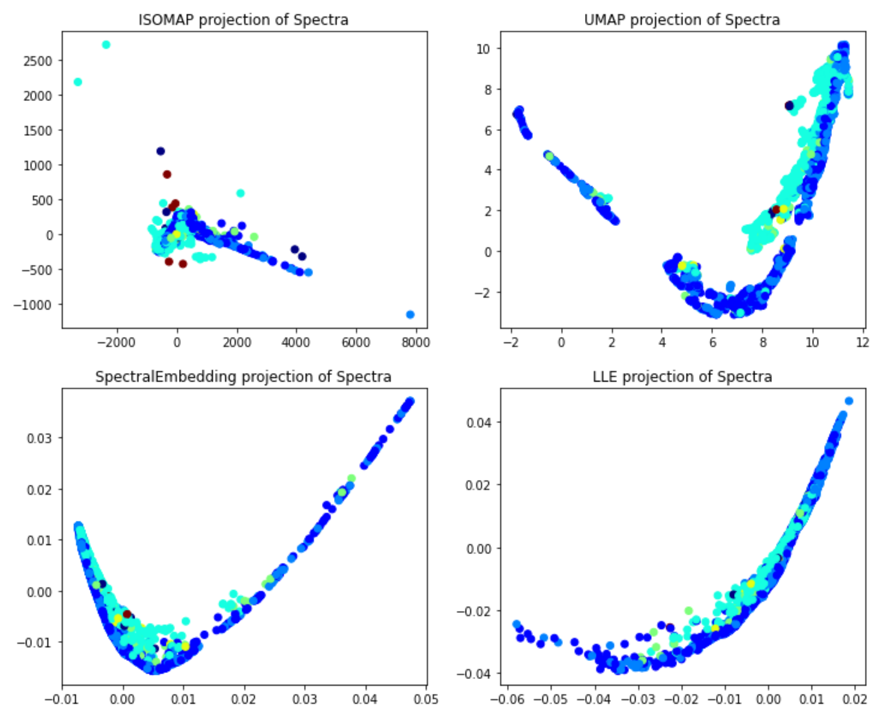
In [12]:
```
se = SpectralEmbedding(n_components=4, n_neighbors=6, random_state=random_state)

SE_PROJECTION = se.fit_transform(spectra_50)
```

```
In [13]: lle = LocallyLinearEmbedding(n_components=4, n_neighbors=6, method='modified', eigen_solver='dense', random_state=random_state)

         LLE_PROJECTION = lle.fit_transform(spectra_50)
         #find the mask to remove the outliers for the plot
         flag = mask_outliers(LLE_PROJECTION)
```

```
 - removing 84 outliers for plot
```

```
In [14]: combined = [ISOMAP_PROJECTION, 'ISOMAP projection of Spectra', UMAP_PROJECTION, 'UMAP projection of Spectra', SE_PROJECTION, 'SpectralEmbedding projection of Spectra',
                     LLE_PROJECTION[~flag], 'LLE projection of Spectra', y_50, y_50[~flag]]
```

```
In [15]: subp_2d(combined)
```



```
In [ ]: subp_3d(combined)
```

75% Data Size

```
In [17]: iso = Isomap(n_components=4, n_neighbors=6)
         ISOMAP_PROJECTION = iso.fit_transform(spectra_75)
```

```
In [18]: umap_obj = UMAP(
             n_components=4,
             metric="euclidean",
             n_neighbors=6,
             min_dist=0.1, #0.5?
             random_state=random_state
         )


         UMAP_PROJECTION = umap_obj.fit_transform(spectra_75)
```

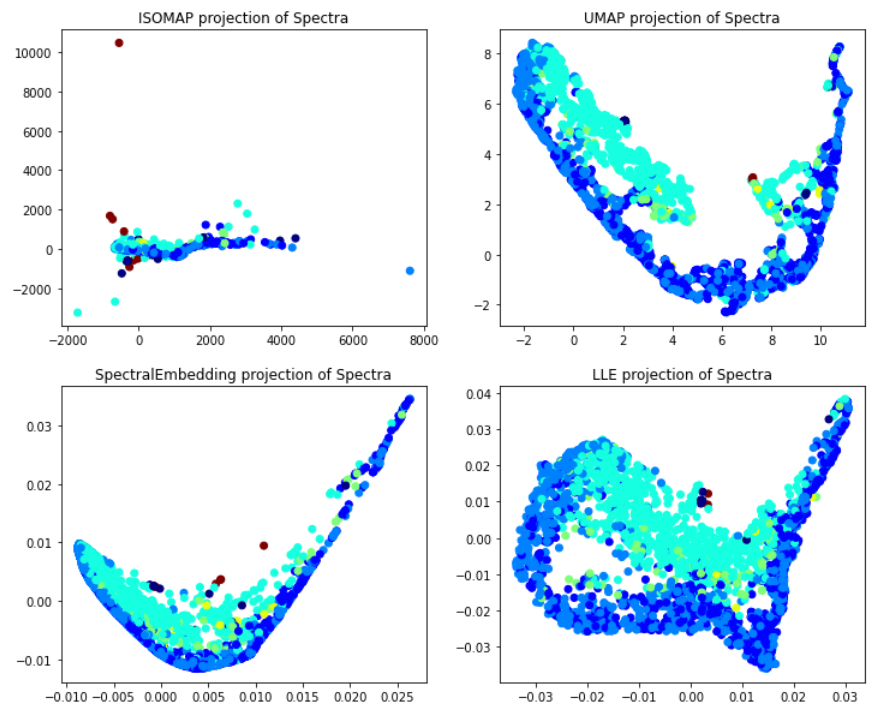
```
In [19]: se = SpectralEmbedding(n_components=4, n_neighbors=6, random_state=random_state)

         SE_PROJECTION = se.fit_transform(spectra_75)
```

```
In [20]: lle = LocallyLinearEmbedding(n_components=4, n_neighbors=6, method='modified', eigen_solver='dense', random_state=random_state)

         LLE_PROJECTION = lle.fit_transform(spectra_75)
         #find the mask to remove the outliers for the plot
         flag = mask_outliers(LLE_PROJECTION)
```

```
 - removing 109 outliers for plot
```

```
In [21]: combined = [ISOMAP_PROJECTION, 'ISOMAP projection of Spectra', UMAP_PROJECTION, 'UMAP projection of Spectra',
                     SE_PROJECTION, 'SpectralEmbedding projection of Spectra',LLE_PROJECTION[~flag], 'LLE projection of Spectra', y_75, y_75[~flag]]
```

```
In [22]: subp_2d(combined)
```



```
In [ ]: subp_3d(combined)
```

100% Data Size

```
In [24]: iso = Isomap(n_components=4, n_neighbors=6)
         ISOMAP_PROJECTION = iso.fit_transform(spectra_100)
```

```
In [25]: umap_obj = UMAP(
             n_components=4,
             metric="euclidean",
             n_neighbors=6,
             min_dist=0.1, #0.5?
             random_state=random_state
         )


         UMAP_PROJECTION = umap_obj.fit_transform(spectra_100)
```

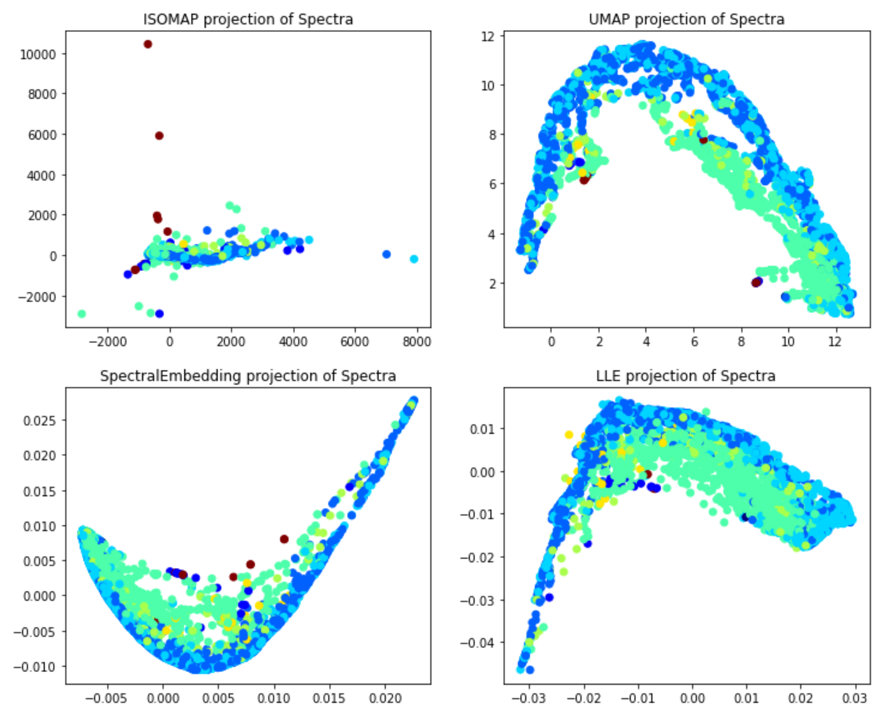
In [26]:
```
se = SpectralEmbedding(n_components=4, n_neighbors=6, random_state=random_state)

SE_PROJECTION = se.fit_transform(spectra_100)
```

In [27]:
```
lle = LocallyLinearEmbedding(n_components=4, n_neighbors=6, method='modified', eigen_solver='dense', random_state=random_state)

LLE_PROJECTION = lle.fit_transform(spectra_100)
#find the mask to remove the outliers for the plot
flag = mask_outliers(LLE_PROJECTION)
```

```
 - removing 48 outliers for plot
```

In [28]:
```
combined = [ISOMAP_PROJECTION, 'ISOMAP projection of Spectra', UMAP_PROJECTION, 'UMAP projection of Spectra',
            SE_PROJECTION, 'SpectralEmbedding projection of Spectra',LLE_PROJECTION[~flag], 'LLE projection of Spectra', y_100, y_100[~flag]]
```

In [29]:
```
subp_2d(combined)
```



In [ ]:
```
subp_3d(combined)
```

1.1.1 How stable is the projection between different subsets of the data?

- The projections are fairly stable as, their shape are maintained between the different subsets of the data, with effections of missing points within the other subsets of the projections, Essentially the overall view of the projection can be established with a smaller amount of the overall data available. The projection just becomes

1.1.2 Which of these manifold methods appears to give the most stable results?

- It appears Spectral Embedding provides the most stable results, The shape of the projection is properly maintained between subsets.

1.2 Number of neighbours

Baseline:

- Dataset Size - 100%
- number of components - 4

Number of neighbors = 10

In [31]:
```
iso = Isomap(n_components=4, n_neighbors=10)
ISOMAP_PROJECTION = iso.fit_transform(spectra_100)
```

In [32]:
```
umap_obj = UMAP(
    n_components=4,
    metric="euclidean",
    n_neighbors=10,
    min_dist=0.1, #0.5?
    random_state=random_state
)


UMAP_PROJECTION = umap_obj.fit_transform(spectra_100)
```

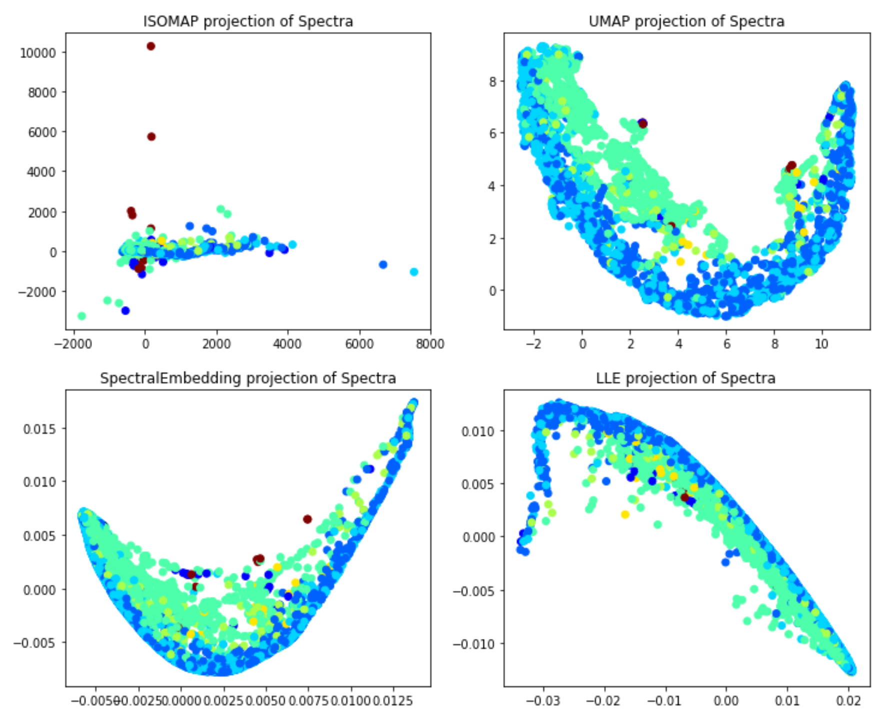
In [33]:
```
se = SpectralEmbedding(n_components=4, n_neighbors=10, random_state=random_state)

SE_PROJECTION = se.fit_transform(spectra_100)
```

In [34]:
```
lle = LocallyLinearEmbedding(n_components=4, n_neighbors=10, method='modified', eigen_solver='dense', random_state=random_state)

LLE_PROJECTION = lle.fit_transform(spectra_100)
#find the mask to remove the outliers for the plot
flag = mask_outliers(LLE_PROJECTION)
```

```
 - removing 52 outliers for plot
```

In [35]:
```
combined = [ISOMAP_PROJECTION, 'ISOMAP projection of Spectra', UMAP_PROJECTION, 'UMAP projection of Spectra',
            SE_PROJECTION, 'SpectralEmbedding projection of Spectra',LLE_PROJECTION[~flag], 'LLE projection of Spectra', y_100, y_100[~flag]]
```

In [36]:
```
subp_2d(combined)
```



Number of neighbors = 20

In [38]:
```
iso = Isomap(n_components=4, n_neighbors=20)
ISOMAP_PROJECTION = iso.fit_transform(spectra_100)
```

In [39]:
```
umap_obj = UMAP(
    n_components=4,
    metric="euclidean",
    n_neighbors=20,
    min_dist=0.1, #0.5?
    random_state=random_state
)


UMAP_PROJECTION = umap_obj.fit_transform(spectra_100)
```

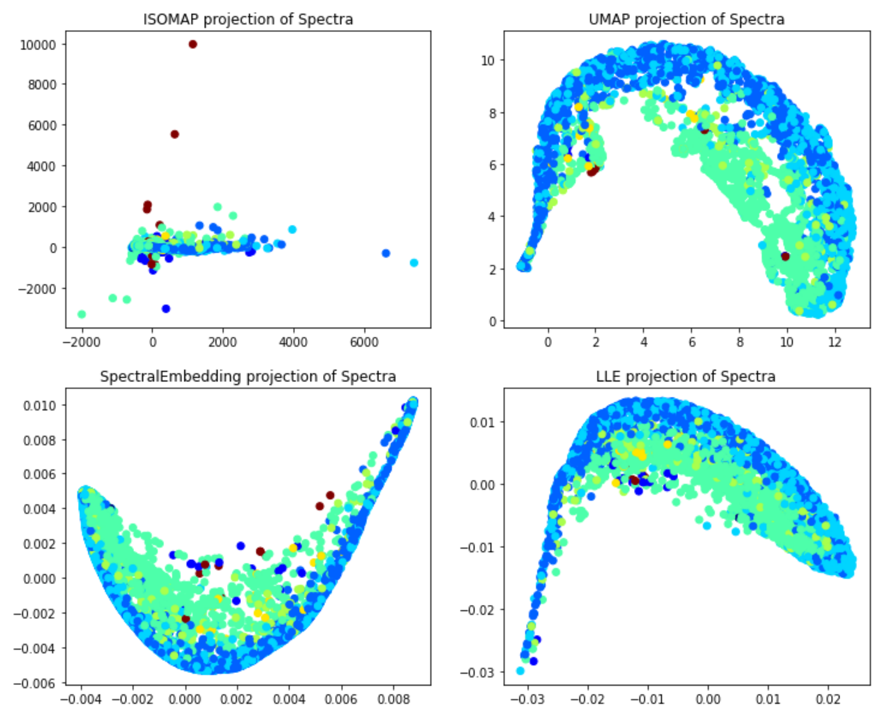
In [40]:
```
se = SpectralEmbedding(n_components=4, n_neighbors=20, random_state=random_state)

SE_PROJECTION = se.fit_transform(spectra_100)
```

In [41]:
```
lle = LocallyLinearEmbedding(n_components=4, n_neighbors=20, method='modified', eigen_solver='dense', random_state=random_state)

LLE_PROJECTION = lle.fit_transform(spectra_100)
#find the mask to remove the outliers for the plot
flag = mask_outliers(LLE_PROJECTION)
```

```
 - removing 42 outliers for plot
```

In [42]:
```
combined = [ISOMAP_PROJECTION, 'ISOMAP projection of Spectra', UMAP_PROJECTION, 'UMAP projection of Spectra',
            SE_PROJECTION, 'SpectralEmbedding projection of Spectra',LLE_PROJECTION[~flag], 'LLE projection of Spectra', y_100, y_100[~flag]]
```

In [43]:
```
subp_2d(combined)
```



In [ ]:
```
subp_3d(combined)
```

Number of neighbors = 50

In [45]:
```python
iso = Isomap(n_components=4, n_neighbors=50)
ISOMAP_PROJECTION = iso.fit_transform(spectra_100)
```

In [46]:
```python
umap_obj = UMAP(
    n_components=4,
    metric="euclidean",
    n_neighbors=50,
    min_dist=0.1, #0.5?, 0.01
    random_state=random_state
)


UMAP_PROJECTION = umap_obj.fit_transform(spectra_100)
```

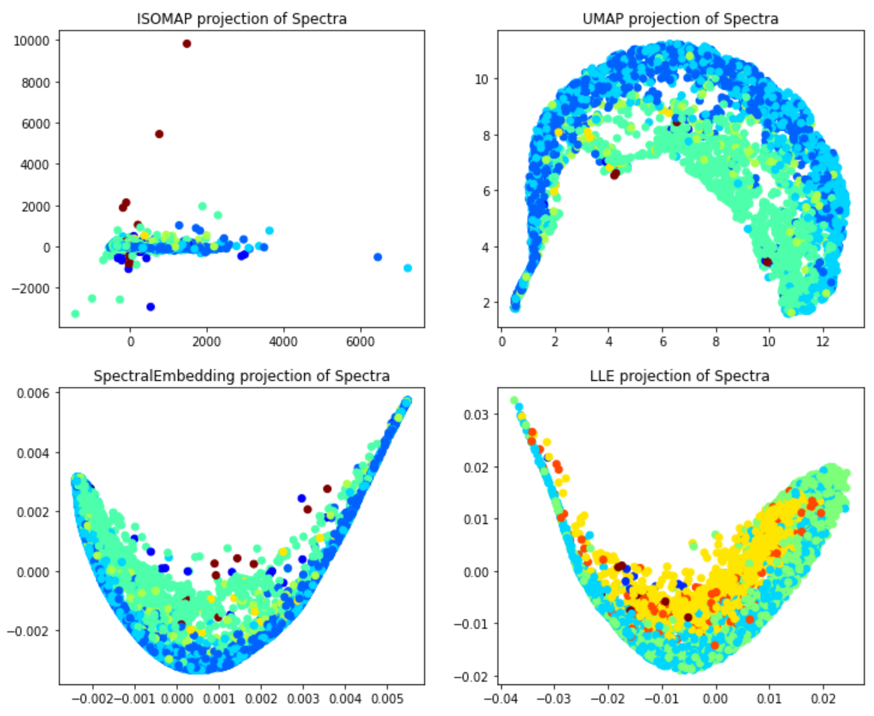
In [47]:
```python
se = SpectralEmbedding(n_components=4, n_neighbors=50, random_state=random_state)

SE_PROJECTION = se.fit_transform(spectra_100)
```

In [48]:
```python
lle = LocallyLinearEmbedding(n_components=4, n_neighbors=50, method='modified', eigen_solver='dense', random_state=random_state)

LLE_PROJECTION = lle.fit_transform(spectra_100)
#find the mask to remove the outliers for the plot
flag = mask_outliers(LLE_PROJECTION)
```

```
 - removing 67 outliers for plot
```

In [49]:
```python
combined = [ISOMAP_PROJECTION, 'ISOMAP projection of Spectra', UMAP_PROJECTION, 'UMAP projection of Spectra',
            SE_PROJECTION, 'SpectralEmbedding projection of Spectra',LLE_PROJECTION[~flag], 'LLE projection of Spectra', y_100, y_100[~flag]]
```

In [50]:
```python
subp_2d(combined)
```



In [ ]:
```python
subp_3d(combined)
```

1.2.1 How does the number of neighbors change the projection?

- The higher the number of neighbors the greater the number of point detected as part of the cluster projection. Making the projected appear fuller.

1.2.2 Which of the manifold methods appears to have the most stable results as the number of neighbors is changed?

Overall, all the manifold maintained their projection structure, with the exception of some additional points. The most stable out of all the manifolds appears to be SE and Isomap in this case

1.2.3 Plots to support your conclusions.

- Plots above supports the coclusions

1.3 Number of components

Baseline:

- Dataset Size - 100%
- number of neighbors - 6
- for LLE n_neighbors > n_components

Number of components = 10

In [52]:
```python
iso = Isomap(n_components=10, n_neighbors=6)
ISOMAP_PROJECTION = iso.fit_transform(spectra_100)
```

In [53]:
```python
umap_obj = UMAP(
    n_components=10,
    metric="euclidean",
    n_neighbors=6,
    min_dist=0.01, #0.5?
    random_state=random_state
)


UMAP_PROJECTION = umap_obj.fit_transform(spectra_100)
```

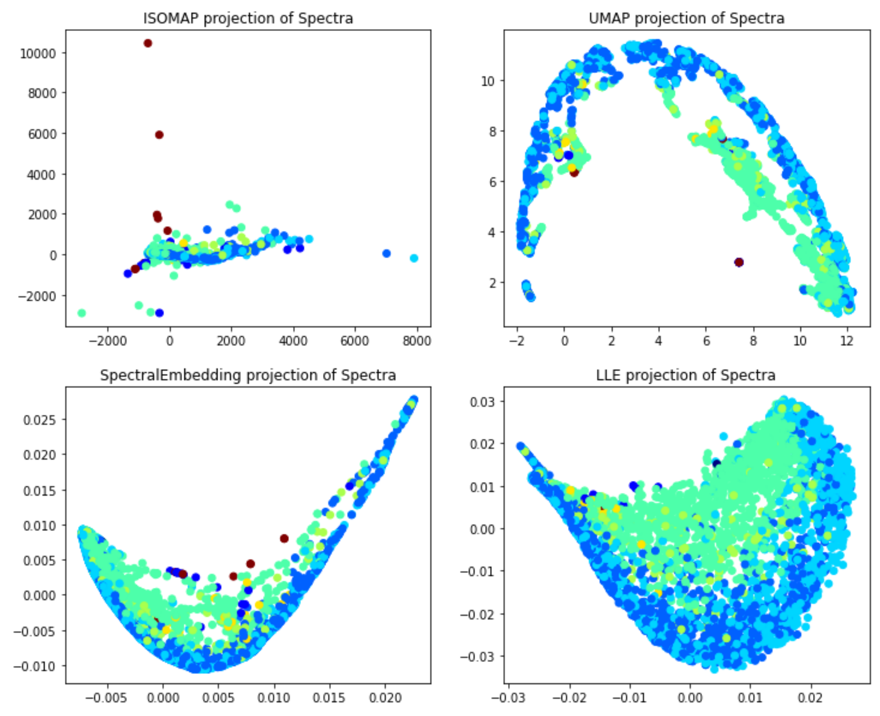
In [54]:
```python
se = SpectralEmbedding(n_components=10, n_neighbors=6, random_state=random_state)

SE_PROJECTION = se.fit_transform(spectra_100)
```

In [55]:
```python
lle = LocallyLinearEmbedding(n_components=10, n_neighbors=20, method='modified', eigen_solver='dense', random_state=random_state)

LLE_PROJECTION = lle.fit_transform(spectra_100)
#find the mask to remove the outliers for the plot
flag = mask_outliers(LLE_PROJECTION)
```

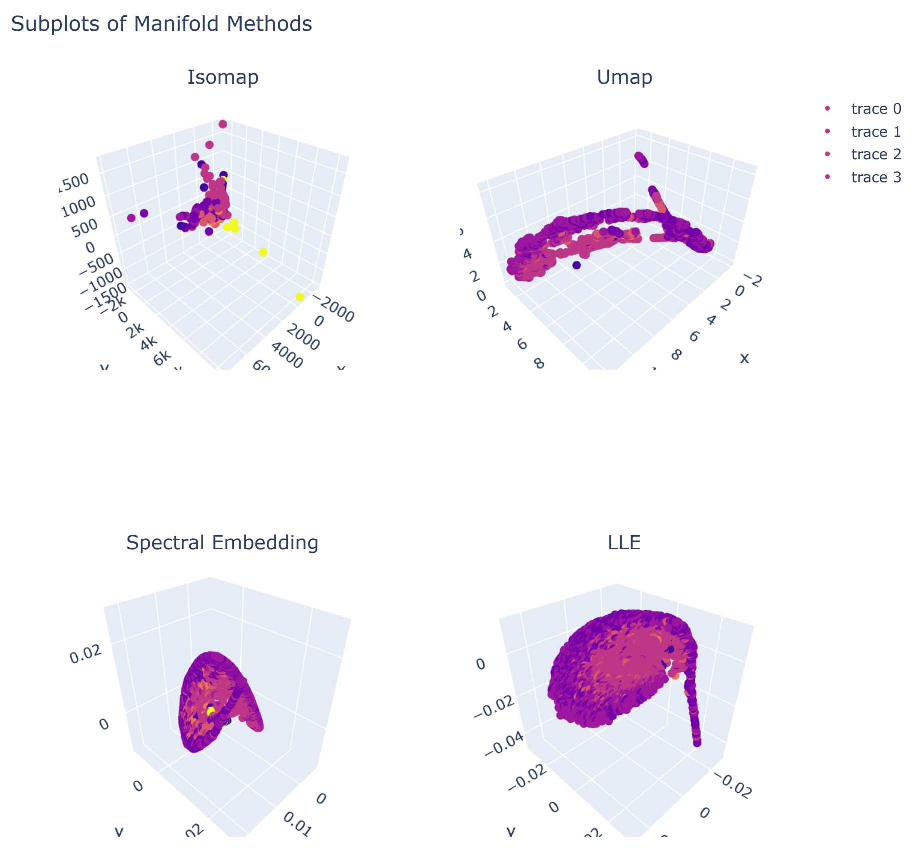
```
 - removing 122 outliers for plot
```

In [56]:
```python
combined = [ISOMAP_PROJECTION, 'ISOMAP projection of Spectra', UMAP_PROJECTION, 'UMAP projection of Spectra',
            SE_PROJECTION, 'SpectralEmbedding projection of Spectra',LLE_PROJECTION[~flag], 'LLE projection of Spectra', y_100, y_100[~flag]]
```

In [57]:
```
subp_2d(combined)
```



In [58]:
```
subp_3d(combined)
```



Number of components = 20

In [59]:
```
iso = Isomap(n_components=20, n_neighbors=6)
ISOMAP_PROJECTION = iso.fit_transform(spectra_100)
```

In [60]:
```
umap_obj = UMAP(
    n_components=20,
    metric="euclidean",
    n_neighbors=6,
    min_dist=0.01, #0.5?
    random_state=random_state
)


UMAP_PROJECTION = umap_obj.fit_transform(spectra_100)
```

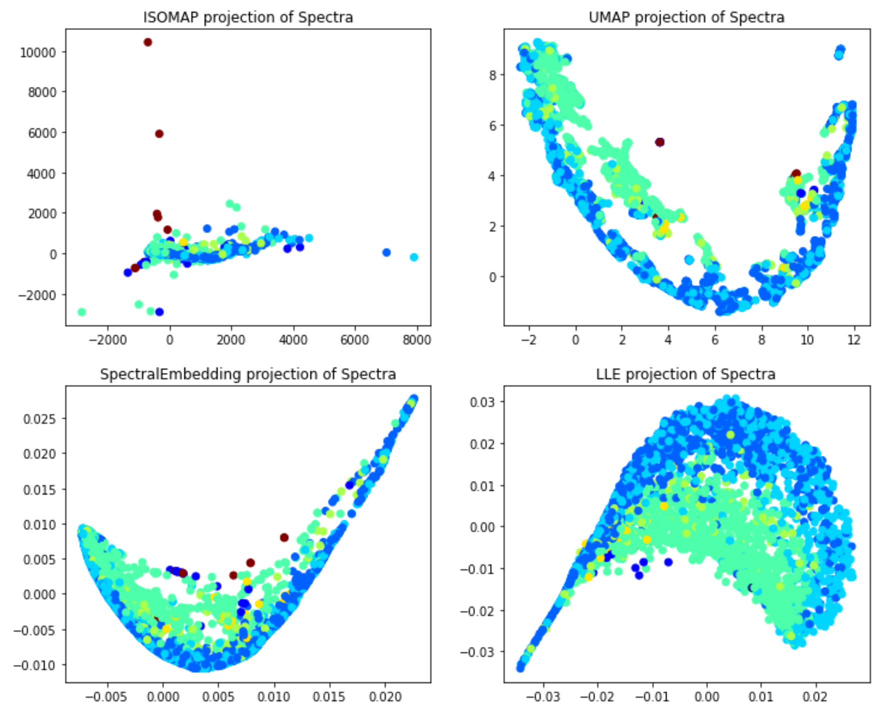
In [61]:
```
se = SpectralEmbedding(n_components=20, n_neighbors=6, random_state=random_state)

SE_PROJECTION = se.fit_transform(spectra_100)
```

In [62]:
```
lle = LocallyLinearEmbedding(n_components=20, n_neighbors=50, method='modified', eigen_solver='dense', random_state=random_state)

LLE_PROJECTION = lle.fit_transform(spectra_100)
#find the mask to remove the outliers for the plot
flag = mask_outliers(LLE_PROJECTION)
```

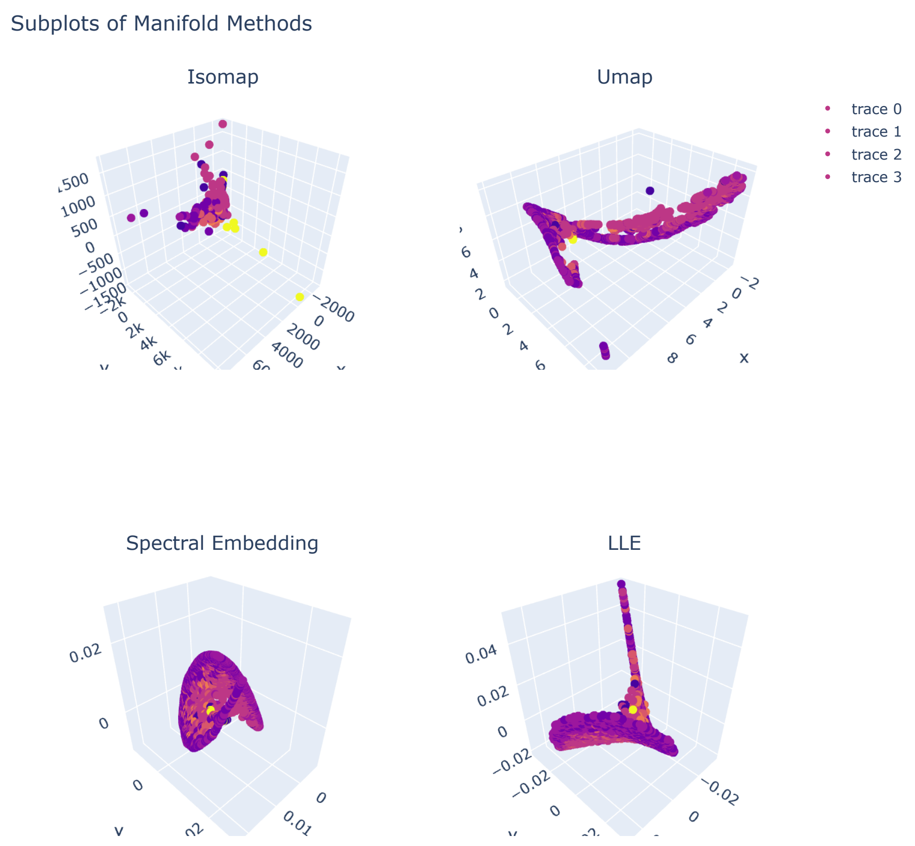
```
 - removing 637 outliers for plot
```

In [63]:
```
combined = [ISOMAP_PROJECTION, 'ISOMAP projection of Spectra', UMAP_PROJECTION, 'UMAP projection of Spectra',
            SE_PROJECTION, 'SpectralEmbedding projection of Spectra',LLE_PROJECTION[~flag], 'LLE projection of Spectra', y_100, y_100[~flag]]
```

```
In [64]: subp_2d(combined)
```



```
In [65]: subp_3d(combined)
```



Number of components = 50

```
In [66]: iso = Isomap(n_components=50, n_neighbors=6)
         ISOMAP_PROJECTION = iso.fit_transform(spectra_100)
```

```
In [67]: umap_obj = UMAP(
             n_components=50,
             metric="euclidean",
             n_neighbors=6,
             min_dist=0.01, #0.5?
             random_state=random_state
         )


         UMAP_PROJECTION = umap_obj.fit_transform(spectra_100)
```

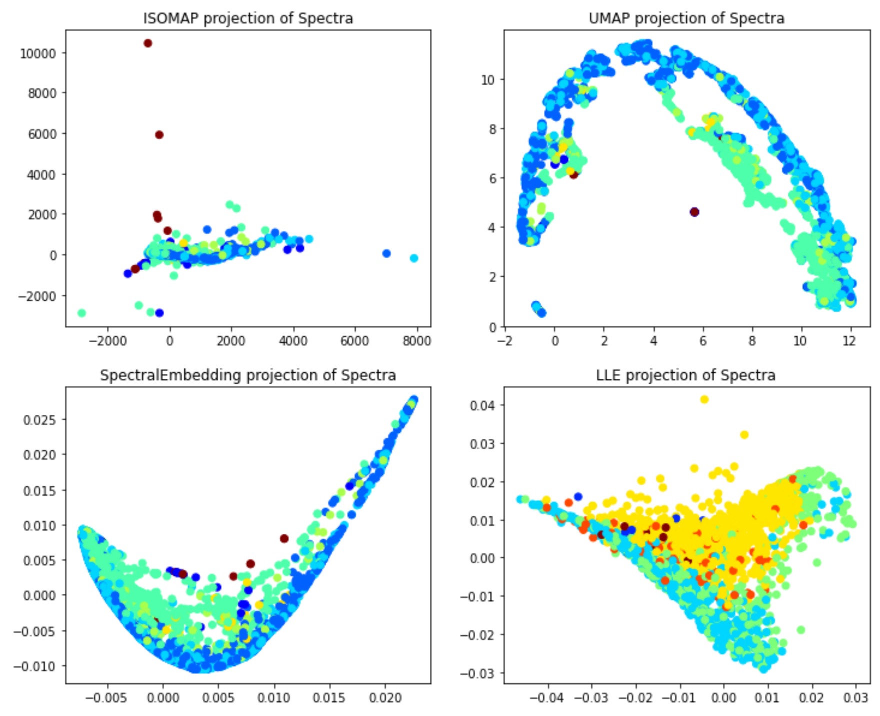
```
In [68]: se = SpectralEmbedding(n_components=50, n_neighbors=6, random_state=random_state)

         SE_PROJECTION = se.fit_transform(spectra_100)
```

```
In [69]: lle = LocallyLinearEmbedding(n_components=50, n_neighbors=60, method='modified', eigen_solver='dense', random_state=random_state)

         LLE_PROJECTION = lle.fit_transform(spectra_100)
         #find the mask to remove the outliers for the plot
         flag = mask_outliers(LLE_PROJECTION)
```

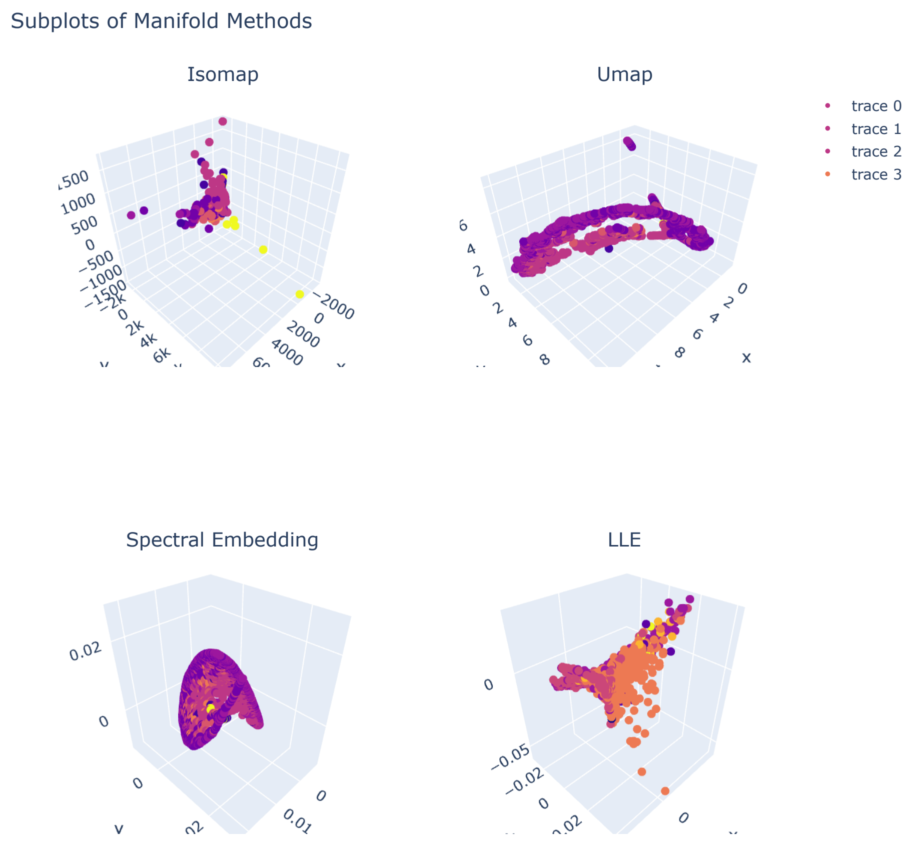
```
 - removing 1254 outliers for plot
```

```
In [70]: combined = [ISOMAP_PROJECTION, 'ISOMAP projection of Spectra', UMAP_PROJECTION, 'UMAP projection of Spectra',
                     SE_PROJECTION, 'SpectralEmbedding projection of Spectra',LLE_PROJECTION[~flag], 'LLE projection of Spectra', y_100, y_100[~flag]]
```

```
In [71]: subp_2d(combined)
```



```
In [72]: subp_3d(combined)
```



1.3.1 How does the number of components change the projection?

- The number of components adds additional features to the overall projection, but based on the projections shown, does not have significant differences, only when the components are specified tobe quite high

1.3.2 Which of the manifold methods appears to have the most stable results as the number of components change?

- ISOMAP and SpectralEmbedding once again provide the most stable results

1.3.3 Plots to support your conclusions.

- Plots can be seen above.

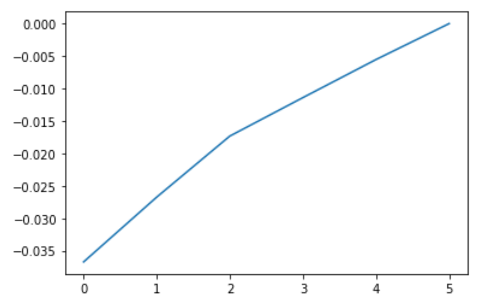
1.3.4

- Using get_eigenvalues function

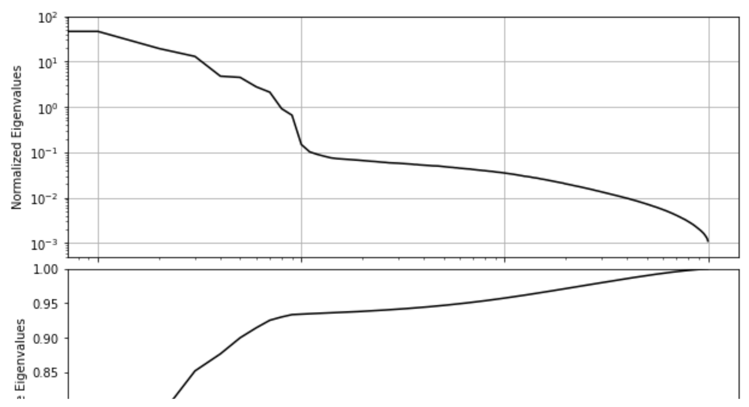
In [73]:
```
def get_eigenvalues(X, n_neighbours, n_clusters):
    """ Helper function that will construct a nearest neighbours
    similarity graph from X, then compute a normalised Laplacian for this graph,
    and then calculate the smallest n_clusters eigenvalues.

    This is useful for identifying the number of clusters expected
    in the data when using SpectralEmbedding
    """
    K = kneighbors_graph(X, n_neighbors=n_neighbours, include_self=True)

    K = 0.5 * (K + K.T)


    # diagonal matrix
    L, D = csgraph_laplacian(
            K, normed=True, return_diag=True
        )

    L = L.tocoo()
    diag_idx = L.row == L.col
    L.data[diag_idx] = 1
    # If the matrix has a small number of diagonals (as in the
    # case of structured matrices coming from images), the
    # dia format might be best suited for matvec products:
    n_diags = np.unique(L.row - L.col).size
    if n_diags <= 7:
        # 3 or less outer diagonals on each side
        L = L.todia()
    else:
        # csr has the fastest matvec and is thus best suited to
        # arpack
        L = L.tocsr()

    L *= -1
    vals, vecs = eigsh(
        L, k=n_clusters, sigma=1.0, which="LM", tol=1e-5
    )

    # sort these based on the eigenvalues
    vecs = vecs[:,np.argsort(vals)]
    vals = vals[np.argsort(vals)]
    plt.plot(vals)

    return vecs, vals
```

In [74]:
```
get_eigenvalues(spectra_100, 50, 6)
```

Out[74]:
```
(array([[ 4.94773630e-05, -1.91431071e-03, -1.64734736e-03,
          2.11238173e-03, -8.61258949e-03, -1.48976097e-02],
        [ 6.47722857e-03, -2.59233340e-02, -2.21387349e-02,
         -4.98826418e-03, -1.31767728e-02, -1.76415558e-02],
        [ 1.44125381e-02, -6.73549126e-03,  3.89903772e-03,
          1.20157840e-02, -1.37507837e-02, -1.36464094e-02],
        ...,
        [-8.47109996e-03,  1.35726789e-02,  1.66151373e-02,
          1.99589828e-02, -1.71516688e-02, -1.76415558e-02],
        [ 2.59123765e-02,  1.19303013e-02,  1.81543885e-02,
         -2.03690229e-02,  1.22492343e-02, -1.72762666e-02],
        [-1.64925677e-02,  6.81090313e-03, -1.59026679e-02,
         -1.65771562e-02, -4.32506279e-03, -1.69030851e-02]]),
 array([-3.67249306e-02, -2.67401090e-02, -1.73332331e-02, -1.14030282e-02,
        -5.53687403e-03, -1.33226763e-15]))
```



In [75]:
```
fig = plt.figure(figsize=(10, 7.5))
fig.subplots_adjust(hspace=0.05, bottom=0.12)

ax = fig.add_subplot(211, xscale='log', yscale='log')
ax.grid()
ax.plot(evals, c='k')
ax.set_ylabel('Normalized Eigenvalues')
ax.xaxis.set_major_formatter(plt.NullFormatter())
ax.set_ylim(5E-4, 100)

ax = fig.add_subplot(212, xscale='log')
ax.grid

ax.semilogx(evals_cs, color='k')
ax.set_xlabel('Eigenvalue Number')
ax.set_ylabel('Cumulative Eigenvalues')
ax.set_ylim(0.65, 1.00)

plt.show()
```



2 Gene Expression Data

In [76]:
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib inline
from scipy.stats import norm

# Install ipympl and uncomment this for interactive plots
#%matplotlib widget
import minisom
from umap import UMAP

from sklearn.cluster import KMeans, SpectralClustering
from sklearn.manifold import LocallyLinearEmbedding
from sklearn import manifold, neighbors
from sklearn.metrics.cluster import normalized_mutual_info_score, adjusted_mutual_info_score, adjusted_rand_score
from sklearn.metrics import classification_report
import pandas as pd
import plotly.express as px

random_state = 25
```

In [77]:
```
def plot_clusters(data, labels):
    palette = sns.color_palette('deep', np.unique(labels).max() + 1)
    colors = [palette[x] if x >= 0 else (0.0, 0.0, 0.0) for x in labels]
    plt.scatter(data.T[0], data.T[1], c=y) # colors,) # **plot_kwds)
    frame = plt.gca()
    frame.axes.get_xaxis().set_visible(False)
    frame.axes.get_yaxis().set_visible(False)
```

In [78]:
```
df = pd.read_csv('14cancer.xtrain', delim_whitespace=True, names=[f'sample {i}' for i in range(1, 145)])
dftest = pd.read_csv('14cancer.xtest', delim_whitespace=True, names=[f'sample {i}' for i in range(1, 55)])
df = df.T.reset_index().drop(columns='index').copy()
dftest = dftest.T.reset_index().drop(columns='index').copy()

labels = [int(x) for x in open('14cancer.ytrain').readline().split()]
df_labels = pd.DataFrame({'label': labels})
labelstest = [int(x) for x in open('14cancer.ytest').readline().split()]
df_labels_test = pd.DataFrame({'label': labelstest})

label_names = {
    1: 'breast',
    2:  'prostate',
    3:  'lung',
    4:  'collerectal',
    5:  'lymphoma',
    6:  'bladder',
    7:  'melanoma',
    8:  'uterus',
    9: 'leukemia',
    10: 'renal',
    11: 'pancreas',
    12: 'ovary',
    13: 'meso',
    14: 'cns'
}
```

In [79]:
```
# Normalizing the data, Attempt with Normalized data.
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
df_normalized = sc.fit_transform(df)
df_normalized = pd.DataFrame(df_normalized)

dftest_normalized = sc.fit_transform(dftest)
dftest_normalized = pd.DataFrame(dftest_normalized)
```

In [80]:
```
# Normalized data
df = df_normalized
dftest = dftest_normalized
```

In [81]:
```
def plot_2d(X, labels, title):
    plt.figure()
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap=plt.cm.jet)
    plt.xlabel('coefficient 1')
    plt.ylabel('coefficient 2')
    plt.title(title)

def plot_3d(X, labels, title):
    fig = px.scatter_3d(
    X, x=0, y=1, z=2,
    color=labels, labels=labels,title=title)
    fig.update_traces(marker_size=8)
    fig.show()
```

In [82]:
```
n_features = df.shape[1] #IMPLEMENT_ME, try normalized dfs
som_shape = (8,8) #(IMPLEMENT_ME, IMPLEMENT_ME)
train = np.array(df)
test = np.array(dftest)

som = minisom.MiniSom(som_shape[0], som_shape[1], n_features, sigma=1, learning_rate=0.5) # sigma=IMPLEMENT_ME, learning_rate=IMPLEMENT_ME,)

# this can make results more stable, but it also takes a long time to process
# som.pca_weights_init(df)

iterations = 2500
som.train(train, iterations)    #(IMPLEMENT_ME, IMPLEMENT_ME)
```

In [83]:
```
# Weights are:
wts = som.get_weights()
# Shape of the weight are:
wts.shape
```

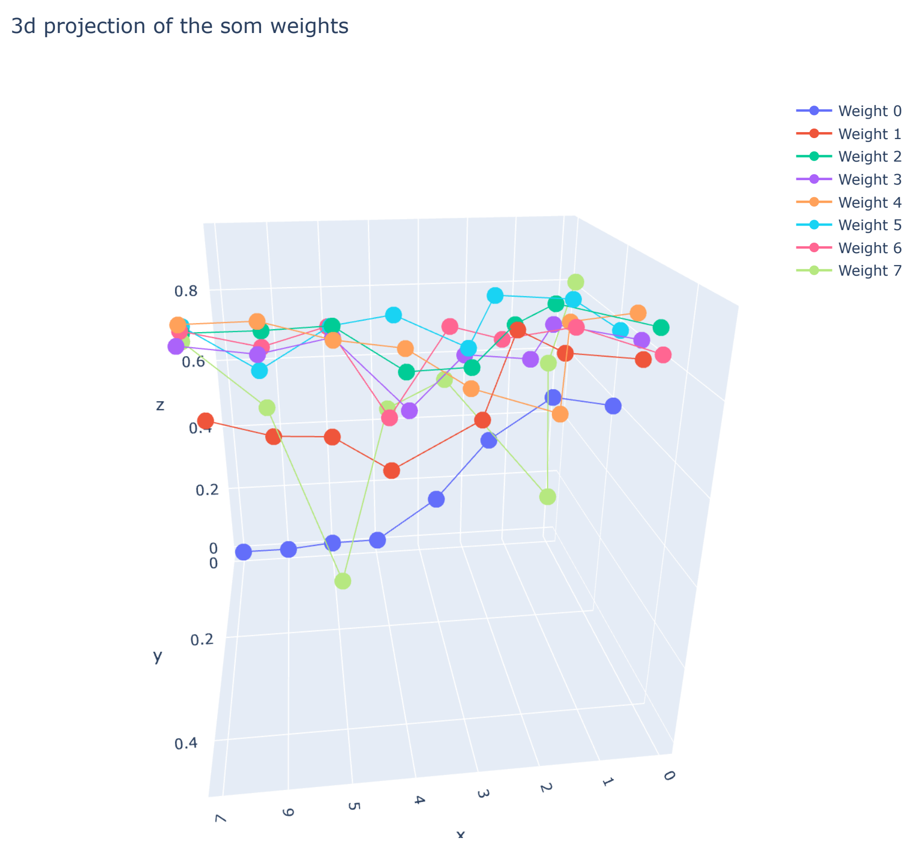
Out[83]:
```
(8, 8, 16063)
```

In [84]:
```
# U-Matrix:
dst_map = som.distance_map()
# Shape of distance map are:
dst_map.shape
```

Out[84]:
```
(8, 8)
```

2.1.1 Plot the weights of the map (som.get_weights())

In [85]:
```python
# Plot a 3d projection of the som weights
# (the map is som_shape[0] X som_shape[1] and each point on the map has an associated weight)

# IMPLEMENT_ME

import numpy as np
import plotly.graph_objs as go

features = [wts[0,:], wts[1,:], wts[2,:], wts[3,:], wts[4,:], wts[5,:], wts[6,:], wts[7,:]] #, wts[8,:], wts[9,:]]

fig = go.Figure()
for i, feat in enumerate(features):
    feat = np.array(feat)
    fig.add_trace(
        go.Scatter3d(
            x=np.arange(len(feat)),
            y=feat[:,0],
            z=feat[:,1],
            name = f"Weight {i}"
        )
    )

fig.update_layout(height=800, width=800,
                  title_text="3d projection of the som weights")
fig.show()
```

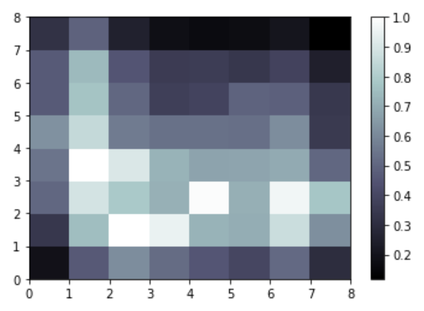


In [86]:
```python
def classify(som, x, y):
    """Classifies each sample in data in one of the classes definited
    using the method labels_map.
    Returns a list of the same length of data where the i-th element
    is the class assigned to data[i].
    """
    winmap = som.labels_map(x, y)
    default_class = np.sum(list(winmap.values())).most_common()[0][0]
    result = []
    for d in x:
        win_position = som.winner(d)
        if win_position in winmap:
            result.append(winmap[win_position].most_common()[0][0])
        else:
            result.append(default_class)
    return result


pred = classify(som, test, df_labels_test.label)


print(adjusted_mutual_info_score(df_labels_test.label, pred))
print(classification_report(df_labels_test.label, pred))
```

```
0.43935004353332674
              precision    recall  f1-score   support

           1       0.60      0.75      0.67         4
           2       1.00      0.50      0.67         6
           3       1.00      0.75      0.86         4
           4       0.27      0.75      0.40         4
           5       0.67      1.00      0.80         6
           6       0.60      1.00      0.75         3
           7       1.00      0.50      0.67         2
           8       0.00      0.00      0.00         2
           9       1.00      0.83      0.91         6
          10       0.40      0.67      0.50         3
          11       1.00      0.33      0.50         3
          12       1.00      0.25      0.40         4
          13       0.50      0.33      0.40         3
          14       1.00      0.75      0.86         4

    accuracy                           0.65        54
   macro avg       0.72      0.60      0.60        54
weighted avg       0.76      0.65      0.64        54
```

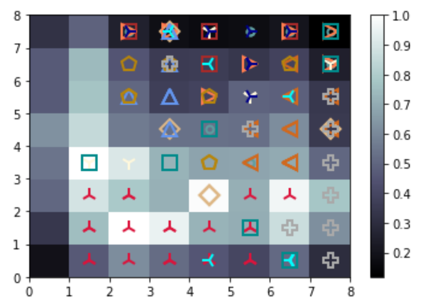
2.1.2 Plot the U-Matrix (som.distance_map())

In [87]:
```python
# Plot the distance_map from the som to get the U-Matrix (NxN).  Choose a suitable cmap when plotting.
# IMPLEMENT_ME

from pylab import plot, axis, show, pcolor, colorbar, bone
bone()
pcolor(som.distance_map().T)       # Distance map as background
colorbar()
show()
```



2.2.3 Plot the clusters (som.winner())

In [88]:
```
# Plot the winner (som.winner(x)) for every datapoint x on its node on the U-Matrix plot above.  Choose a different colour for each class.
# this should look something like the seeds_clusters.png plot
# IMPLEMENT_ME

bone()
pcolor(som.distance_map().T)
colorbar() #gives legend

# different colors and markers for each label
markers = ['o', 's', 'D', '.', ',', '<', '>', '^', '1', '2', '3', '4', 's', 'p', 'P', '*']
colors = ['blueviolet', 'brown', 'burlywood', 'cadetblue', 'chartreuse', 'chocolate',
          'coral', 'cornflowerblue', 'cornsilk', 'crimson', 'cyan', 'darkblue', 'darkcyan',
          'darkgoldenrod', 'darkgray', 'darkgreen', 'darkgrey']

for cnt,xx in enumerate(train):
    w = som.winner(xx)
    plot(w[0]+.5,w[1]+.5,markers[labels[cnt]],markerfacecolor='None',
         markeredgecolor=colors[labels[cnt]],markersize=12,markeredgewidth=2)
axis([0,som.get_weights().shape[0],0,som.get_weights().shape[1]])
show()
```



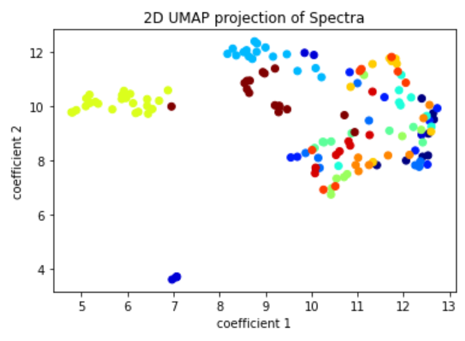
2.1.4 From the above results, can you identify any clusters? Which cancer types cluster together?

- There are a number of clusters that can be identified from the results above, The red markered cancer types cluster together easily as well as the white cross markered cancer types. Other clusters include the blue marker as well as the orange triangles.

2.2

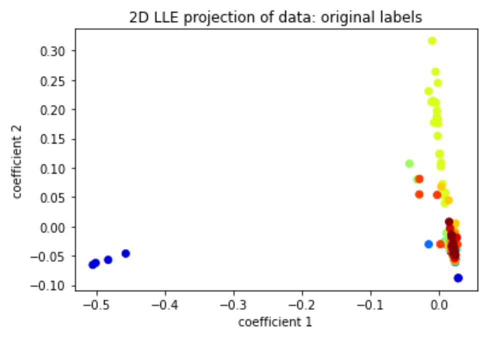
2.2.1 Perform UMAP and LLE on the same data and plot the leading two components of their embeddings. Comment on the outputs.

In [89]:
```
umap_obj = UMAP(
    n_components=2,
    metric="euclidean",
    n_neighbors=6,
    min_dist=0.1,
    random_state=random_state)

UMAP_PROJECTION = umap_obj.fit_transform(df) #IMPLEMENT_ME

plot_2d(UMAP_PROJECTION, df_labels.label, '2D UMAP projection of Spectra')
```



Overall the clusters can be fairly distinguished with the exception of some points that clearly does not belong to specific clusters. UMAP does a great job with regards to clustering the data using 2 components.

In [90]:
```
lle = LocallyLinearEmbedding(n_components=2, n_neighbors=6, method='modified', eigen_solver='dense', random_state=random_state)

LLE_PROJECTION = lle.fit_transform(df) #IMPLEMENT_ME

plot_2d(LLE_PROJECTION, df_labels.label, '2D LLE projection of data: original labels')
```



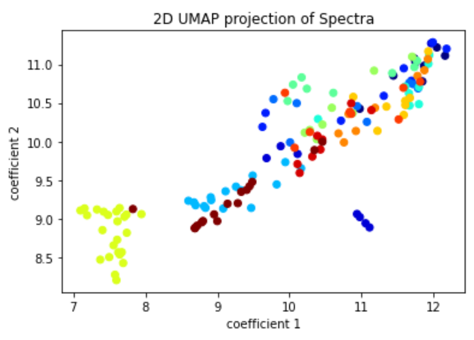
Compared to UMAP which uses 2 components as well, the LLE cluster is not as defined as performs poorly.

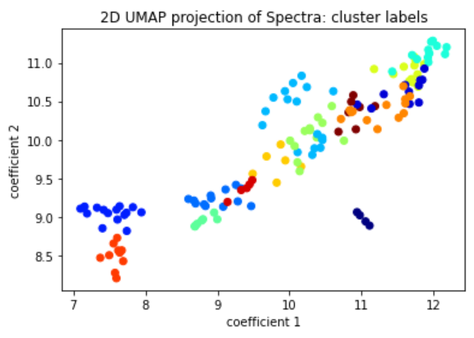
2.2.2 Perform k-means (with 14 clusters) on the output embeddings from UMAP and LLE. Plot the clustersc(with their predicted labels), and calculate the adjusted mutual information score.

UMAP k-means

In [91]:
```
umap_obj = UMAP(
    n_components=4,
    metric="euclidean",
    n_neighbors=6,
    min_dist=0.1,
    random_state=random_state)

UMAP_PROJECTION = umap_obj.fit_transform(df) #IMPLEMENT_ME

plot_2d(UMAP_PROJECTION, df_labels.label, '2D UMAP projection of Spectra')

km = KMeans(init='k-means++', n_clusters=14)
km.fit(UMAP_PROJECTION)

plot_2d(UMAP_PROJECTION, km.labels_, '2D UMAP projection of Spectra: cluster labels')

print(adjusted_mutual_info_score(df_labels.label, km.labels_))
```

```
0.4619948576016986
```

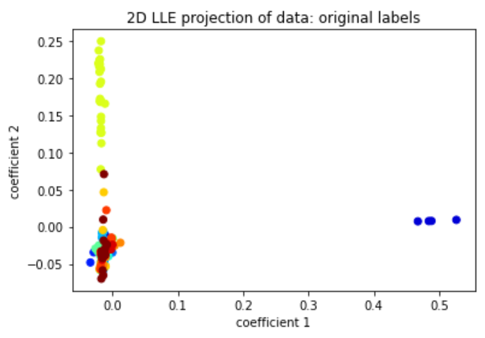


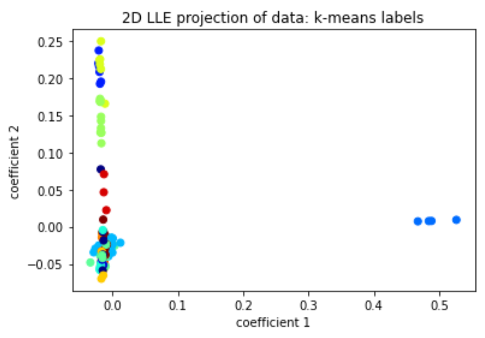


LLE k-means

In [92]:
```
lle = LocallyLinearEmbedding(n_components=4, n_neighbors=6, method='modified', eigen_solver='dense', random_state=random_state)

LLE_PROJECTION = lle.fit_transform(df) #IMPLEMENT_ME

plot_2d(LLE_PROJECTION, df_labels.label, '2D LLE projection of data: original labels')

km = KMeans(init='k-means++', n_clusters=14)

km.fit(LLE_PROJECTION)

plot_2d(LLE_PROJECTION, km.labels_, '2D LLE projection of data: k-means labels')

print(adjusted_mutual_info_score(df_labels.label, km.labels_))
```

```
0.33455814495 70319
```





2.2.3 From these results, which dimensionality reduction method would you advise? LLE, UMAP, or SOM?

- Based on the results i would recommend UMAP since it has the highest adjusted mutual information score in this specific scenario. UMAP is easier to define and run. SOM generally uses up more computational time and requires alot of resources in order to be stable.