Desarrollo de sistema para calibrar electromiogramas de forma automática

D. Anchuela Cantarero

Ingeniería Biomédica, Universidad San Pablo CEU, Madrid, España, d.anchuela@usp.ceu.es

Resumen

La hago más adelante supongo.

# Introducción.

El reciente avance que ha tenido el aprendizaje automático en los últimos años nos abre las puertas a poder enfocar de una manera más eficiente problemas que anteriormente resultaban más complejos. Por otra parte, uno de los principales campos dónde se puede ver el potencial de este tipo de tecnología es en la medicina. El aprendizaje automático juega un papel fundamental a la hora de realizar toma de decisiones, reconocimiento de patrones en diferentes pacientes, predicción de posibles enfermedades, etc. Esto se logra gracias a la facilidad que tienen los ordenadores para procesar miles de datos en poco tiempo.

En este estudio, aprovecharemos las técnicas que nos da este campo para diseñar un calibrador automático que controlará un juego a partir de una señal de electromiograma. La aplicación del aprendizaje automático en este caso está en que el programa recogerá los datos de la señal de EMG del jugador para posteriormente ajustar los parámetros del juego de manera que cualquier persona pueda jugar sin que nosotros tengamos que ajustar los parámetros manualmente, lo cual es bueno ya que aunque fuésemos nosotros los que quisiéramos hacerlo manualmente, nunca podríamos producir un resultado tan bueno como el que conseguirá el programa después de entrenarse con los datos de ese jugador.

En este caso aplicaremos el calibrador al famoso juego del *T-Rex* de *Google Chrome*. Nosotros partimos de que ya tenemos un programa que hace saltar al dinosaurio cuando detecta una contracción en la señal de EMG. Sin embargo, parece lógico pensar que la contracción no se va a detectar de la misma manera en todas las personas ya que probablemente se detecte mejor una contracción en un adulto que en un niño ya que a priori tiene menos fuerza. En consecuencia, lo que vamos a hacer en este estudio es crear un programa capaz de entrenarse a partir de unas cuantas contracciones del jugador para posteriormente ajustar los parámetros del juego de manera que se haga saltar al dinosaurio solo en los momentos en los que ese jugador en concreto está haciendo una contracción. A priori parece una aplicación algo simple por parte de este tipo de tecnología. Sin embargo, cabe mencionar que los principios que vamos a utilizar en este caso son aplicables a cualquier problema en el que se requiera adaptar automáticamente algún parámetro de nuestro dispositivo de acuerdo con las necesidades del paciente. Por ejemplo, si quisiéramos ajustar automáticamente los parámetros de un brazo robótico para una persona dependiendo de las distintas etapas de su vida.

Para llevar a cabo este proyecto, es necesario elegir correctamente qué algoritmo de aprendizaje automático es el óptimo teniendo en cuenta el tipo de datos que nos da una señal de EMG. Para ello, recogeremos muestras de EMG de diferentes personas, haremos un preprocesado de la señal para quedarnos con lo que verdaderamente necesitamos, que es detectar simplemente si hay contracción o no, hacer un preprocesado de los datos para de manera que se ajusten mejor a los algoritmos de aprendizaje automático, compararemos la puntuación obtenida del funcionamiento de esos algoritmos y por último implementaremos esta nueva funcionalidad en nuestro juego.

# Materiales y métodos.

A continuación, pasamos a explicar los materiales utilizados para llevar a cabo este estudio, así como la descripción de los métodos que hemos seguido.

## Recogida de muestras de electrocardiograma

Para poder realizar un correcto estudio de qué técnicas de preprocesado de datos y de agrupamiento ofrecen un mejor rendimiento, es necesario obtener varias muestras de EMG de diferentes personas, para posteriormente cuantificar el rendimiento de los distintos algoritmos y seleccionar el más adecuado para el problema.

## Placa *Bitalino*

La placa *Bitalino*[1]es un dispositivo ampliamente utilizado a la hora de la recogida de distintos tipos de parámetros fisiológicos debido a su gran versatilidad y a su facilidad de uso.

Las muestras de EMG las recogemos utilizando el software *Open Signals* [2], una placa *Bitalino,* y tres electrodos Para ello conectamos por Bluetooth el *Bitalino* al ordenador. Posteriormente la configuramos desde el *Open* *Signals* introduciendo la dirección MAC de la placa, y especificamos que vamos a recoger señales de EMG*.* Por otra parte, conectamos los electrodos a la terminal de EMG de la placa, y nos los ponemos en el brazo de la siguiente manera:

* Electrodos positivo y negativo conectados en dos puntos del antebrazo para recoger la diferencia de potencial del músculo al contraer.
* Electrodo neutro haciendo contacto con una zona del cuerpo sin actividad eléctrica como puede ser el codo.

Grabamos una muestra de actividad mioelectrica, y la guardamos como *.txt* para analizarla posteriormente.

## Preprocesamiento de los datos

Una vez tenemos las muestras de la señal de EMG, necesitamos preprocesarlas para optimizar el funcionamiento del algoritmo de *clustering (~~término supuestamente explicado previamente en el abstract(?))~~*~~.~~

## Extracción de funciones

Se conoce como ingeniería de funciones al hecho de extraer aquellas características de nuestros datos para que nuestros algoritmos funcionen mejor.

En nuestro caso solo necesitamos saber si se está produciendo o no una contracción. Por eso mismo, necesitamos procesar nuestra señal para obtener una nueva, de manera que apartemos de nuestro estudio toda aquella información que no es relevante en este caso. Una señal que representa una contracción en un EMG presenta muchas oscilaciones durante la contracción. En nuestro caso, estas oscilaciones complican mucho el trabajo ya que, como necesitamos analizar los puntos de esta señal, nos podemos encontrar que, en una contracción, el valor de EMG es 0 debido a que está oscilando. Haciendo este procesado de señal, nos quedamos con la señal simplificada en donde solamente se distingue si hay contracción o no (ver figura 1).

Gráfico, Histograma

Descripción generada automáticamente con confianza media

Figura 1: Ilustración del procesado de señal de EMG, donde se aprecia la línea rosa cuando no hay contracción y las demás representan la señal del EMG simplificada y ampliada en el momento de la contracción.

## Escalamiento

Una vez tenemos las muestras de la señal simplificada de EMG, necesitamos preprocesarlas para optimizar el funcionamiento del algoritmo de *clustering*.

El primer paso, que es obligatorio antes de iniciar con el preprocesado de los datos, es hacer un escalamiento [3]. Esto es necesario ya que los algoritmos de clustering usan las distancias entre los puntos para definir los clusters. Si los *features* no tienen la misma magnitud, aquellos con las escalas más grandes dominarán al calcular las distancias.

Hay muchas formas de reescalar datos para todos los lenguajes de programación destinados al análisis de datos. Nosotros utilizaremos la función *minmax\_scale* de Python, pero cualquier función que lo que haga sea transformar los datos para ponerlos entre 0 y 1, es suficiente.

## Transformación de características

El objetivo de este punto es aplicar una transformación matemática a nuestros datos para intentar que adopten una forma lo más parecida posible a una distribución normal. Este tipo de tratamiento para los datos es muy útil ya que pueden corregir problemas de posibles valores atípicos y para reducir la dependencia entre media y varianza.

Para este estudio probaremos los siguientes tipos de transformaciones, inspirados por las transformaciones de Box-Cox [4]. Los pequeños valores añadidos a las transformaciones se emplean para evitar divergencias cuando los datos son 0 (aunque en la transformación de raíz no es estrictamente necesario, se hace igualmente por simplificar la implementación):

* Transformación logarítmica:

Datos\_log = np.log(0.0000001+datos)

* Transformación de raíz:

Datos\_raiz = np.sqrt(0.01+datos)

* Transformación recíproca:

Datos\_reci = 1/(0.01+datos)

## Reducción de dimensionalidad

Los sets de datos que obtenemos del archivo de simplificación de la señal de EMG, están compuestos por todas esas funciones que extrajimos y que se combinan para formar cada punto del EMG. Muchas veces se da que estas funciones están correlacionadas, por lo que puede ser importante aplicar una reducción de dimensionalidad para optimizar los cálculos, teniendo en cuenta la cantidad de información útil que se podría perder. Para ello realizaremos un análisis de componentes principales [5] (PCA). Lo que hace este tipo de técnica es precisamente analizar las dimensiones de los datos que no aportan demasiada información importante para reducir el set de datos, optimizando el rendimiento general del programa.

## Separación de datos de prueba y entrenamiento

Para que el algoritmo de clustering funcione correctamente, es necesario dividir el set de datos en un conjunto de prueba y un conjunto de entrenamiento [6]. En nuestro caso estableceremos que un 70% es para prueba y el 30% restante para entrenamiento.

Es importante mencionar que, en nuestro caso, como queremos crear un calibrador para una persona en concreto, no debemos mezclar los sets de prueba y entrenamiento de diferentes personas para entrenar al algoritmo. Es decir, el algoritmo solo entrenará con los datos de entrenamiento de la persona en cuestión y predecirá con los datos de prueba de esa misma persona.

## Clustering

Una vez tenemos ya nuestros datos separados, necesitamos emplear nuestro modelo de *clustering*, que será el que determinará los diferentes grupos en los que se detecta que se separan nuestros datos.

Hay muchos algoritmos de *clustering* diferentes, pero recordando cuál es nuestro propósito, debemos hacer una selección sobre cuáles nos serán útiles y cuáles no. En nuestro caso, vamos a emplear el sistema para detectar si el usuario está contrayendo o no el músculo. En consecuencia, solo existen dos posibles grupos en los que se tienen que dividir los datos (si hay o no contracción). Por eso, de todos los algoritmos de agrupamiento que hay, solo podemos quedarnos con aquellos en los cuales se pueda elegir el número de *clusters* que tiene que haber.

Son los siguientes:

* *KMeans*
* *Gaussian Mixture Model*
* Agrupamiento jerárquico

## *K-Means*

El algoritmo K-Means [7] selecciona aleatoriamente 2 puntos (porque en nuestro caso tenemos 2 *clusters*) de nuestro set de datos. Estos 2 puntos serán los centroides de nuestros *clusters*. Posteriormente evalúa los demás puntos para asignarlos a un grupo, determinando la distancia más cercana que hay entre ese punto en concreto con cada uno de los *clusters.*

La selección de los *clusters* iniciales es aleatoria, pero el análisis se va a repetir hasta que en análisis consecutivos se obtengan los mismos resultados, lo que significará que el algoritmo ha encontrado una solución. Sin embargo, de momento no se puede decir que la solución sea óptima, por eso se repite el algoritmo varias veces.

## *Gaussian Mixture model*

*Modelos de mezcla gaussiana* [8] es un tipo de algoritmo es muy parecido a *KMeans* pero nos ofrece un poco más de flexibilidad ya que estamos suponiendo que los datos se distribuyen de forma gaussiana (lo cual puede ser muy bueno ya que una de las transformaciones era para hacer que nuestros datos se ajusten mejor a una gaussiana). En cambio, el algoritmo KMeans no permite que su matriz de covarianza sea elíptica, sino circular, por lo que en este caso se ve un poco más limitado. Esto se traduce en que en este caso podemos tener *clusters* de forma elíptica ya que ahora contamos con una desviación estándar tanto en el eje X como en el eje Y.

En este caso, a diferencia del anterior, se va a calcular la probabilidad que tiene cada punto de pertenecer a un *cluster* o a otro. Cuanto más cerca esté el punto del centroide del *cluster*, más probable será que pertenezca a él.

## Agrupamiento jerárquico

En el caso del algoritmo de agrupamiento jerárquico [9], el modelo trata a cada dato como un *cluster* diferente. Posteriormente se van fusionando los *clusters* que están más cerca entre sí, hasta que llegamos al número de *clusters* deseados. Para determinar la distancia entre *clusters* se puede hacer con distintas métricas, pero nosotros utilizaremos la distancia euclídea

Por otra parte, este algoritmo tiene una diferencia fundamental con respecto a los otros dos y es que no incluye la funcionalidad de predecir la clasificación para nuevos datos, por lo que, en la implementación, deberemos incluir también un clasificador, que en este caso será el *KNeighborsClassifier* [10]*,* cuyo funcionamiento es muy similar al de *KMeans* en el sentido de que se basa en la distancia entre *cluster* y datos.

En nuestro caso utilizaremos 2 vecinos ya que seguimos teniendo el mismo problema que antes: si hay contracción o no. Otro parámetro importante para configurar correctamente el algoritmo es determinar el tipo de métrica a seguir, en nuestro caso elegiremos “*minkowski” y p=2* para utilizar la distancia euclídea.

## Estudio del número de *clusters*

Llegados a este punto, es interesante verificar la asunción de que nuestros datos realmente tienen las propiedades necesarias para ser separados exactamente en dos grupos, o si, por el contrario, aunque nosotros estemos forzando el número de *clusters*, el número para un agrupamiento óptimo es otro.

Para visualizar esto, podemos utilizar el algoritmo del codo [11], en el que se valora la inercia obtenida por el método *KMeans*. En este tipo de algoritmo se ve representado como el mayor cambio en la inercia se da cuando tenemos el número óptimo de *clusters*. Por esa forma que toma la línea en el cambio brusco, a este algoritmo se le da el nombre de “algoritmo del codo”.

También podemos hacer un estudio del número óptimo de *clusters* basándonos en el algoritmo de agrupamiento jerárquico, es decir teniendo en cuenta la distancia entre los distintos “*subclusters*”. Esto lo hacemos creando un dendrograma [11].

## Elección de la mejor opción

Llegados a este punto, toca hacer un análisis de cuál de todas las opciones que hemos estudiado nos ofrece un rendimiento óptimo a la hora de agrupar los datos en 2 *clusters*. Para ello, haremos uso del conocido *Davies* *Bouldin* *Score*[12]. Esta puntuación se obtiene a partir de medir el promedio de la separación entre cada *cluster* con su *cluster* más similar.

En este caso, tenemos que estudiar la puntuación de todas las combinaciones de todos los modelos de *clustering*, con la posibilidad de hacer o no reducción de dimensionalidad y con todas las opciones de transformación de variable. De esta manera obtendremos una combinación que nos dará un *Davies Boulding Score* más bajo, y por tanto la que determinaremos como la mejor para implementarla en el juego del dinosaurio.

Para hacer esto, hacemos uso de varios bucles anidados para ir comprobando cada posible combinación para cada muestra de EMG que tenemos. Posteriormente pasamos a promediar las puntuaciones obtenidas de cada muestra con respecto a cada combinación para obtener un resultado general sobre el cual poder sacar conclusiones verídicas.

# Resultados

## *Estudio del número de clusters* por algoritmo del codo

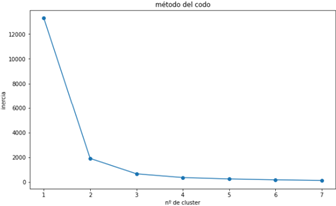


Figura 2: Resultado gráfico del algoritmo del codo.

En este caso se ve como el cambio más brusco está en el número 2, lo que significa que efectivamente es 2 el número óptimo de *clusters* para nuestra muestra.

## *Estudio del número de clusters* por dendrograma

Imagen que contiene Histograma

Descripción generada automáticamente

Figura 3: Resultado gráfico del estudio del dendrograma*.*

En este caso también observamos como la distancia más larga se da cuando tratamos de unir los 2 *clusters* principales, por lo que también es 2 el número óptimo.

## *Resultado del promedio de cada combinación*

Gráfico, Histograma

Descripción generada automáticamente

Figura 4: Distintas combinaciones de métodos de preprocesado de datos y *clustering*.

# Discusión

Como podemos observar, la puntuación óptima es la obtenida aplicando el logaritmo, la reducción de dimensionalidad y el *Gaussian* *Mixture* *model.* Este resultado tiene mucho sentido ya que, como explicamos anteriormente, el Gaussian Mixture model asume que los datos están gaussianizados, y en este caso es así ya que previamente han sido transformados con el logaritmo. Además, se puede observar como en todos los casos, la reducción de dimensionalidad ofrece una mejora en el rendimiento del clustering.

Es importante mencionar que realmente las puntuaciones obtenidas tienen un pequeño grado de aleatoriedad debido a que dependen de la partición que se haya hecho de los datos de prueba y de entrenamiento y esta es aleatoria. Esto significa que haciendo más pruebas nos podría salir que es mejor opción usar otro algoritmo de clustering debido a que la diferencia es muy pequeña.

# Conclusión

Teniendo en cuenta todo lo explicado previamente, vamos a acabar eligiendo la opción de logaritmo, reducción de dimensionalidad y *Gaussian* *Mixture* *model* debido a que, aunque se podría dar el caso de que no fuese la óptima, estos datos se han obtenido a partir de varias muestras de varias personas diferentes, por lo que los vamos a considerar como los óptimos. Es importante tener en cuenta también que, aunque se diese el caso de que esta configuración no fuese la óptima, la diferencia es tan pequeña que es asumible.

# Implementación del modelo en el juego T-Rex de Google

Dónde lo meto??

# Referencias

1. Guerreiro, J., Martins, R., Silva, H., Lourenço, A., & Fred, A. L. (2013, July). BITalino-A multimodal platform for physiological computing. In ICINCO (1) (pp. 500-506).
2. biosignalsplux | OpenSignals. Biosignalsplux.com. Published 2020. Accedió en septiembre 21, 2021.

https://biosignalsplux.com/products/software/opensignals.html

1. sklearn.preprocessing.minmax\_scale. scikit-learn. Published 2021. Accedió en noviembre 9, 2021. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.minmax\_scale.html
2. ‌ Pita Fernández, S., & Pértega Díaz, S. (1997). Relación entre variables cuantitativas. Cad Aten Primaria, 4, 141-4. Accedió en septiembre 19, 2021.
3. Kiko Correoso. Análisis de componentes principales con python – Pybonacci. Pybonacci.org. Published October 8, 2012. Accedió en octubre 16, 2021. https://pybonacci.org/2012/10/08/analisis-de-componentes-principales-con-python/
4. sklearn.model\_selection.train\_test\_split. scikit-learn. Published 2021. Accedió en octubre 19, 2021. https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.train\_test\_split.html
5. Cambronero, C. G., & Moreno, I. G. (2006). Algoritmos de aprendizaje: knn & kmeans. Intelgencia en Redes de Comunicación, Universidad Carlos III de Madrid, 23. Accedió en septiembre 25, 2021.
6. Instituto de Informatica UACh. [INFO337] Modelos de mezcla de Gaussianas parte 1. YouTube. Published online December 10, 2020. Accedió en octubre 15, 2021. ttps://www.youtube.com/watch?v=s3sTSA\_GXV8&t=5106s&ab\_channel=InstitutodeInformaticaUACh
7. Ligdi González. Algoritmo Agrupamiento Jerárquico - Práctica - Aprende IA. Aprende IA. Published November 10, 2020. Accessed November 21, 2021. https://aprendeia.com/algoritmo-agrupamiento-jerarquico-practica/. Accedió en octubre 1, 2021.
8. sklearn.neighbors.KNeighborsClassifier. scikit-learn. Published 2021. Accedió en noviembre 5, 2021. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
9. ‌‌Villardón, J. L. V. (2007). Introducción al análisis de clúster. Departamento de Estadística, Universidad de Salamanca. 22p. Accedió en septiembre 25, 2021.
10. sklearn.metrics.davies\_bouldin\_score. scikit-learn. Published 2021. Accessed November 21, 2021. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.davies\_bouldin\_score.html#. Accedió en septiembre 30, 2021.