UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE MEDICINA

LICENCIATURA EN NEUROCIENCIAS

Asignatura:

Modelos Computacionales III

Título del Proyecto:

Algoritmo de Clasificación de Espigas No Supervisado basado en Coeficientes Wavelet, Componentes Principales y Agrupamiento tipo K-medias

Autores:

Andrea Fernanda Campos Pérez

José Carlos Barreras Maldonado

Nombre del Profesor:

Dr. Sergio Parra Sánchez

Fecha:

Febrero, 2021.

ÍNDICE

Introducción

En el actual campo de la neurociencia de sistemas los experimentos generalmente se basan en registrar la actividad extracelular de neuronas en áreas particulares del cerebro, con el objetivo de estudiar la estructura y la función de los circuitos neuronales subyacentes a procesos cognitivos, como la memoria, la percepción, el lenguaje, etc. Gracias a estos experimentos se han hecho relevantes descubrimientos acerca de las dinámicas de procesamiento de información sensorial en el cerebro [6], de toma de decisiones [6, 7, 8], de memoria de trabajo [7], y de mecanismos motores [8], por mencionar algunos.

Los registros extracelulares se realizan mediante electrodos implantados en el cerebro que captan los potenciales de acción (de ahora en adelante llamados espigas) de las neuronas cercanas, y también la actividad indistinguible de neuronas lejanas. La identificación de qué actividad corresponde a qué neurona se realiza mediante una técnica de procesamiento de la señal conocida como clasificación de espigas o *Spike Sorting* [2,3]. Dicha identificación se basa en la forma de las espigas, que depende de la morfología de la neurona y de su distancia relativa al electrodo.

Implementar algoritmos de clasificación de espigas es imperativo cuando queremos analizar las dinámicas individuales de respuesta y codificación neuronal [3,4]. Por ejemplo, es importante distinguir las espigas de cada neurona para determinar sus propiedades de tuning individual, las características de sus disparos, su relación con otras neuronas y con el potencial de campo local (LFP), sus dinámicas temporales de autocorrelación, etc. Estudiar las respuestas individuales neuronales de varias áreas cerebrales es indispensable para la generación de conclusiones relevantes acerca de los mecanismos subyacentes a procesos cognitivos normales y anormales, por lo que los algoritmos de clasificación de espigas se deben continuar estudiando, generando, evaluando y refinando.

En el presente proyecto se planea desarrollar y aplicar un algoritmo clasificación de espigas, con el objetivo de optimizar su desempeño en el procesamiento de bases de datos de registros electrofisiológicos de macacos entrenados. Esperamos que los resultados de este proyecto permitan establecer un algoritmo para la clasificación de espigas óptimo que sea utilizado en favor de generar datos confiables de neuronas individuales para su posterior análisis.

Objetivos

General

Generar un algoritmo computacional óptimo y eficiente en el lenguaje de programación Python, para clasificar la actividad de neuronas individuales registradas en diferentes áreas cerebrales.

Específicos

1. Investigar las bases teóricas y metodológicas de los algoritmos de clasificación de espigas.
2. Generar un algoritmo de clasificación de espigas en Python.
3. Aplicar el algoritmo en registros de respuestas neuronales del cerebro del macaco para obtener resultados sobre el desempeño del programa.

Justificación

La clasificación de espigas es un paso vital de procesamiento de datos electrofisiológicos cerebrales ya que permite la obtención de la actividad (potenciales de acción o espigas) de neuronas individuales. Sin embargo, aún entendiendo su gran relevancia, en el campo actual de las neurociencias no existe un consenso sobre la mejor técnica para llevar a cabo este proceso, resultando en algoritmos no reproducibles entre laboratorios, arbitrarios y ampliamente sesgados. Por ende, la importancia de este proyecto es obtener un algoritmo confiable, basado en la literatura [2, 3, 4, 5], para separar las respuestas de neuronas individuales de registros cerebrales en base a la forma de su disparo. Luego, estas respuestas unitarias podrán analizarse estadísticamente con el objetivo de generar conclusiones relevantes sobre la dinámica de procesos cognitivos complejos, en condiciones típicas y atípicas, como la memoria de trabajo, la toma de decisiones, la percepción, la motricidad, entre otros [6, 7, 8]. En el campo de la neurociencia cognitiva y de sistemas, entender las características computacionales de las neuronas individuales es el primer paso para entender el funcionamiento cerebral.

Método y Resultados

El algoritmo de clasificación de espigas que se desarrolló consta de 7 secciones, basadas en la literatura [2, 3, 4, 5], las cuales son: **1)** Manejo de archivos NEV (Neural EVents); **2)** *Denoising* de los potenciales registrados por el electrodo; **3)** Alineamiento de los potenciales de acción mediante interpolación de *splines* cúbicos por partes (*piecewise*); **4)** Extracción y Selección de Características mediante Coeficientes Wavelets, Análisis de Componentes Principales (PCA) y criterio Kolmogorov-Smirnov; **5)** Agrupamiento mediante el algoritmo de *K-means*; **6)** Evaluación del resultado de la clasificación mediante coeficientes de correlación de Pearson y Prueba de hipótesis; y **7)** Guardado de los resultados. En los siguientes apartados se explica cada sección de procesamiento y se muestran los resultados correspondientes.

1) Manejo de archivos NEV (Neural EVents)

Los archivos NEV contienen la información proveniente de registros electrofisiológicos, en donde se incluyen los tiempos de disparo, canales o electrodos detectando estos potenciales de acción y la forma de las espigas. Los archivos disponibles para la evaluación del código fueron generados por el laboratorio del Dr. Román Rossi-Pool a partir de los registros electrofisiológicos en mono Rhesus durante tareas conductuales de discriminación bimodal. Una sesión de registro está conformada por múltiples ensayos de la tarea. A su vez, cada ensayo tiene un archivo NEV respectivo. Por lo tanto, la carpeta de cada sesión de registro contiene tantos archivos NEV como ensayos realizados por el mono durante dicha sesión.

El primer segmento del programa consiste en la apertura de los archivos NEV, extracción de la información contenida en estos y el almacenamiento de dicha información en archivos con extensión DAT que facilitan su manejo. Además, esta nueva serie de archivos se complementa con información asociada al protocolo y psicofísica de la tarea. Para esto, el código solicita al usuario la ruta hacia la carpeta con los archivos NEV de interés, junto con la ruta en la cual se desean almacenar los nuevos archivos DAT. En esta última, los nuevos archivos son almacenados en una carpeta que lleva el mismo nombre que aquella que contiene los archivos NEV.

Una vez que el código tiene esta información, genera una lista ordenada con el nombre de cada uno de los archivos NEV sobre el cual realizará una iteración para la apertura secuencial de estos. La apertura de los archivos se realiza con la función *NevFile* de la librería *brpylib* que ofrece de manera gratuita la compañía *Blackrock Mycrosystems* **©** *.* Una vez que los archivos son abiertos y su contenido es almacenado en variables temporales, el archivo se cierra y se procede al guardado de estas variables en los archivos de extensión DAT.

<u>Resultado del manejo de archivos NEV</u>

Figura 1. Ejemplo de algunos de los archivos de extensión DAT generados a partir de los archivos NEV de los registros electrofisiológicos en monos. Los archivos .dat son los que se usarán como *input* de la siguiente sección.

2) Denoising de los potenciales registrados por el electrodo

Para proceder con el *Spike Sorting*, el código solicita la ruta hacia la nueva carpeta con los archivos DAT. El código despliega entonces la cantidad de ensayos contenidos en la sesión de registro y pide al usuario elegir entre la selección de un rango definido de ensayos para realizar el *Sorting*, en donde el usuario debe determinar el ensayo inicial y final, o la selección aleatoria, donde el usuario debe elegir la cantidad de ensayos que serán elegidos aleatoriamente. A continuación se despliegan los electrodos con los que se realizó el registro durante el experimento electrofisiológico y el usuario debe elegir sobre cual proceder. También, se debe indicar la ruta en la cual se desean almacenar las imágenes resultantes y el archivo con los resultados del *Spike Sorting*. En el caso, por ejemplo, de haber elegido el electrodo 1, en esta ruta se crea una carpeta llamada “electrodo 1”. A su vez, en esta carpeta se genera una llamada “imágenes” en la cual se almacenan las figuras resultantes del *sorting*.

Posteriormente, el usuario decide entre realizar o no la eliminación ruido (*denoising*) sobre las formas de onda de las espigas. La eliminación de ruido consiste en la aplicación de un filtro Savitzky-Golay por medio de la función *savgol_filter* de la librería *scipy.signal.* Este tipo de filtro consiste en el cálculo de una regresión polinomial local, en este caso de grado 3, para determinar el valor de cada punto y obtener una versión suavizada de los datos. Se utiliza un paso de 5, entendido como la cantidad de coeficientes del polinomio.

Resultado del *denoising* de las formas de onda de las espigas

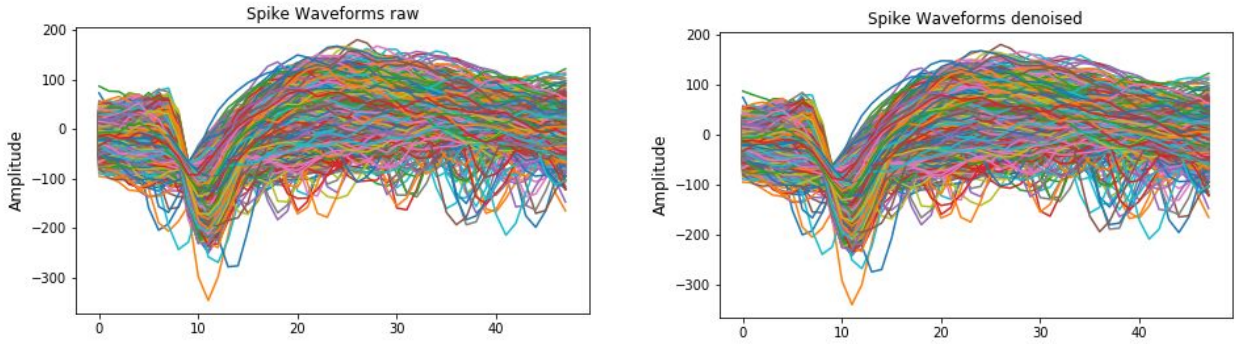

Figura 2. Formas de onda de las espigas antes (*Spike Waveforms raw*) y después (*Spike Waveforms denoised*) de la eliminación del ruido. Si el usuario elige no hacer el *denoising*, las formas de onda de las espigas permanecen como en la figura de *Spike Waveforms raw*.

3) Alineamiento de los potenciales de acción mediante interpolación de splines cúbicos por partes (piecewise)

Una vez que se procedió con el *denoising* (o no, aunque es recomendable que sí se realice), se procede a la interpolación de las formas de onda mediante *splines* cúbicos *piecewise* y la alineación de los potenciales al mismo pico mínimo. Esto es importante para poder comparar los potenciales, minimizando la variabilidad entre las formas de las espigas en una misma

ventana. Si se dejan a los potenciales desalineados, habrían efectos en la extracción de características, sobre todo en la Transformada de Wavelet.

La interpolación se hizo mediante el interpolador *PchipInterpolator* de la librería *scipy.interpolate*. Este interpolador permite la obtención de nuevos puntos continuos en base a los puntos que ya conocemos de nuestras ondas mediante *splines* cúbicas. Este *spline* ajusta polinomios de grado 3 en pequeñas partes del set de valores para ajustarlos, generado errores bajos en la estimación de los nuevos puntos continuos.

Finalmente, la alineación se hizo mediante la ubicación del instante del valor mínimo interpolado, es decir, del pico mínimo y la posterior organización del arreglo de las formas de onda interpoladas a este valor mínimo

Resultado de la interpolación y la alineación de las formas de onda

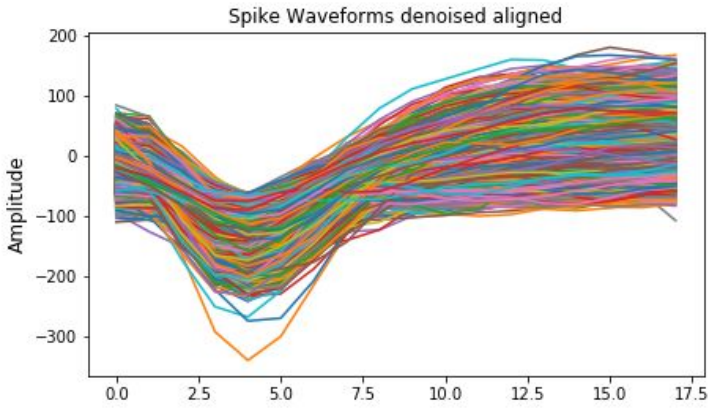

Figura 3. Formas de onda de los potenciales de acción después de la interpolación y el alineamiento al mismo pico mínimo. Con este resultado ya es posible proceder a la extracción de características.

4) Extracción y Selección de Características mediante Coeficientes Wavelets, Análisis de Componentes Principales (PCA) y criterio Kolmogorov-Smirnov

En esta sección del código se hace la extracción de características de cada forma de onda alineada para reducir la dimensionalidad de los datos a las dimensiones más informativas. Esto se logra mediante la transformada de wavelet y PCA de los datos. Las *features* que extraemos y seleccionamos nos ayudan a separar las ondas según sus características en el espacio fase para luego agruparlas.

Primero se realiza la transformada wavelet, que es una representación de tiempo-frecuencia de la señal con resolución óptima, y se define como la convolución entre la señal (x(t)) y la función wavelet ψ a,b (t). En este caso la transformada es de 4 niveles (multinivel), lo que nos permite obtener detalles de la señal a diferentes escalas, y se usa la wavelet de Haar.

Este procedimiento se hace con la función *wavedec* de la librería *PyWavelets (pywt).* Como resultado obtenemos los coeficientes wavelet que caracterizan la forma de cada onda de manera más discriminativa.

Luego, se realiza el PCA para encontrar otras características de nuestras ondas de manera automática. PCA se aboca a encontrar un set ordenado de vectores de base ortogonales que capturen las direcciones en los datos de mayor varianza. Estos vectores se obtienen al computar los eigenvectores de la matriz de covarianza de los datos. Para representar cualquier espiga, los componentes principales se escalan y se suman. El factor de escala de cada componente se denomina PCA score, y es lo que guardamos, así como los coeficientes Wavelet. Este análisis se hace mediante la clase *PCA* dentro de nuestro módulo de funciones (*functions.pyx*). El código fuente de esta clase pertenece a la librería *matplotlib.mlab.*

Posteriormente, ya que los componentes están ordenados en término de cuánta variabilidad capturan, elegimos los primeros *k* componentes que describen el 90% de la varianza de los datos (i.e., los más representativos). Estos PCA score y los coeficientes Wavelets se concatenan en una sola matriz que se llama *Features.*

El paso final de esta sección es la selección de las *Features,* ya sean coeficientes wavelet o PCA scores, que separen mejor las diferentes clases de espigas. La única condición que buscamos es que estas tengan una distribución multimodal, lo que nos indica la presencia de más de una forma de espigas, es decir, de más de una neurona.

Para realizar esta selección usamos el test *Kolmogorov - Smirnov* de normalidad que computa la comparación entre la distribución acumulativa de cada *Feature* real con la distribución acumulativa de una distribución gaussiana que tenga la misma μ y σ que estos datos reales. Este cálculo se hace mediante la función *scipy.stats.kstest* que nos devuelve el valor del estadístico KS, que indica el grado de normalidad de cada *Feature*, y su valor de p, que nos indica la significancia del estadístico. Usamos los valores del estadístico KS para ordenar las *Features* de la menos a la más normal, y nos quedamos con las menos normales, ya que esto nos sugiere una distribución multimodal. Las características seleccionadas proveen una representación comprimida de las espigas que sirven como un input para el algoritmo de agrupamiento.

A partir de estos resultados, el código despliega una gráfica tridimensional de las tres características más representativas de los datos. Además, se muestra la matriz de relación entre todas las parejas de características y en la diagonal se muestra la distribución individual de cada una. Por último, se despliega una gráfica de dos dimensiones para cada una de las parejas de las tres *Features* principales, junto con un mapa de calor que refleja la densidad de los datos proyectados sobre estas tres características. El objetivo de estas distintas visualizaciones es la de ayudarnos a decidir en cuantos grupos (clusters) dividiremos nuestros datos.

Resultado de la extracción y selección de características

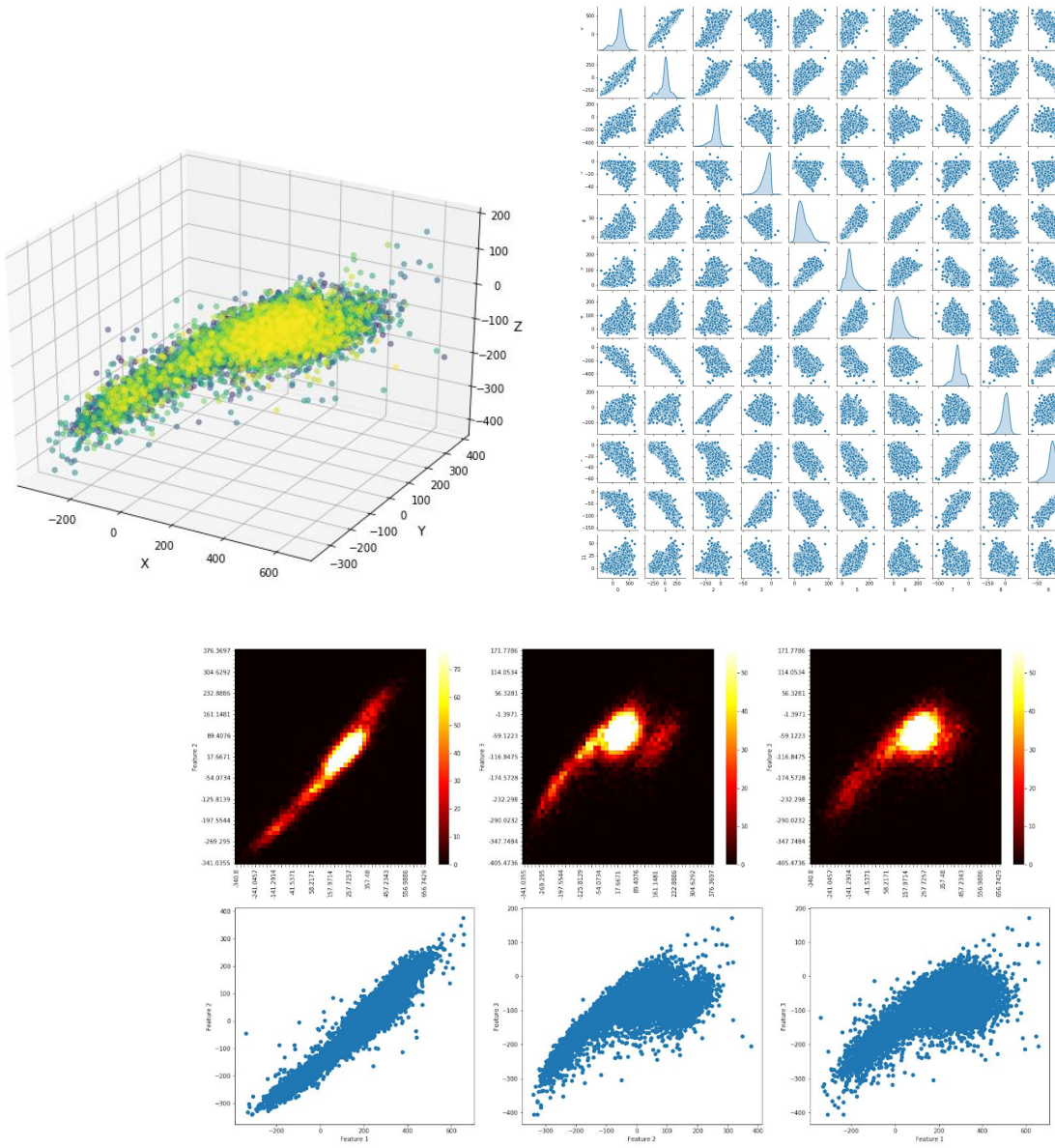

Figura 4. Visualización de las características de los datos extraídas y seleccionadas. En el panel de superior izquierdo se muestra la representación tridimensional de las tres características más informativas de los datos. En el panel superior derecho se muestra la matriz de relación entre todos los pares posibles de *Features* (en este caso son 12); en la diagonal están las distribuciones de las 12 *Features* individuales. En el panel inferior se ilustra el mapa de calor que refleja la densidad de datos proyectados en el espacio de características y su correspondiente gráfico de dispersión. Cada gráfico tiene como eje un par de *Features* de entre las 3 más representativas. De izquierda a derecha: Feature 1 - Feature 2; Feature 2 - Feature 3; y Feature 1 - Feature 3.

5) Agrupamiento mediante el algoritmo de K-means

A partir de las características extraídas en el paso anterior, se realiza el algoritmo de agrupamiento K-means. Para esto se utiliza la función *KMeans* de la librería *sklearn.cluster.* El objetivo de este paso es separar el conjunto de espigas en aquellas que compartan características similares, de modo que cada grupo esté conformado por todas las espigas provenientes de una única neurona putativa.

Las gráficas desplegadas hasta este punto (Figura 4 sobre todo) ayudan al usuario a intuir la cantidad de neuronas que fueron detectadas por el electrodo, así que, en base a ellas el usuario debe indicar el número de grupos en los que quiere dividir sus datos. Una vez elegida esta cantidad, el código despliega un gráfico con la métrica de *silhouette* en relación a la cantidad de grupos elegidos más tres (Figura 5). La métrica de *silhouette* se calcula a través de la función *silhouette_score* de la librería *sklearn.metrics* y nos indica cuán similar es un objeto al resto de los elementos de su propio grupo (cohesión) en comparación con los elementos de los otros grupos (separación). De este modo, un valor promedio elevado en relación a una cierta cantidad de grupos (clusters), es una medida de confianza de que el agrupamiento de los elementos es correcto con esa cantidad de grupos. Una nota importante es que debido a que esta métrica depende de la comparación con elementos de otros grupos, su cálculo sólo tiene sentido cuando se busca más de un cluster.

En el siguiente paso, utilizando la información observada en esta última gráfica (Figura 5), el usuario indica de manera definitiva la cantidad de grupos para el algoritmo de K-means. Cuando el número de grupos elegidos es mayor a 2, se usa la función de *KMeans* de *sklearn*, mientras que si el número de clústeres es 1, se usa la función *kmeans* programada por nosotros que está dentro del módulo *functions.pyx.* Se realizó esta distinción por la función de *sklearn* no acepta menos de dos clústeres.

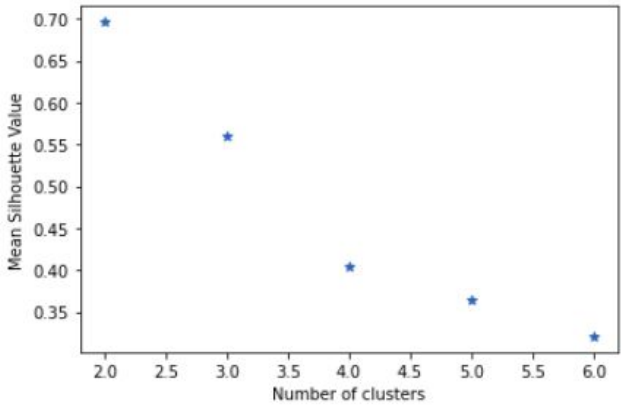

Figura 5. Gráfica con los valores de *silhouette* promedio para las distintas cantidades de grupos. En este caso el usuario indicó 3 grupos, por lo cual la gráfica llega hasta 6. Nótese que la gráfica comienza en 2 grupos, ya que la métrica de *silhoutte* requiere de la comparación de los elementos de al menos dos grupos. Nuestro código siempre calcula esta métrica desde 2 clusters hasta el número de clústeres elegidos por el usuario más tres.

Una vez que se indica la cantidad de grupos, y se computa el agrupamiento mediante k-means, se genera una gráfica de dispersión de los datos proyectados en el espacio de características donde el color indica su grupo o cluster. Se grafican tres proyecciones, una por cada pareja de las tres características más representativas de los datos (Figura 6).

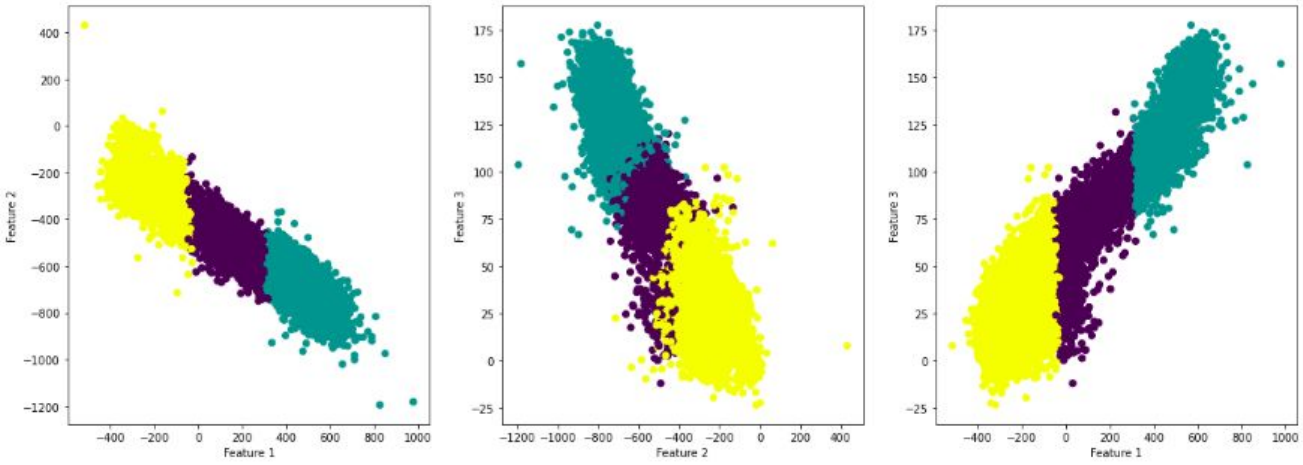

Figura 6. Cada gráfica tiene por ejes las distintas combinaciones en pares de las tres características más representativas de los datos. Los elementos de un mismo color pertenecen a un mismo grupo y por ende se trata de las espigas de una misma neurona putativa.

Inmediatamente después, el usuario debe indicar si desea (o no) realizar una eliminación de ruido de los clusters. Por defecto, este paso considera como ruido a toda espiga cuya distancia al centroide de su grupo respectivo es mayor a cuatro desviaciones estándar. No obstante, el código ofrece al usuario la opción de utilizar una cantidad distinta de desviaciones estándar a partir de la cual el elemento se considere como ruido. A partir de este punto, los elementos clasificados como ruido son considerados como un nuevo grupo.

Por último, a partir de los resultados del algoritmo de agrupamiento se generan distintas gráficas que permiten al usuario evaluar visualmente la calidad de la separación (y/o clasificación) y la eliminación del ruido. La primera es una gráfica de dispersión tridimensional en donde cada grupo puede distinguirse por su color. Cada eje de esta gráfica es una de las 3 características más representativas de los datos (Figura 7, superior izquierda). Después, se despliega una gráfica con las formas de espiga pertenecientes a cada grupo (Figura 7, superior derecha). Esta última gráfica se complementa con un mapa de calor de cada grupo que refleja la cantidad de espigas que comparten forma en términos de intensidad (Figura 7, inferior). La escala de intensidad por defecto de esta gráfica va de 0 a 0.35 veces el valor de densidad máximo. Sin embargo, el código permite al usuario cambiar la fracción que multiplica dicho valor de densidad máximo.

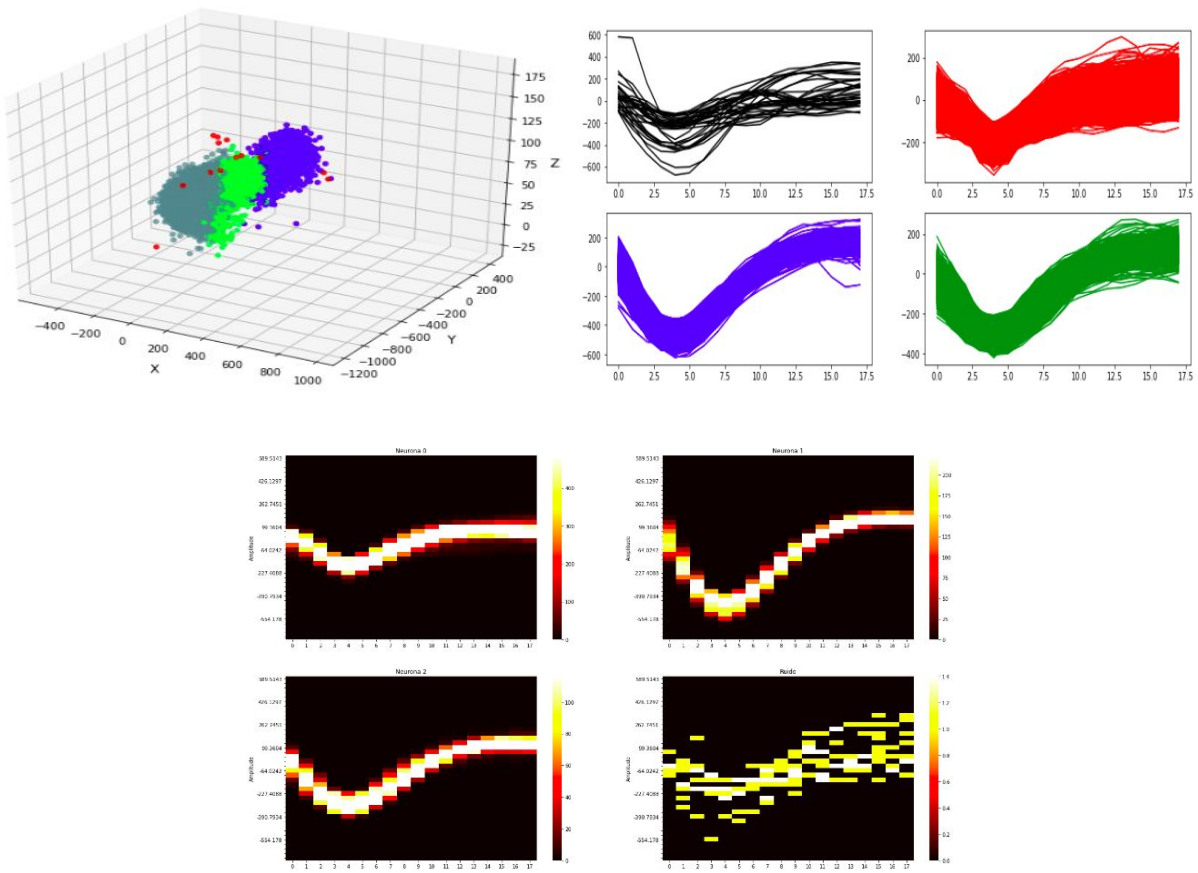

Figura 7. En el panel superior izquierdo se muestra la gráfica de dispersión tridimensional que tiene por ejes las 3 características más representativas de los datos. Los elementos de un mismo color pertenecen a un mismo grupo. En este caso los elementos de color rojo fueron aquellos clasificados como ruido. En el panel superior derecho se observan las formas de espigas dentro de cada grupo. En este caso las espigas de color negro representan a aquellas clasificadas como ruido. En la parte inferior se muestran los mapas de calor que reflejan la densidad de las distintas formas de espiga de cada grupo y del ruido.

La siguiente serie de gráficas consiste en un raster para cada uno de los grupos (clusters o neuronas). Cada raster tiene por eje de abscisas los tiempos de la tarea y por eje de ordenadas los ensayos de la misma. De este modo, la presencia de un potencial de acción perteneciente a una neurona putativa se grafica como una línea vertical en el raster del grupo correspondiente, en el tiempo y ensayo en el que fue detectado (Figura 8). Finalmente, se genera una gráfica que indica, para cada grupo, la cantidad de espigas detectadas en cada ensayo por neurona (o ruido si es el caso). Está gráfica permite evaluar la estabilidad que presentó cada neurona putativa a lo largo del registro (Figura 9).

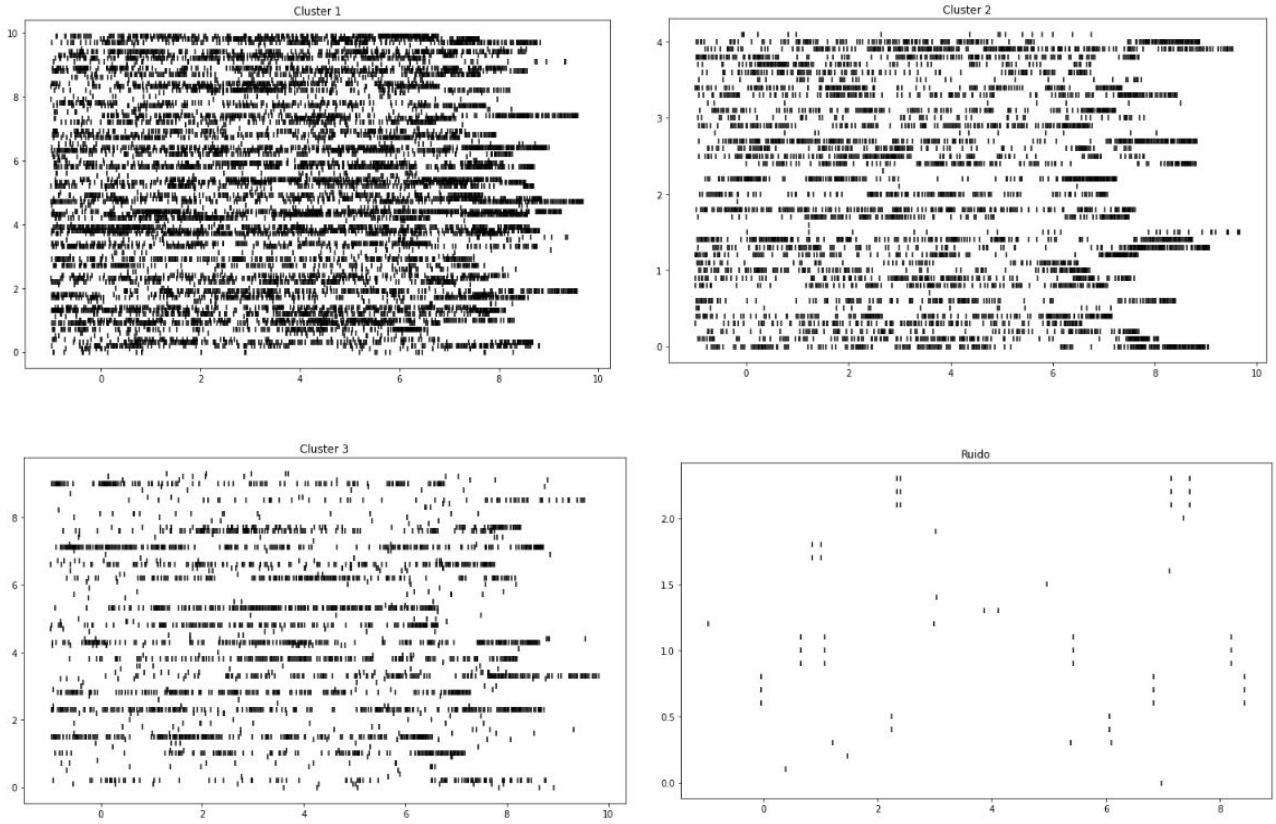

Figura 8. Gráficos raster de los distintos grupos. El eje de las abscisas representa el tiempo a lo largo de la tarea que se encuentra realizando el mono. El eje de las ordenadas son los distintos ensayos que conforman el registro y se encuentra en escala de 10. En este caso el registro tiene 100 ensayos, pero sólo uno de los grupos presentó actividad en todos ellos.

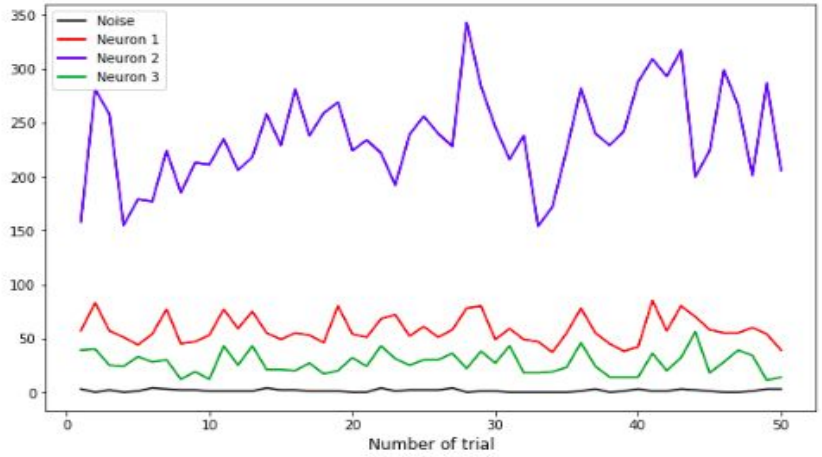

Figura 9. Gráfica de estabilidad para 3 neuronas. Se observa la cantidad de espigas totales que tiene cada uno de los grupos en cada ensayo. En este caso el registro cuenta con 50 ensayos. El hecho de que la cantidad de espigas oscila alrededor de valores similares indica una estabilidad en la actividad de la neurona putativa.

6) Evaluación del resultado de la clasificación mediante coeficientes de correlación de Pearson y Prueba de hipótesis

Esta penúltima sección del algoritmo se aboca a evaluar el resultado del agrupamiento realizado en la sección anterior mediante la correlación de Pearson entre el número de espigas por ensayo de cada uno de los grupos (neuronas), la prueba de hipótesis (valor p) de que no existe relación entre los fenómenos observados y el intervalo de confianza del 95% del coeficiente de correlación. Estas métricas se calculan mediante la función *corrcoef* de nuestro módulo *functions.pyx.*

El coeficiente de correlación de Pearson, que va de -1 a 1, nos indicará si las espigas de una neurona putativa están correlacionadas con las de las demás neuronas. Si el valor es cercano a 1 entonces existe una correlación fuerte entre ese par de neuronas, lo que nos sugiere que la clasificación podría ser incorrecta y que se trata de las espigas de una sola neurona. Sin embargo, debemos analizar con precaución estos resultados, porque podría ser que la actividad de las neuronas putativas esté correlacionada por el efecto de la atención durante la ejecución de una tarea cognitiva, por ejemplo.

Por otro lado, el valor de p indica la probabilidad de que no existe relación lineal entre los fenómenos observados. Por ende, si el valor de p es muy bajo sí están correlacionados, y viceversa. Finalmente, el intervalo de confianza propone un rango plausible de valores (inferior y superior) para cada coeficiente de correlación con una confianza del 95%. Si nuestro parámetro real está dentro del rango es altamente probable que sea correcto. Estas dos métricas proporcionan un apoyo al valor del coeficiente de correlación de Pearson y nos deja inferir si nuestra clasificación fue correcta.

<u>Resultado de la evaluación de la clasificación</u>

```
Los resultados de la correlación de Pearson entre el número de espigas por ensayo de la
s distintas neuronas son:

R: Matriz de correlación

[[1.         0.81530288]
 [0.81530288 1.        ]]

P: Prueba de hipótesis. No existe relación entre los fenómenos observados (p-value)

[[1.00000000e+00 2.49215173e-39]
 [2.49215173e-39 1.00000000e+00]]

RL: Limite inferior para Intervalo de confianza del 95%

[[1.        0.7557472]
 [0.7557472 1.       ]]

RU: Limite superior para Intervalo de confianza del 95%

[[1.         0.86148286]
 [0.86148286 1.        ]]
```

Figura 10. Ejemplo del resultado de la evaluación de la clasificación mediante correlación de Pearson y Prueba de hipótesis. Se muestra la matriz de correlación de las espigas por ensayo entre cada par de neuronas

clasificadas, en este ejemplo solo se clasificaron dos neuronas, por eso la matriz es de 2x2. Debajo se muestran los valores p de dichas correlaciones. Por último, se indican los límites inferiores y superiores del intervalo de confianza del 95% de los coeficientes de correlación de Pearson de la matriz superior. El valor de la diagonal de todas estas matrices es 1 pues contiene las métricas de cada variable contra sí misma. En este ejemplo podemos concluir que la clasificación no fue buena y que probablemente se trate de una sola neurona, debido a que el valor de R = 0.815, el valor p = 2.49^{-39} y el intervalo de confianza es (0.755, 0.861) dentro del cual se encuentra el valor de R de nuestros datos.

7) Guardado de los resultados

Como último paso, el código guarda los distintos datos resultantes de la clasificación de espigas en un archivo con extensión mat. Para esto, primero la información es almacenada en un diccionario con *keys* específicos. Los datos a almacenar, junto con sus *keys* respectivos, son los siguientes:

- *TimeStamps:* arreglo con los tiempos en los que se detectó cada espiga.
- *Spike_Trial:* arreglo con el número de los ensayos que conforman el registro.
- *class:* arreglo con el grupo al que pertenece cada tiempo de disparo en TimeStamps.
- *meta:* arreglo con el nombre de los grupos y lo que representan. Por ejemplo, el grupo 1 representa a la neurona 1, mientras que el grupo -1 representa al ruido.
- *corr:* arreglo con los resultados de las correlaciones y pruebas de hipótesis.
- *centroids:* arreglo con los centroides de cada uno de los grupos (clusters).
- *metric:* arreglo con los valores promedio de la métrica silhouette calculadas durante el sorting (solo se almacena cuando hay más de un cluster).

Para guardar el diccionario se utiliza la función *savemat* de la librería *scipy.io*. El archivo toma el nombre de "**PATH/sesión**_sorted_elec**X**_seq_**ensayo inicial**-**ensayo final**.mat" si la clasificación se hizo con un rango específico de ensayos, o "**PATH/sesión**_sorted_elec**X**_rnd_**número de ensayos**.mat" si la clasificación se hizo con un número de ensayos aleatoriamente elegidos. Este nuevo archivo es almacenado en la carpeta que se creó al inicio del programa, nombrada como "electrodo**X**" (dependiendo del electrodo que elegimos analizar), y en donde se estuvieron almacenando los gráficos resultantes de las secciones anteriores. A partir de este archivo con extensión mat, es posible cargar las neuronas directamente en Python utilizando la función *loadmat* de *scipy.io* para realizar algún análisis de interés sobre la actividad neuronal individual.

Resultado del guardado de datos de la clasificación

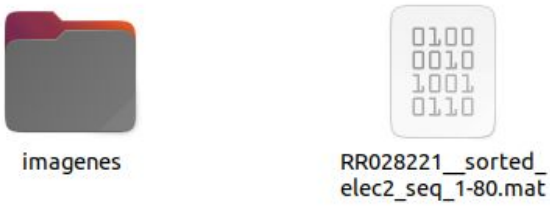

Figura 11. Archivo .mat donde se almacenaron los resultados de la clasificación de espigas. También, se muestra la carpeta “imagenes” donde se guardaron las gráficas de las secciones anteriores. Ambos archivos están dentro de la carpeta “electrodo**X**”, en el *path* que le indicamos al programa al inicio.

Anexo. Módulo *functions.pyx*

Algunos de los pasos del código requieren del módulo *functions.pyx, que* fue optimizado por medio de una compilación simple con Cython. En este se encuentra definida la clase PCA utilizada en la sección de 4 del algoritmo. Además, contiene las siguientes funciones desarrolladas por nosotros:

- *axis_ heatmap:* Computa la discretización de puntos de las características de las espigas y sus ejes, para poder formar las gráficas de calor de la Figura 4 inferior.
- *show_classification:* Genera la gráfica de dispersión tridimensional resultante del agrupamiento y la gráfica de las formas de espiga clasificadas (Figura 7 paneles superiores).
- *rasterplot:* Genera los rasters de cada uno de los grupos (Figura 8).
- *spikesheat_plot:* Genera el gráfico de calor de las formas de espigas de las neuronas clasificadas (Figura 7 inferior).
- *stability_plot:* Genera el gráfico de estabilidad de las neuronas clasificadas (Figura 9).
- *kmeans:* Permite realizar el algoritmo de agrupamiento K-means para un solo grupo.
- *corrcoef:* Calcula el coeficiente R de la correlación de Pearson entre todas las espigas de cada neurona (grupo) resultante de la clasificación, también calcula el valor p que indica la probabilidad de que no exista correlación entre los datos y el intervalo de confianza del 95% del coeficiente R.

Observaciones finales: Aspectos a mejorar

Nuestro algoritmo de clasificación de espigas tiene ciertas limitaciones. La primera es que solo se considera la transformada de Wavelet y el Análisis de Componentes Principales (PCA) para la extracción de características. En este rubro, en el futuro se podrían considerar otras características de los datos, como las geométricas, de distribución y de fase. Estas incluyen tomar en cuenta el valor mínimo y máximo de las ondas, su energía, su área, el valor mínimo, máximo y la diferencia entre su primer y segunda derivada, la media, mediana, moda de las ondas, la kurtosis, la desviación estándar, entre otras medidas de caracterización de las ondas que apoyen su clasificación.

Otra limitación es que solo usamos la técnica K-means de agrupamiento, por lo que se propone explorar otros algoritmos. Entre ellos, DBSCAN, por su nombre en inglés *Density-based Spatial Clustering of Applications with Noise*, en el que el agrupamiento de datos se basa en densidad porque encuentra un número de grupos (clusters) comenzando por una estimación de la distribución de densidad de los datos correspondientes. Otro algoritmo que se podría explorar es el *soft K-means* que se basa en la asignación probabilística de los datos a los clústeres en contraste de la asignación basada en la menor distancia euclidiana al centroide, como es el caso del algoritmo de *hard K-means* usado en este trabajo. Adicionalmente, se podría explorar el algoritmo *K-TOPS* [1] que se basa en los principios de K-means más la optimización de la plantilla o *template* en el espacio fase para el agrupamiento.

Por otra parte, en este programa la evaluación de los resultados de la clasificación se hace únicamente mediante la correlación de Pearson entre las espigas de las neuronas clasificadas, la prueba de hipótesis de que estas no estén correlacionadas y el intervalo de confianza del coeficiente R. Sin embargo, en la actualidad se ha propuesto [4, 5] la aplicación del algoritmo de clasificación en conjuntos de datos simulados, donde se conoce la identidad de cada espiga, con el objetivo de evaluar el desempeño y la precisión de la clasificación. Dichos datos simulados deben imitar las características de los registros reales en términos de distribución de ruido, características de las espigas, etc.

Por último, este programa para la clasificación de espigas únicamente fue optimizado mediante la compilación simple del módulo de funciones con Cython, ya que existieron dificultades en la aplicación de Numba por incompatibilidades en algunas funciones. Debido a esto se propone explorar la plausibilidad de eficientar aún más el algoritmo mediante la aplicación más robusta de Cython, las modificaciones de las funciones para que sean compatibles con Numba, e inclusive mediante *memoization* y *look-up tables*. Sobre todo en la sección de la graficación, ya que la parte de los cálculos son rápidos al realizarse con funciones de Numpy y SciPy principalmente.

Referencias

1. Caro-Martín, C. R., Delgado-García, J. M., Gruart, A., & Sánchez-Campusano, R. (2018). Spike sorting based on shape, phase, and distribution features, and K-TOPS clustering with validity and error indices. *Scientific Reports, 8*(1), 17796.
2. Lewicki, M. (1998). A review of methods for spike sorting: The detection and classification of neural action potentials. *Network: Comput. Neural Syst., 9*, R53–R78
3. Quiroga, R. (2007). Spike sorting. *Scholarpedia 2,* 3583.
4. Quiroga, R. (2012). Spike sorting. *Curr. Biol. 22*, R45–R46.
5. Quiroga, R. Q., Nadasdy, Z., & Ben-Shaul, Y. (2004). Unsupervised Spike Detection and Sorting with Wavelets and Superparamagnetic Clustering. *Neural Computation, 16*(8), 1661–1687.
6. Romo, R., Hernández, A. & Zainos, A. (2004). Neuronal correlates of a perceptual decision in ventral premotor cortex. *Neuron, 41*, 165-173.
7. Rossi-Pool, R., Zainos, A., Alvarez, M., Zizumbo, J., Vergara, J., & Romo, R. (2017). Decoding a Decision Process in the Neuronal Population of Dorsal Premotor Cortex. *Neuron, 96(*6), 1432-1446.e7.
8. Thura, D., & Cisek, P. (2014). Deliberation and Commitment in the Premotor and Primary Motor Cortex during Dynamic Decision Making. *Neuron, 81*(6), 1401–1416.