bioqtm385-inclass-03b-dimensionality-reduction-spike-sorting-2024

October 3, 2024

##BIO/QTM 385: In class exercise for Wednesday, September 25th (dimensionality reduction, continued)

(Answers to these questions will be part of Homework #2, due on 10/2)

Carol Zhou. Collaborated with Aanya Vusirikala and Sweta Balaji. Generative AI is used for Question 2. I asked AI to transform the gmm_labels and graph the line plot using the same color as gmm_plot.

For this exercise, we will explore how dimensionality reduction works when applied to some real data. As always, all questions to be answered will be in blue and places to write your answers will be in green. Work on this section after finishing the previous notebook.

###Import Useful Modules

[1]:
```
#import various useful packages
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as random
import pandas as pd
%matplotlib inline

#importing dimensionality reduction packages
from sklearn.decomposition import PCA
from sklearn.decomposition import NMF
from sklearn.manifold import MDS
from sklearn.manifold import Isomap
from sklearn.manifold import LocallyLinearEmbedding
from sklearn.manifold import TSNE
!pip install -q umap-learn[plot]
import umap

#importing clustering packages
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
```

```
                        56.9/56.9 kB
897.6 kB/s eta 0:00:00
                        18.3/18.3 MB
32.0 MB/s eta 0:00:00
```

```
                                     85.7/85.7 kB
3.1 MB/s eta 0:00:00
```

###Spike Sorting Spike sorting is a common (and annoying) problem in electrophysiological data, where spikes from multiple neurons appear on a single electrode and one wishes to assign each spike to its associated neuron. This problem will deal with an example neural data set taken by Prof. Samuel Sober in the Emory Biology Department.

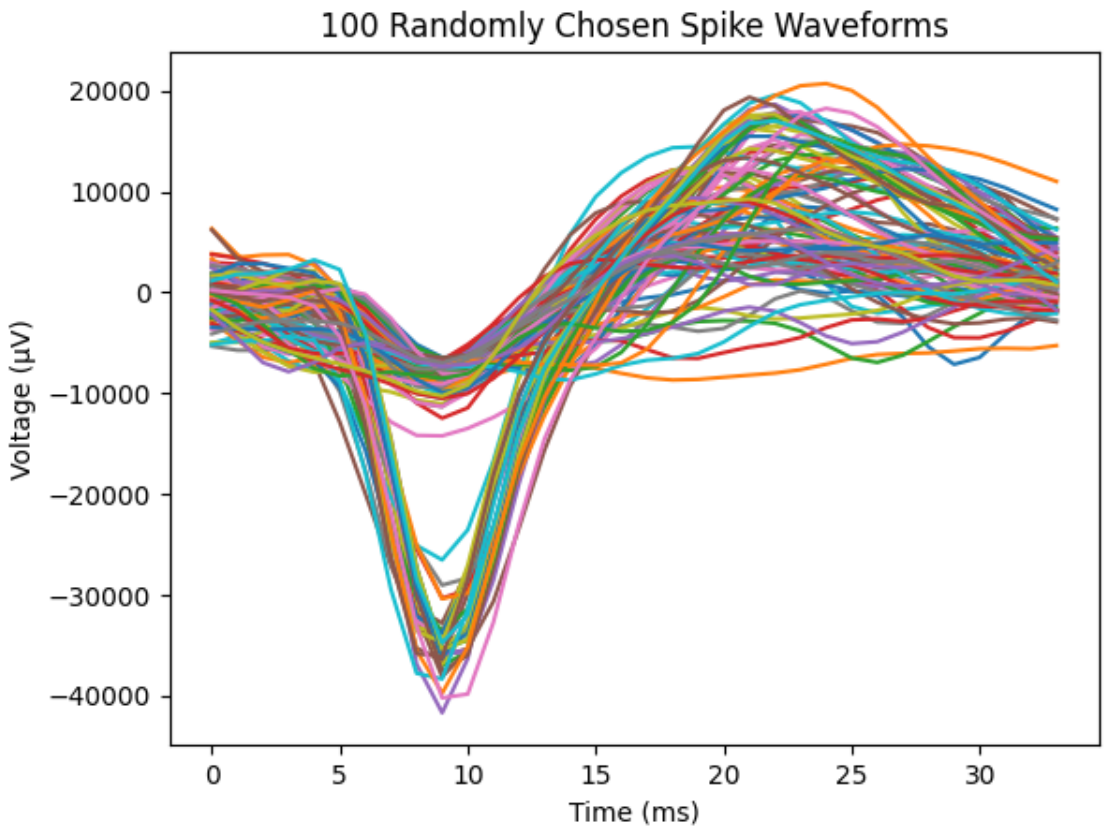
To start, we will import the data into the notebook as an an $N \times d$ matrix, where each row is a different voltage recording from an electrical channel during a spike, and each column is a different time point. Time is increasing with along the columns, with each column representing a time increase of 1ms (t=0,1,...,33 ms), and the units of the electrical voltages are in μV. There are 3636 different recordings here.

[2]:
```
url = 'https://raw.githubusercontent.com/gordonberman/bioqtm385_fall2020/master/
 ↪data/spike_data.csv'
spike_data_df = pd.read_csv(url,header=None)
spike_data = spike_data_df.to_numpy()
np.shape(spike_data)
```

[2]:
```
(3636, 34)
```

Question #1: Plot 100 randomly-chosen spike waveforms (i.e., rows) from the data. How many different 'types' of spike patterns do you see?

[ ]:
```
random_rows = random.sample(range(spike_data.shape[0]), 100)

for i in range(100):
  plt.plot((spike_data[random_rows, :])[i, :])

plt.xlabel("Time (ms)")
plt.ylabel("Voltage (μV)")
plt.title("100 Randomly Chosen Spike Waveforms")
plt.show()
```



How many different 'types' of spike patterns do you see? I see potentially three types of spike patterns. There is one that goes a little negative at time = 9 ms, and there is another goes more negative at time = 9 ms. At 20-25 ms, there are a few lines that go negative. This could be another type of spike pattern.

Question #2: In general, each 'type' of spike likely corresponds to a different neuron. Using all of the dimensionality reduction and clustering techniques you know so far (or others that you happen to know or look up):

a) determine how many neurons are in your data set

and

b) assign each recording to the appropriate neuron

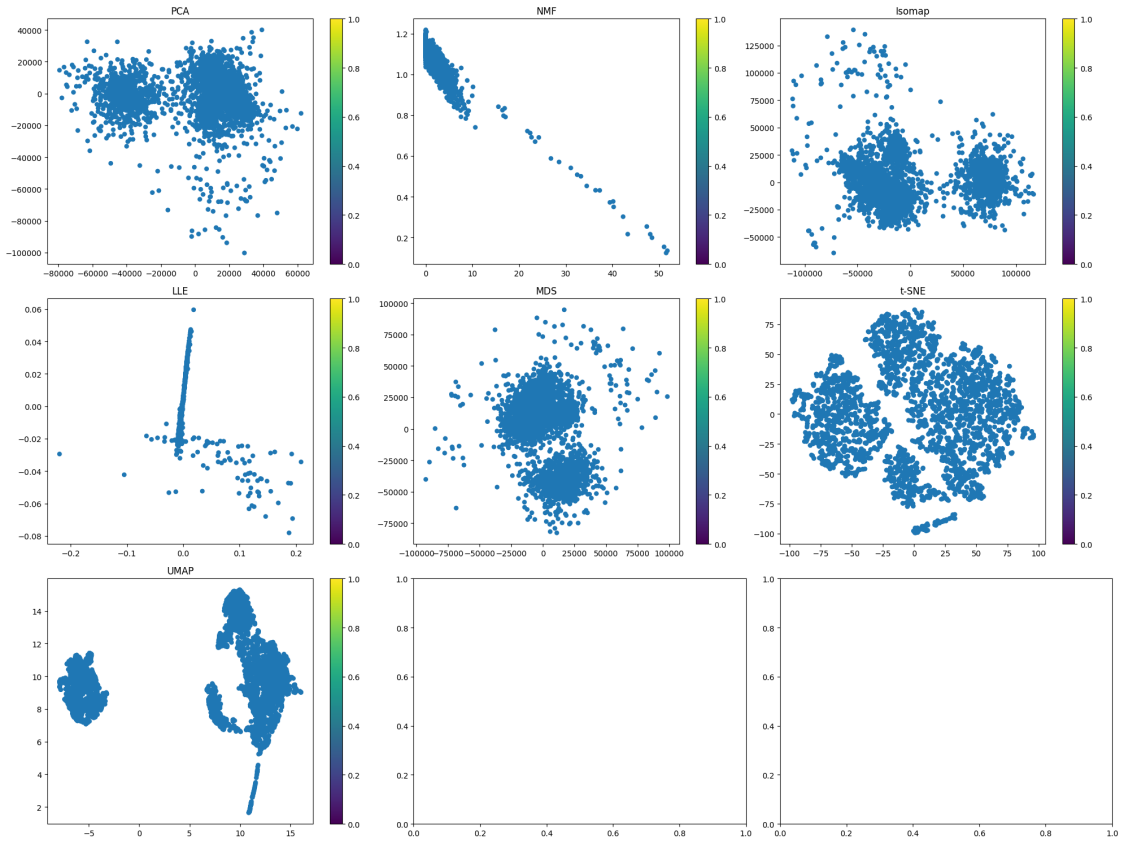
Use plots and quantitative evidence to back-up your assertions.

[9]:
```
fig, axes = plt.subplots(3, 3, figsize=(20, 15))
axes = axes.ravel()

# PCA
spikePCA = PCA()
spikePCA.fit(spike_data)
```

```
projections_PCA = spikePCA.transform(spike_data)
axes[0].scatter(projections_PCA[:, 0], projections_PCA[:, 1], edgecolor="none",␣
 ↪cmap=plt.colormaps['nipy_spectral'])
axes[0].set_title('PCA')
fig.colorbar(axes[0].collections[0], ax=axes[0])

# NMF
# Since the dataset contains negative values, we can shift all values up by␣
 ↪adding the absolute value of the minimum
spike_data_shifted = spike_data - np.min(spike_data)
spikeNMF = NMF(n_components=2)
projections_NMF = spikeNMF.fit_transform(spike_data_shifted)
axes[1].scatter(projections_NMF[:, 0], projections_NMF[:, 1], edgecolor="none",␣
 ↪cmap=plt.colormaps['nipy_spectral'])
axes[1].set_title('NMF')
fig.colorbar(axes[1].collections[0], ax=axes[1])

# Isomap
isomap = Isomap(n_components=2, n_neighbors=10)
projections_isomap = isomap.fit_transform(spike_data)
axes[2].scatter(projections_isomap[:, 0], projections_isomap[:, 1],␣
 ↪edgecolor="none", cmap=plt.colormaps['nipy_spectral'])
axes[2].set_title('Isomap')
fig.colorbar(axes[2].collections[0], ax=axes[2])

# Locally Linear Embedding (LLE)
lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10)
projections_lle = lle.fit_transform(spike_data)
axes[3].scatter(projections_lle[:, 0], projections_lle[:, 1], edgecolor="none",␣
 ↪cmap=plt.colormaps['nipy_spectral'])
axes[3].set_title('LLE')
fig.colorbar(axes[3].collections[0], ax=axes[3])

# Multi-Dimensional Scaling (MDS)
mds = MDS(n_components=2)
projections_mds = mds.fit_transform(spike_data)
axes[4].scatter(projections_mds[:, 0], projections_mds[:, 1], edgecolor="none",␣
 ↪cmap=plt.colormaps['nipy_spectral'])
axes[4].set_title('MDS')
fig.colorbar(axes[4].collections[0], ax=axes[4])

# t-SNE
tsne_p10 = TSNE(n_components=2, perplexity=10, method='exact')
projections_tsne_p10 = tsne_p10.fit_transform(spike_data)
axes[5].scatter(projections_tsne_p10[:, 0], projections_tsne_p10[:, 1],␣
 ↪edgecolor="none", cmap=plt.colormaps['nipy_spectral'])
```

```
axes[5].set_title('t-SNE')
fig.colorbar(axes[5].collections[0], ax=axes[5])

# UMAP
spikeUMAP = umap.UMAP(n_components=2, n_neighbors=10, min_dist=.1)
projections_umap = spikeUMAP.fit_transform(spike_data)
axes[6].scatter(projections_umap[:, 0], projections_umap[:, 1],␣
 ↪edgecolor="none", cmap=plt.colormaps['nipy_spectral'])
axes[6].set_title('UMAP')
fig.colorbar(axes[6].collections[0], ax=axes[6])

plt.tight_layout()
plt.show()
```

```
<ipython-input-9-3a1f0909ea6f>:8: UserWarning: No data for colormapping provided
via 'c'. Parameters 'cmap' will be ignored
  axes[0].scatter(projections_PCA[:, 0], projections_PCA[:, 1],
edgecolor="none", cmap=plt.colormaps['nipy_spectral'])
<ipython-input-9-3a1f0909ea6f>:17: UserWarning: No data for colormapping
provided via 'c'. Parameters 'cmap' will be ignored
  axes[1].scatter(projections_NMF[:, 0], projections_NMF[:, 1],
edgecolor="none", cmap=plt.colormaps['nipy_spectral'])
<ipython-input-9-3a1f0909ea6f>:24: UserWarning: No data for colormapping
provided via 'c'. Parameters 'cmap' will be ignored
  axes[2].scatter(projections_isomap[:, 0], projections_isomap[:, 1],
edgecolor="none", cmap=plt.colormaps['nipy_spectral'])
<ipython-input-9-3a1f0909ea6f>:31: UserWarning: No data for colormapping
provided via 'c'. Parameters 'cmap' will be ignored
  axes[3].scatter(projections_lle[:, 0], projections_lle[:, 1],
edgecolor="none", cmap=plt.colormaps['nipy_spectral'])
<ipython-input-9-3a1f0909ea6f>:38: UserWarning: No data for colormapping
provided via 'c'. Parameters 'cmap' will be ignored
  axes[4].scatter(projections_mds[:, 0], projections_mds[:, 1],
edgecolor="none", cmap=plt.colormaps['nipy_spectral'])
<ipython-input-9-3a1f0909ea6f>:45: UserWarning: No data for colormapping
provided via 'c'. Parameters 'cmap' will be ignored
  axes[5].scatter(projections_tsne_p10[:, 0], projections_tsne_p10[:, 1],
edgecolor="none", cmap=plt.colormaps['nipy_spectral'])
<ipython-input-9-3a1f0909ea6f>:52: UserWarning: No data for colormapping
provided via 'c'. Parameters 'cmap' will be ignored
  axes[6].scatter(projections_umap[:, 0], projections_umap[:, 1],
edgecolor="none", cmap=plt.colormaps['nipy_spectral'])
```

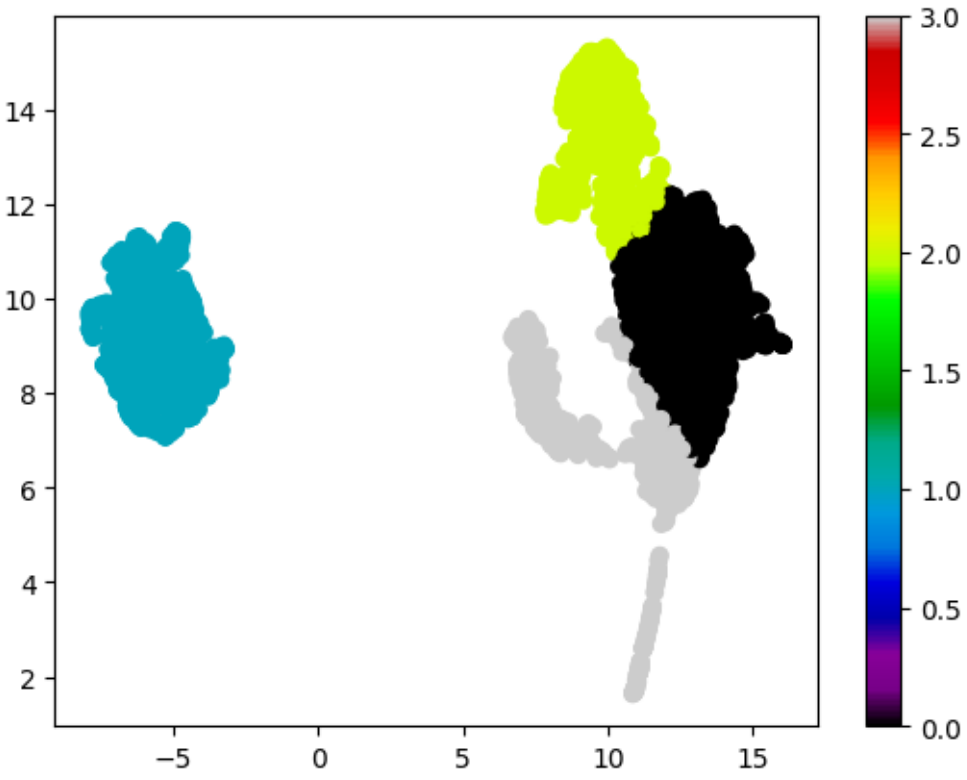


[3]:
```
spikeUMAP = umap.UMAP(n_components=2, n_neighbors=10, min_dist=.1)
projections_umap = spikeUMAP.fit_transform(spike_data)
```

[5]:
```
# @title More Helper Functions

from matplotlib.patches import Ellipse

def draw_ellipse(position, covariance, ax=None, **kwargs):
    """Draw an ellipse with a given position and covariance"""
    ax = ax or plt.gca()

    # Convert covariance to principal axes
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    # Draw the Ellipse
```

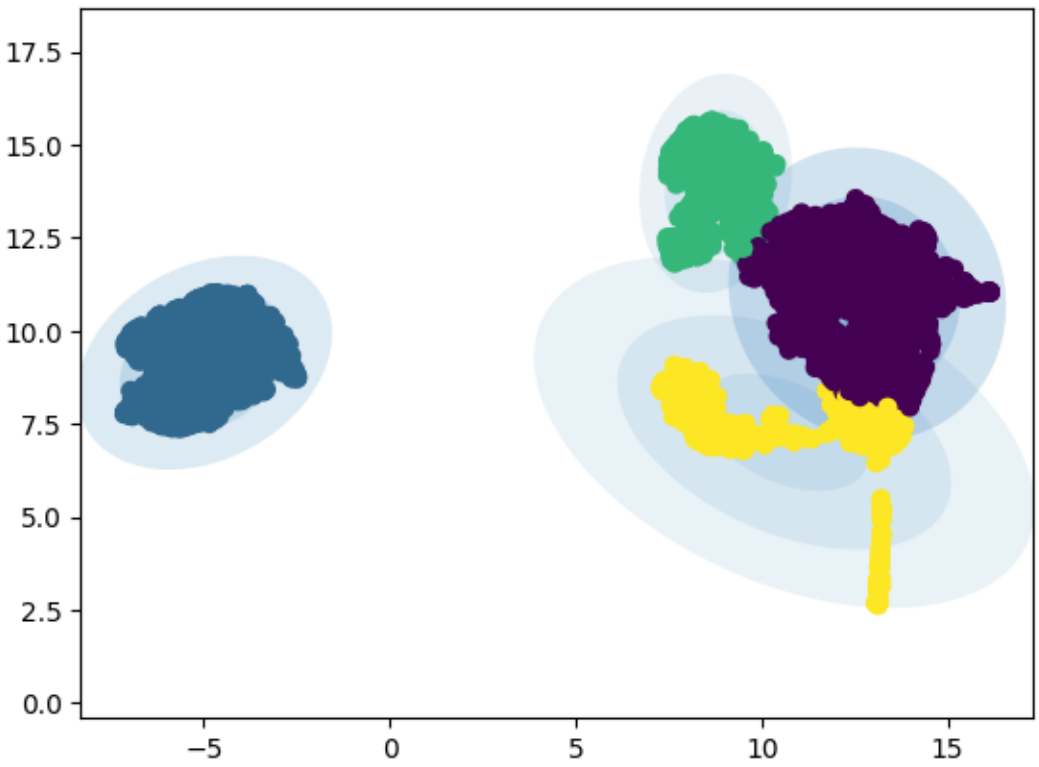
```
    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height,
                             angle, **kwargs))

def plot_gmm(gmm, X, label=True, ax=None):
    ax = ax or plt.gca()
    labels = gmm.fit(X).predict(X)
    if label:
        ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis', zorder=2)
    else:
        ax.scatter(X[:, 0], X[:, 1], s=40, zorder=2)
    ax.axis('equal')

    w_factor = 0.2 / gmm.weights_.max()
    for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * w_factor)
```

[11]:
```
km = KMeans(n_clusters=4)
km.fit(projections_umap)
km_labels = km.predict(projections_umap)

plt.scatter(projections_umap[:,0], projections_umap[:,1], c=km_labels, cmap=plt.
 ↪colormaps['nipy_spectral'])
plt.colorbar()
plt.show()
```

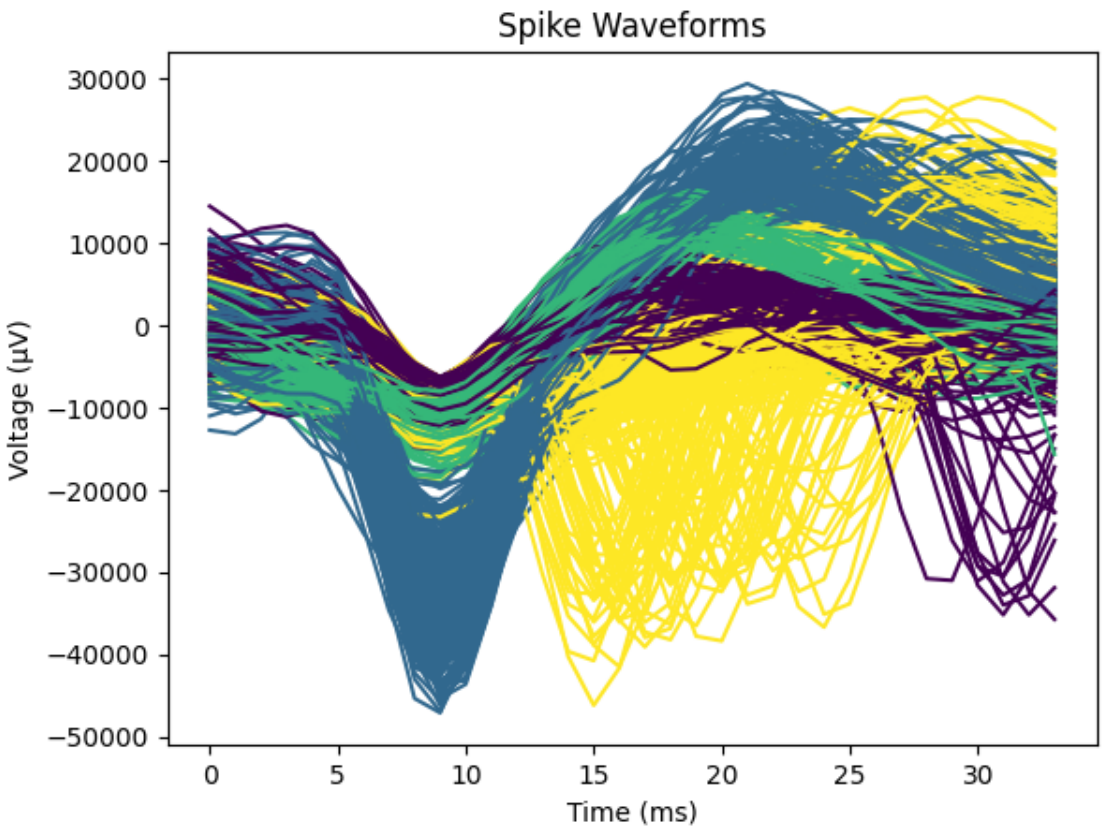


[16]:
```
gmm = GaussianMixture(n_components=4).fit(projections_umap)
gmm_labels = gmm.predict(projections_umap)
gmm_probs = gmm.predict_proba(projections_umap)
new_spikes = np.hstack((spike_data, gmm_labels.reshape(-1, 1)))

plot_gmm(gmm, projections_umap)
```

```
<ipython-input-5-6f9866724b46>:20: MatplotlibDeprecationWarning: Passing the
angle parameter of __init__() positionally is deprecated since Matplotlib 3.6;
the parameter will become keyword-only two minor releases later.
  ax.add_patch(Ellipse(position, nsig * width, nsig * height,
```



[17]:
```
new_spikes = np.hstack((spike_data, gmm_labels.reshape(-1, 1)))

# Define colors for GMM labels for consistency
unique_labels = np.unique(gmm_labels)
colors = plt.cm.viridis(np.linspace(0, 1, len(unique_labels)))
label_color_map = {label: colors[i] for i, label in enumerate(unique_labels)}

for i in range(3636):
    label = int(new_spikes[i, -1])
    plt.plot(new_spikes[i, :-1], color=label_color_map[label], label=f'Label␣
 ↪{label}' if i == 0 else "")

plt.xlabel('Time (ms)')
plt.ylabel('Voltage ( V)')
plt.title('Spike Waveforms')
plt.show()
```



[24]:
```
np.bincount(gmm_labels, minlength=4)/3636*100
```

[24]:
```
array([37.21122112, 28.21782178, 17.95929593, 16.61166117])
```

There are 4 different neurons. Out of the 7 dimensioinality reduction methods, t-SNE and UMAP have the best distict clusters, in which each cluster could indicate the a potential type of neurons. This is because both t-SNE and UMAP both preserve the local relationships and capture the non-linear structures in the data. For spike sorting, it is important to maintain the proximity of similar spikes to help identifying clusters that correspond to different neuronal spikes. Moreover, UMAP represents the underlying global structure better because unlike the closely-together clusters shown on the t-SNE plot, UMAP shows that there is one cluster that is very different from the rest. This is because UMAP balances local and global structure preservation, unlike t-SNE that solely preserves local structure. This is why I chose UMAP to proceed to the next clustering step. I have chose GMM clustering method to identify the individual types of neurons. Since neuronal spikes have different shapes, the clusters are likely to have different shapes and sizes as well. Additionally, GMM allows each point to belong to multiple clusters with different probabilities since it is possible that a spike could just be noise and is not a neuron. Thus, I think in this case, GMM is more suited than K-means. According to the "Spike Waveforms" plot, it is apparent that the blue lines (probably a type of neuron) are very distinct from the rest. Given that none of the percentage of each cluster/type of neuron that GMM identified is too small, I don't think any of the spikes are noise. Thus, I believe there are 4 types of neurons.