Máster de Ingeniería de Telecomunicación
Plan 2017

*Trabajo Fin de Máster*

# "Análisis de métodos de aprendizaje automático para la clasificación de señales EEG de pacientes con epilepsia"

---


Dimas Álvarez Mijares

Tutor/es

Jordi Solé Casals

David García Vizcaino


FECHA: Enero 2020

# ÍNDICE

# Índice de Figuras



## 1. Resumen

La epilepsia es un trastorno neurológico que afecta a más de 700.000 personas en España y varios millones considerando todo el planeta. Puede llegar a ser peligrosa para los que

la sufren e incluso letal en algunos casos si no se trata a tiempo. Es por ello que, como sucede en la mayoría de las enfermedades, el diagnóstico temprano permite incrementar la efectividad de los tratamientos.

Para ayudar en la detección de la enfermedad, en este trabajo se pretende diseñar un programa capaz de determinar si una señal proveniente de un encefalograma (EEG) se corresponde con una de un paciente con epilepsia o un paciente sano. Para ello se analizarán diversos métodos basados en el aprendizaje automático y se realizarán pruebas con cada uno de ellos.

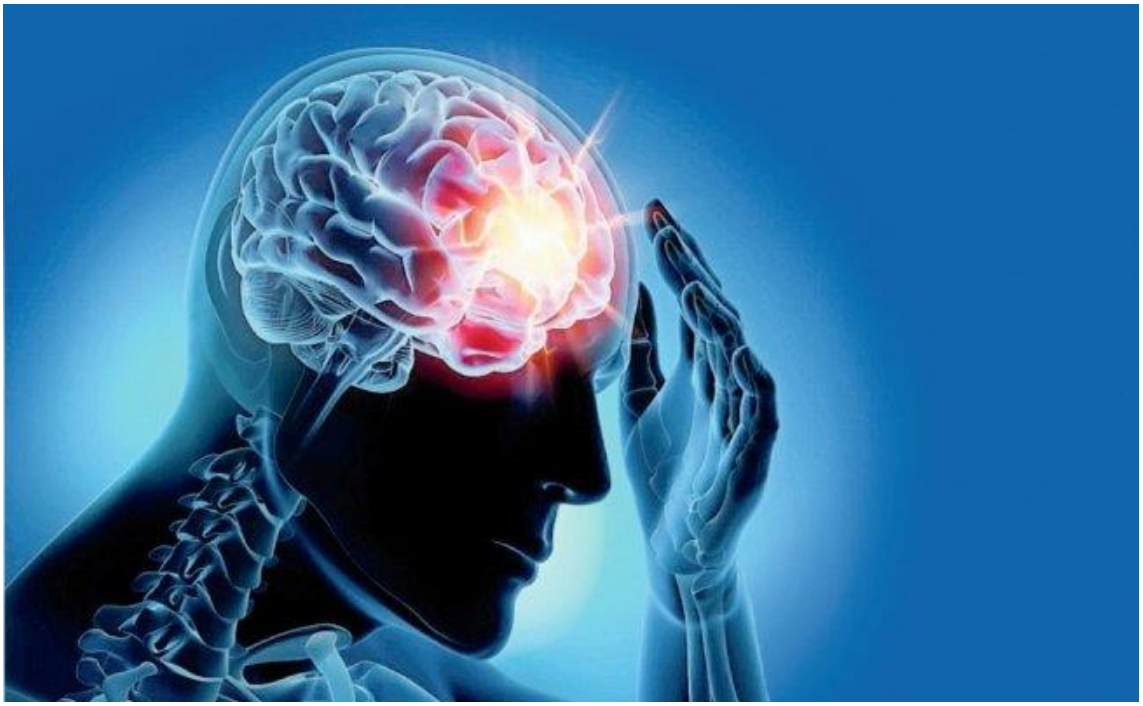

Figura 1 Representación artística de la epilepsia.

# 2. Introducción

## 2.1. Motivación del trabajo.

Los avances en computación y nuevas tecnologías no pueden resolver los problemas de las personas por sí solos, se deben de crear aplicaciones concretas basadas en estos avances y así mejorar la vida de los demás. En este trabajo la idea principal es aprovechar las ventajas del aprendizaje automático, también conocido como Machine Learning y relacionarlo con el mundo de la medicina para el diagnóstico de enfermedades.

En este caso se ha decidido optar por una enfermedad como la epilepsia por la disponibilidad de un conjunto muy amplio de señales que permitirán entrenar el algoritmo de aprendizaje que se detallará más adelante. Este conjunto deseñales también ha sido objeto de estudio en varios de los trabajos que se mencionarán durante el desarrollo de este documento.

Por otra parte, las aplicaciones de los algoritmos de Machine Learning son un tema de actualidad y sus posibilidades son muy elevadas, por lo que la oportunidad de explotar sus puntos fuertes e investigar sobre su viabilidad en el campo de la medicina es realmente interesante. Y dentro de este tipo de algoritmos se pueden encontrar los algoritmos Deep Learning o redes neuronales, que se pueden entender como una rama dentro del Machine Learning y cuyas características podrían ser más efectivas para este trabajo.

## 2.2. Objetivos.

Desarrollo de una herramienta capaz de clasificar (con una tasa de acierto lo más elevada posible) las señales de EEG en dos grupos, las que provienen de pacientes con epilepsia y las que provienen de pacientes sanos.

Estudio del algoritmo de Deep Learning LSTM. Se revisará su viabilidad para este tipo de señales y forma de clasificarlas.

Estudio de otros trabajos y documentación relacionada con algoritmos de Deep Learning para decidir sobre cuál es el más adecuado. Además, se probarán variantes sobre LSTM que permitan mejorar los resultados obtenidos.

Comparación de otros métodos basados en Machine Learning para escoger el más adecuado y con mayor tasa de acierto.

Establecer puntos de mejora y pautas para seguir en el futuro con vistas a aumentar la tasa de acierto del algoritmo.

# 3. Desarrollo

## 3.1. Punto de partida.

Para la realización de este trabajo se cuenta, en primer lugar, con una base datos de EEG de pacientes con epilepsia y un paper en el cual se realiza un análisis de esa base de datos [1]. La base de datos está formada por 7500 archivos de texto ".txt" que representan las señales de los EEG.

La mitad de las señales de la base de datos se corresponden con pacientes que sufren epilepsia (Focales) y la otra mitad con pacientes sanos (No focales). Cada archivo contiene dos columnas que representan dos canales distintos de cada señal focal o no focal con 10240 muestras cada uno, siendo un canal una medición de EEG con un sensor determinado.

A partir de los datos extraídos de [1], sabemos que existen diferencias estadísticas entre las señales focales y las no focales. Las dos conclusiones principales de los test realizados son las siguientes:

- Existe un mayor rechazo en los test de aleatoriedad e independencia no lineal para las señales focales.
- En las señales no focales existe un mayor rechazo para los test de estacionariedad.

Sin embargo, a simple vista no se puede encontrar un patrón que permita clasificarlas. A continuación, se muestra un ejemplo de 3 pares de señales focales y no focales sin procesar.

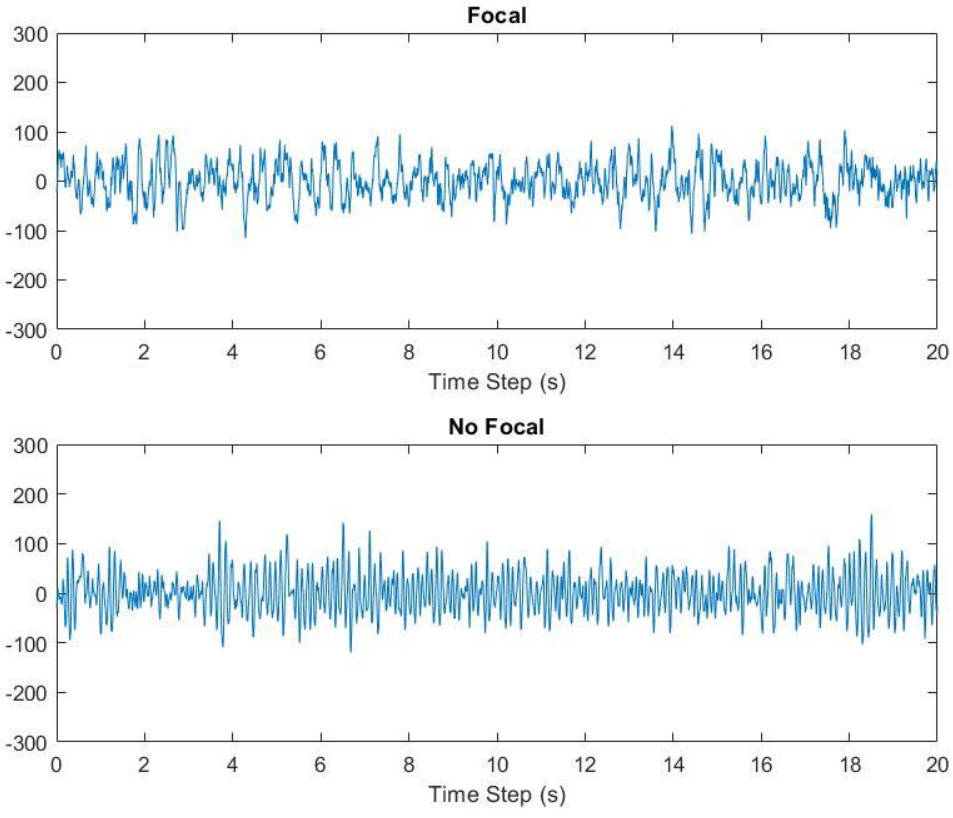

Figura 2 Señales Data_F_Ind0001 (Focal) y Data_N_Ind0001 (No Focal).

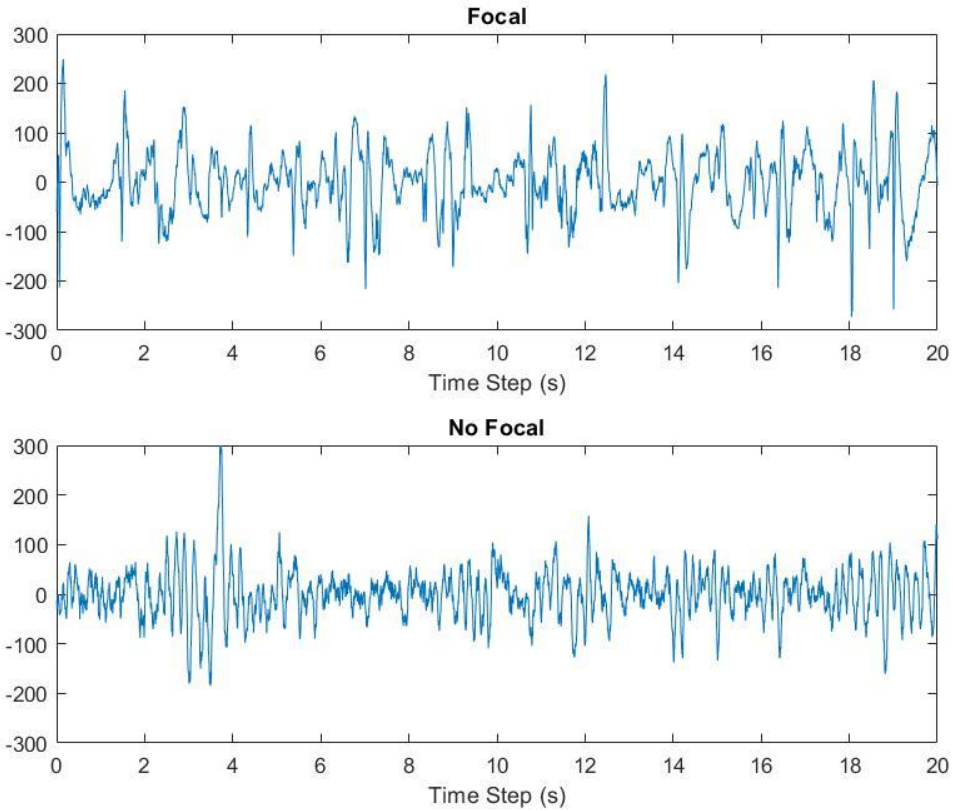


Figura 3 Señales Data_F_Ind0010 (Focal) y Data_N_Ind0010 (No Focal).

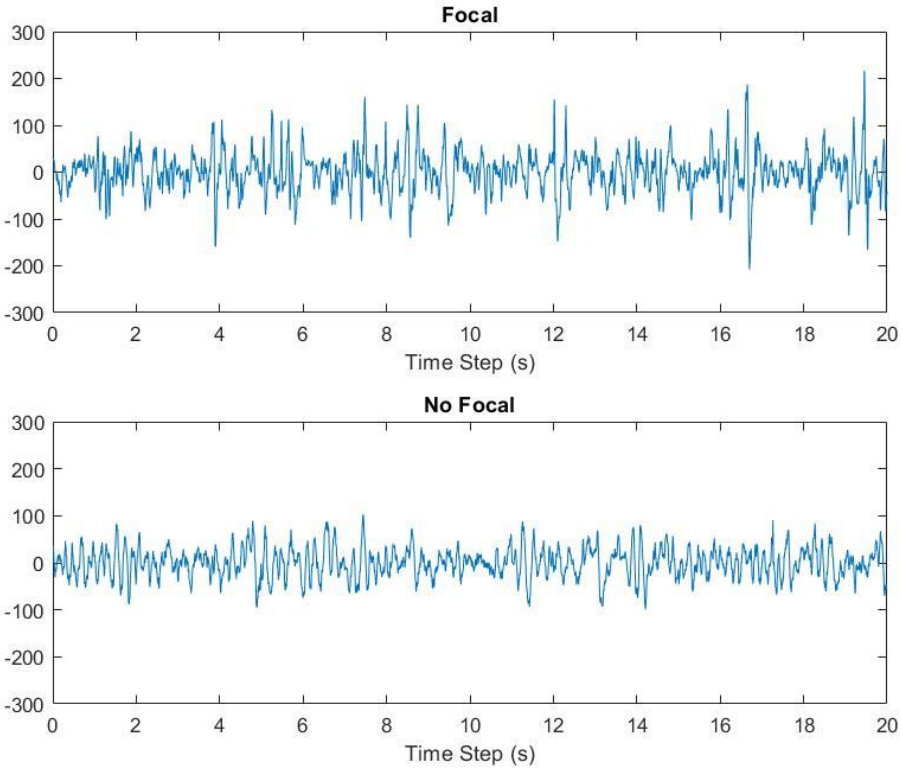


Figura 4 Señales Data_F_Ind0100 (Focal) y Data_N_Ind0100 (No Focal).

### 3.2. Primeros pasos con la base de datos.

Una vez que se tiene claro el conjunto de datos de partida, lo primero que hace falta para poder ejecutar el algoritmo de Deep learning es preparar los datos para su lectura con el software Matlab, que es el que se utilizará durante todas las pruebas realizadas en este trabajo.

Para ello colocan los 7500 archivos en un mismo directorio y, aprovechando ciertas partes del código que se utilizó en [1] para realizar la carga de datos, se prepara un código que permite cargar el número de señales de cada tipo que se desee y marcar el tipo de cada señal con una “F” o “N” en función de si son Focales o No focales respectivamente.

Este etiquetado o “labeling” en inglés, es de vital importancia en los algoritmos de aprendizaje automático supervisado, es decir, cuando a la hora de entrenar el algoritmo se le indica de qué clase es cada registro de entrada. Se debe tener especial cuidado en evitar que se mezclen las etiquetas al manejar los datos o podría provocar que el algoritmo no funcione correctamente.

En nuestro caso, el código utilizado genera una variable de tipo “cell” en la que cada celda incluye una matriz de los datos leídos de cada archivo, de forma que se tengan dos filas en cada matriz (una por cada canal) de 10240 muestras cada una. De igual manera, se genera a la vez un vector “Categorical” con la etiqueta correspondiente a cada señal.

El algoritmo LSTM o Long Short Term Memory como indican sus siglas se trata de un algoritmo de Deep learning basado en redes neuronales recurrentes. La principal ventaja

de este tipo de algoritmos es que no solo procesan datos en un determinado momento sino las secuencias completas.

El concepto que se pretende comprobar si es viable o no en este trabajo es el de utilizar las secuencias completas de cada señal y su etiqueta como parámetros de entrada de la red LSTM como datos de entrenamiento. Es decir, introducir las señales tal cual las leemos de los archivos sin calcular características (también llamadas "features" en inglés) y que sea la propia red quien se adapte a la señal para clasificarlas correctamente.

### 3.3. Pruebas con el algoritmo LSTM.

A partir de este punto, en este proyecto se realizaron multitud de pruebas con la base de datos para tratar de entrenar la red LSTM, utilizando un método de ensayo y error hasta ir puliendo los errores típicos en este tipo de desarrollos. Por tanto, se procederá a explicar los pasos que se han ido siguiendo y mejoras varias hasta llegar a las conclusiones de este apartado.

En primer lugar, para realizar el entrenamiento del algoritmo se deben escoger las señales que se utilizarán y cuáles se van a dejar para test. En nuestro caso dividimos el conjunto de señales en un 90% de las mismas para entrenamiento y un 10 % para test, utilizando un solo canal para simplificar el problema ya que la información de ambos canales es muy similar entre ellos. Además, para agilizar las pruebas no siempre se utilizará el conjunto entero de señales, para este desarrollo se utilizaron 2000 señales hasta asegurar que no había errores en el código.

En la primera prueba se utilizó un modelo de LSTM similar al que se puede encontrar en [2]. Se trata de un ejemplo de uso de este tipo de redes neuronales para clasificar señales de Electrocardiogramas (ECG). En el mismo se llegó a alcanzar una tasa de acierto cercana al 56% aplicando el algoritmo con las secuencias completas y sin cálculo de características, por lo tanto, sirve para tener una primera idea de los resultados que se pueden llegar a obtener en nuestras pruebas.

La arquitectura de un algoritmo de Deep learning se basa en el conjunto de datos de entrada, las capas (también conocidas como "Layers" en inglés) y el conjunto de salida. Matlab dispone de varias herramientas muy útiles para la definición de estos algoritmos, una de ellas es el "Deep Network Designer" que permite elaborar de manera un poco más intuitiva la arquitectura de la red. Mediante esta herramienta se puede representar gráficamente en que consiste esta primera arquitectura que se probó con nuestros datos, ver Figura 5Figura 4.

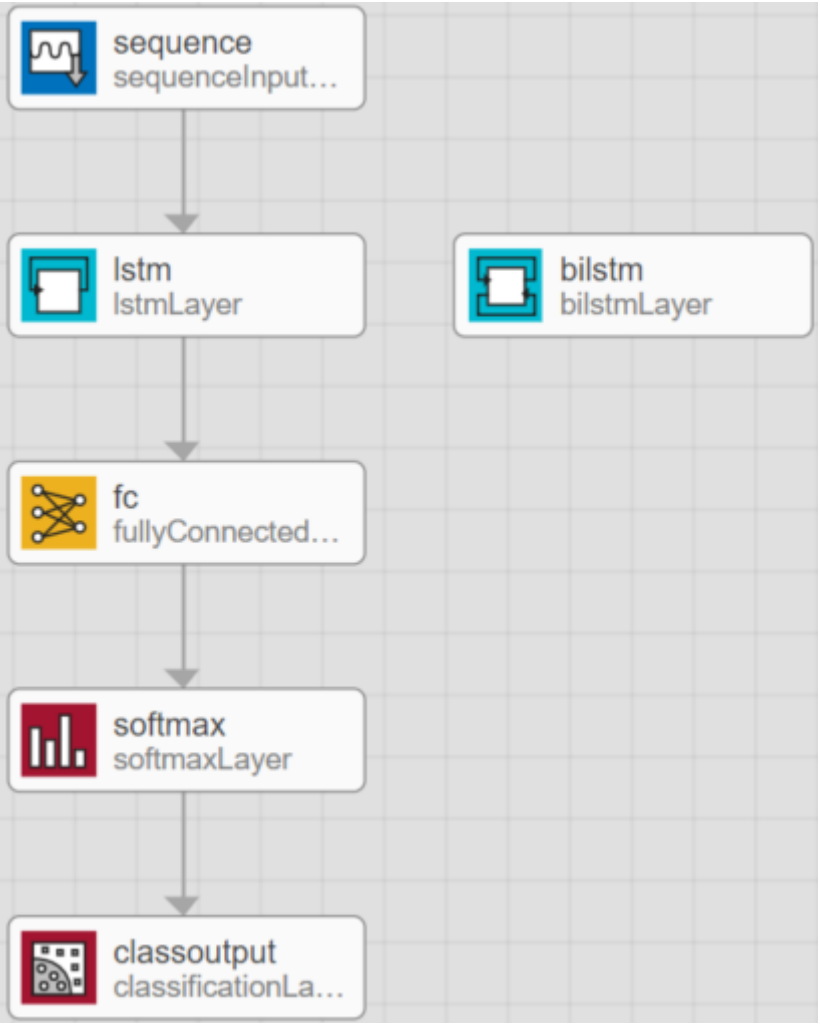


Figura 5 Arquitectura de la red neuronal con LSTM o BILSTM.

En la imagen se puede ver también la inclusión de la capa BILSTM. Esta funciona de manera similar al LSTM, pero de forma bidireccional, es decir, mientras LSTM solo examina la secuencia hacia delante, BILSTM también la examina hacia atrás.

En la figura 3 también hay que destacar la importancia de la capa "FullyConected". En un algoritmo de Deep learning es el propio algoritmo el encargado de calcular las características de la señal para terminar, en nuestro caso, clasificándola en uno de nuestros dos grupos. El número de características que se van a calcular se puede definir en la arquitectura de la capa LSTM eligiendo un número de "HiddenUnits" y es el "FullyConected" el encargado de unir esta capa con las dos últimas que sirven para hacer la clasificación en los dos grupos definidos.

Antes de mostrar el resultado de las ejecuciones del algoritmo, conviene dar los últimos detalles sobre la variedad de configuraciones posibles del algoritmo en relación con lo que se ha podido probar en el desarrollo del trabajo.

- Escoger un número de HiddenUnits elevado aumenta la complejidad del algoritmo y normalmente mejora la tasa de acierto, sin embargo, también provoca que se tarde mucho más en completar cada iteración. En nuestras pruebas se usaron valores de entre 10 y 150 unidades.

- LSTM permite definir un “ventaneado” de la señal para ahorrar memoria en los cálculos de modo que solo se opera con partes de la secuencia. Para nuestro caso se utilizaron secuencias de 1000 elementos. Si no se utiliza esta opción, la memoria no es capaz de procesar las secuencias de 10240 muestras.
- El número de iteraciones debe ser el suficiente para que el algoritmo sea capaz de clasificar tanto las señales de entrenamiento como las de test. En los problemas de machine learning en general puede pasar que si se itera demasiado el programa termina por sobreajustarse a las señales de entrenamiento y el error aumenta cuando se procesan las señales de test. Este problema se conoce como “Overfitting” en inglés.
- Existen otra variedad de parámetros como el método de gradiente que se utiliza, el tamaño del Batch, el ritmo inicial de aprendizaje, entre otros. Ajustar cada uno de ellos para obtener la mayor eficiencia no es algo trivial y a menudo si alguno de ellos no está adaptado al problema en cuestión puede provocar que el algoritmo no funcione como se espera.

Con todo esto en cuenta, se muestra a continuación el resultado de entrenar la red con una de las varias configuraciones que se utilizaron en las pruebas. Solo se muestra un caso porque en el resto apenas se encontraron diferencias.

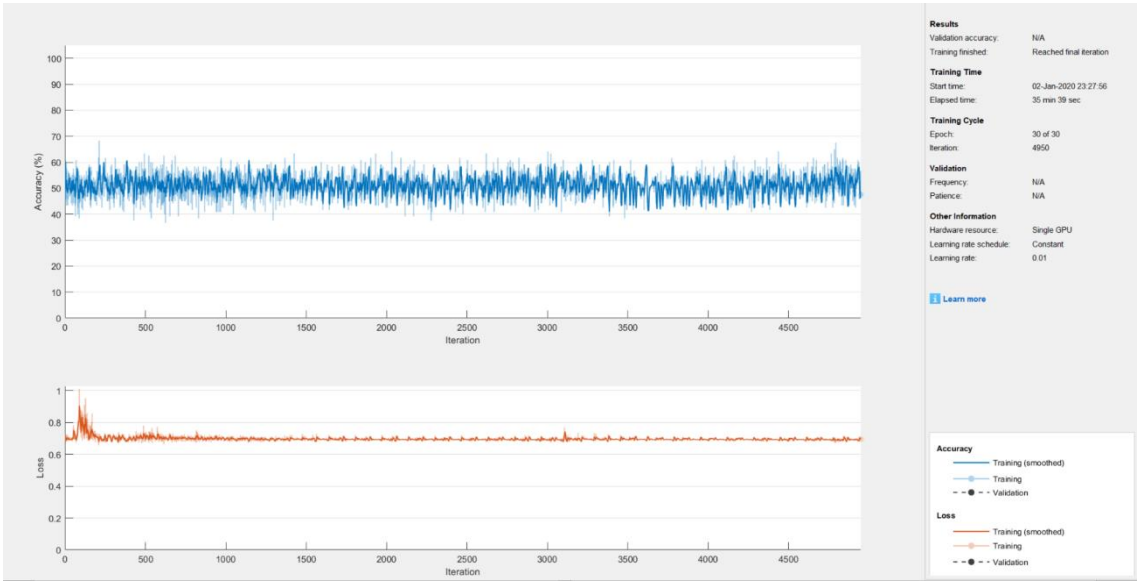


Figura 6 Evolución de la precisión en el grupo de entrenamiento. Pruebas con LSTM simple.

Como se puede ver en la Figura 6, la tasa de acierto se mantiene alrededor del 50%, lo cual indica que el algoritmo no es capaz de encontrar una solución al problema y no sabe clasificar las señales adecuadamente. Concretamente, al no converger en una solución, termina por decidir que todas las señales pertenecen al mismo grupo. Lamentablemente, en todas las pruebas realizadas siempre se ha llegado a esta solución sin importar la configuración de la red LSTM.

Otra de las opciones posibles para intentar que el algoritmo aprenda a clasificar las señales es aumentar la complejidad del mismo añadiendo más capas de LSTM. Sin embargo, la única diferencia está en el tiempo que tarda en terminar el número de iteraciones especificado. En la Figura 7 se puede ver una arquitectura más elaborada para la red en

la que se mezclan LSTM y BILSTM además de una capa “dropout” que elimina secuencias aleatoriamente que puede ayudar a mejorar el entrenamiento.

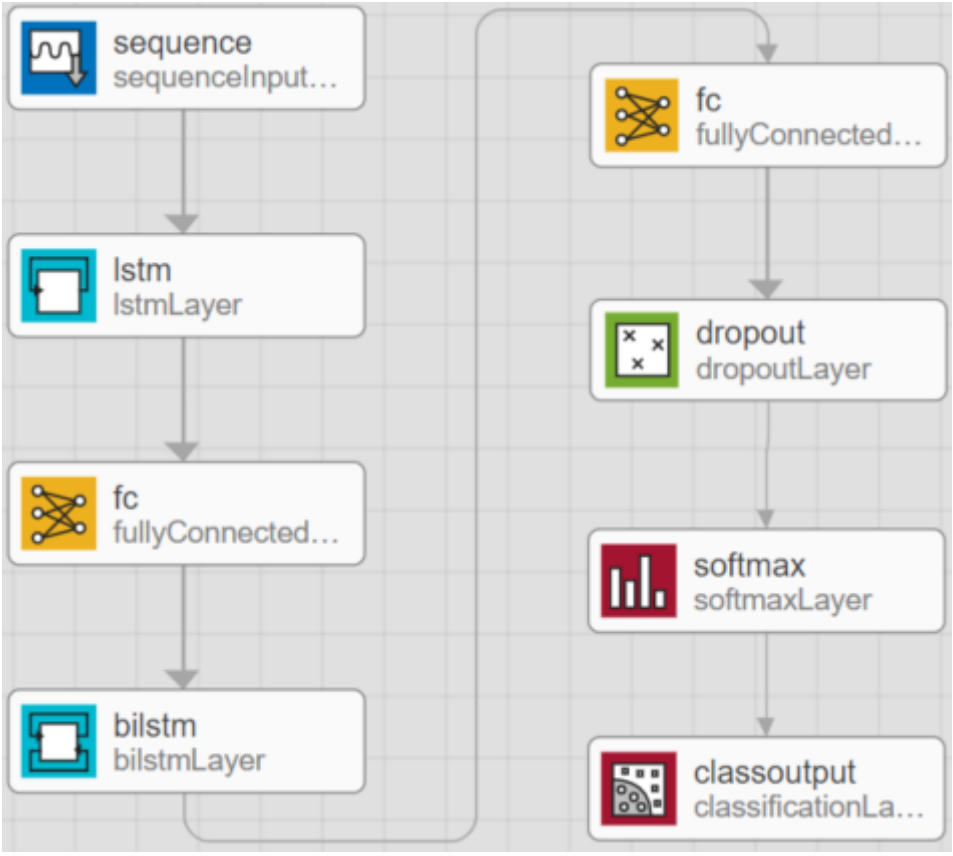


Figura 7 Arquitectura de la red neuronal más compleja.

Con esta y otras configuraciones tampoco se mejoraron los resultados, por lo que el siguiente punto del desarrollo pasa por realizar modificaciones en los datos de partida.

### 3.4. Métodos de optimización de la red neuronal.

Partiendo de las pruebas anteriores, el siguiente paso es buscar una forma de el algoritmo sea capaz de aprender a clasificar las señales. Una forma de conseguir esto es mediante la normalización de las señales [3].

Para realizar esto se puede calcular la media y la desviación estándar del conjunto de señales de entrenamiento y realizar la siguiente operación antes de utilizarlas como entrada en el algoritmo:

$$S\_nor = \frac{S - \mu_s}{\sigma_s}$$

Otra opción es mediante la función “zscore” de Matlab, que aplica la normalización a la señal y además devuelve la media y desviación. Estos valores calculados se guardan para utilizarlos de nuevo con las señales de test antes de utilizar la red entrenada para clasificarlas.

Esta operación de normalización se utilizó en las pruebas con nuestro conjunto de datos, pero no cambiaron los resultados, aunque se trata de una buena práctica que se debe seguir utilizando en el resto de las pruebas antes de utilizar LSTM.

Por otra parte, en [2] también se ve otra buena práctica para aplicar que se trata de ordenar de manera aleatoria los datos antes de introducirlos en el algoritmo. De igual manera, también conviene escoger de manera aleatoria cuales se dejan para entrenamiento y cuales son para test, teniendo en cuenta que exista un número similar de señales de cada tipo (Focales y No focales) en ambos conjuntos.

Otra opción que se ha probado para este conjunto de datos es realizar un filtrado previo para reducir la magnitud de componentes de alta frecuencia que afecten a la clasificación de las señales. Este paso tampoco generó mejoras en estas pruebas, pero tendrá más peso a la hora de utilizar el cálculo de características de la señal que se verá más adelante.

En este punto y solo a modo de comprobación de que no se están cometiendo errores durante el etiquetado de las señales, entre otros posibles errores, se decide hacer una prueba que puede parecer trivial, pero sirve a modo de comprobación. Dicha prueba consiste en modificar las señales de uno de los tipos de modo que sea diferenciables a simple vista de las del otro tipo y probar de nuevo el algoritmo.

Para este caso se prepararon las señales de igual manera que en el resto de las pruebas y teniendo en cuenta estos últimos conceptos, aunque, antes de introducir las señales en el algoritmo, se multiplicó por un valor fijo cada una de las muestras de cada señal del tipo Focal. En la Figura 8 se muestra cómo se configuraría la red para esta prueba mediante el código de Matlab directamente.

```
numHiddenUnits = 100;
numClasses = 2;

layers = [ ...
    sequenceInputLayer(1)
    bilstmLayer(numHiddenUnits,'OutputMode','last')
    dropoutLayer
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];

options = trainingOptions('adam',...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize', MiniBatchSize, ...
    'InitialLearnRate', 0.01, ...
    'SequenceLength', 1000,...
    'GradientThreshold', 1, ...
    'ExecutionEnvironment',"auto",...
    'plots','training-progress', ...
    'Verbose',false);


net = trainNetwork(XTrain_Nor_Trunc,YTrain,layers,options);
trainPred = classify(net,XTrain_Nor_Trunc,'SequenceLength', 1000);
LSTMAccuracy = sum(trainPred == YTrain)/numel(YTrain)*100
```

```
testPred = classify(net,XTest_Nor_Trunc,'SequenceLength', 1000);
LSTMAccuracy_Test = sum(testPred == YTest)/numel(YTest)*100
```

Figura 8 Código de Matlab utilizado para configurar la capa LSTM.

A continuación, en la Figura 9 se puede ver como ahora los valores de la tasa de acierto han aumentado significativamente. En concreto, los resultados muestran una tasa de acierto de alrededor al 82%. En la Figura 10 y la Figura 11 se puede ver la matriz de confusión de los datos de entrenamiento y de test. Este tipo de gráficas son muy útiles para entender rápidamente el resultado de la ejecución del algoritmo, en ellas se aprecia el número de señales que tras pasar por la red se colocan en un grupo u otro y si fue correcta la clasificación (Verde) o no (Rojo).


Figura 9 Evolución de la precisión en el grupo de entrenamiento. Pruebas con BILSTM y datos alterados para el grupo F.

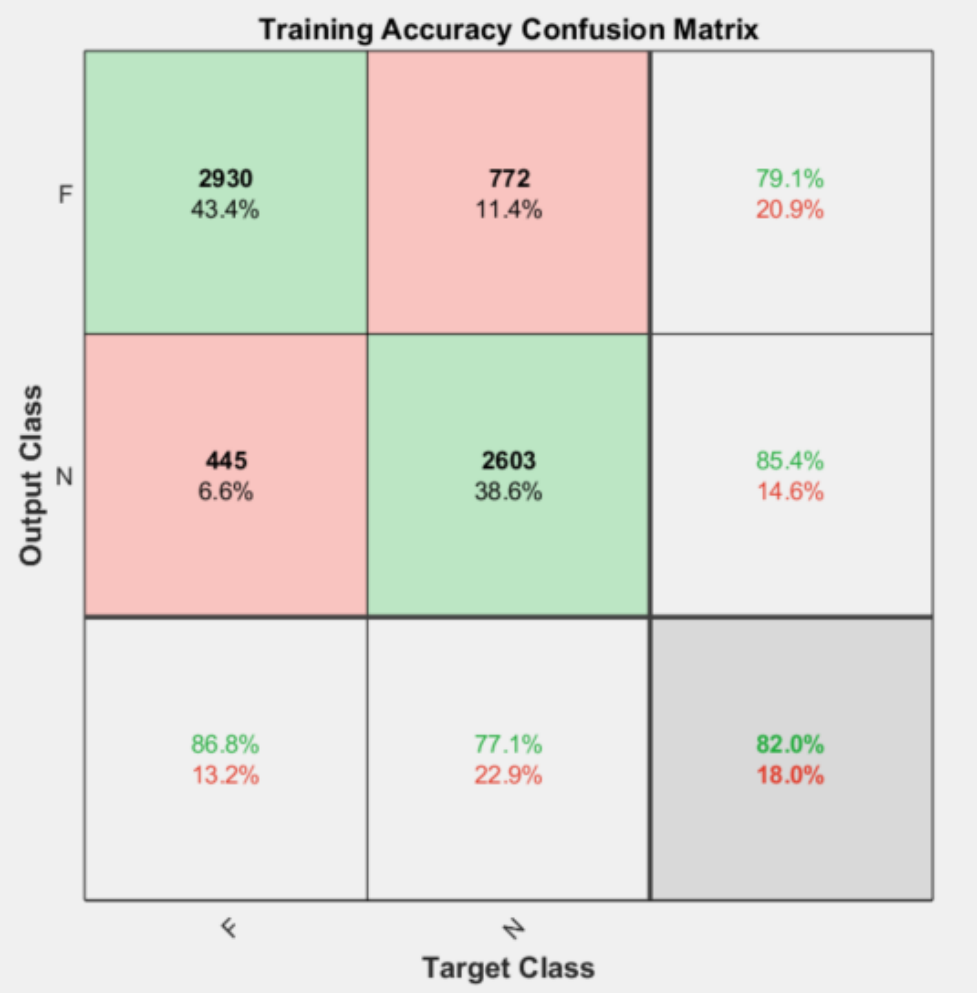


Figura 10 Matriz de confusión de datos de entrenamiento.

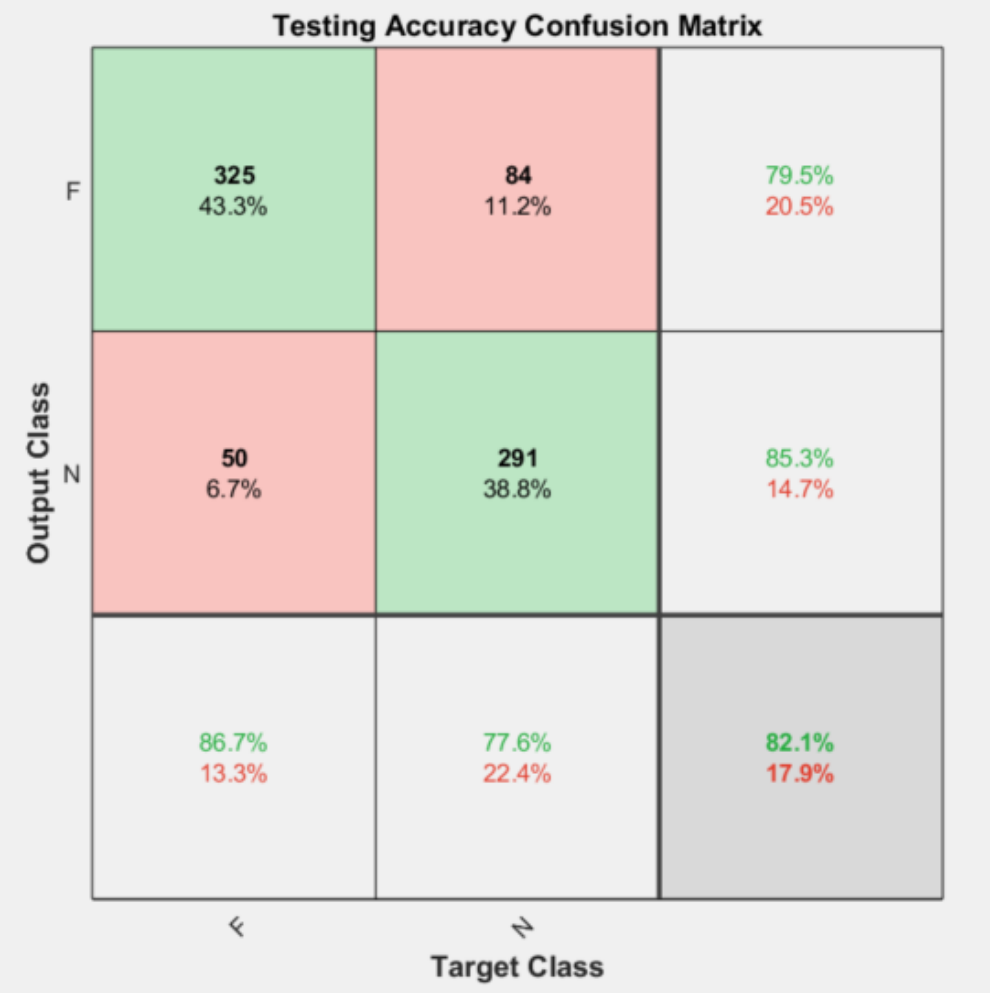


Figura 11 Matriz de confusión de datos de test.

Tras ver estos resultados, queda demostrado que el algoritmo funciona si existen diferencias entre ambos tipos de señales y que no se están cometiendo errores en el manejo de datos. Por tanto, lo que se puede deducir también es que las señales tal cual se leen de los archivos y aunque se les realice un leve procesado no aparentan tener diferencias que el algoritmo LSTM sea capaz de detectar, al menos en las pruebas que se han realizado.

### 3.5. Herramienta "Classification Learner" y cálculo de características.

Ya que hasta ahora no se han conseguido buenos resultados, se procede a probar otra herramienta de las disponibles en Matlab. Se trata de Classification Learner, una herramienta que permite probar multitud de algoritmos de Machine Learning sobre un conjunto de datos de entrada.

Para utilizarlo se deben preparar los datos tal y como se muestra en los ejemplos que se facilitan en la documentación de la herramienta [4]. Una vez los datos están preparados de la manera adecuada, se puede abrir la herramienta y cargar los datos. En la Figura 12 se puede ver como luce la herramienta en una de las pruebas que se realizaron.

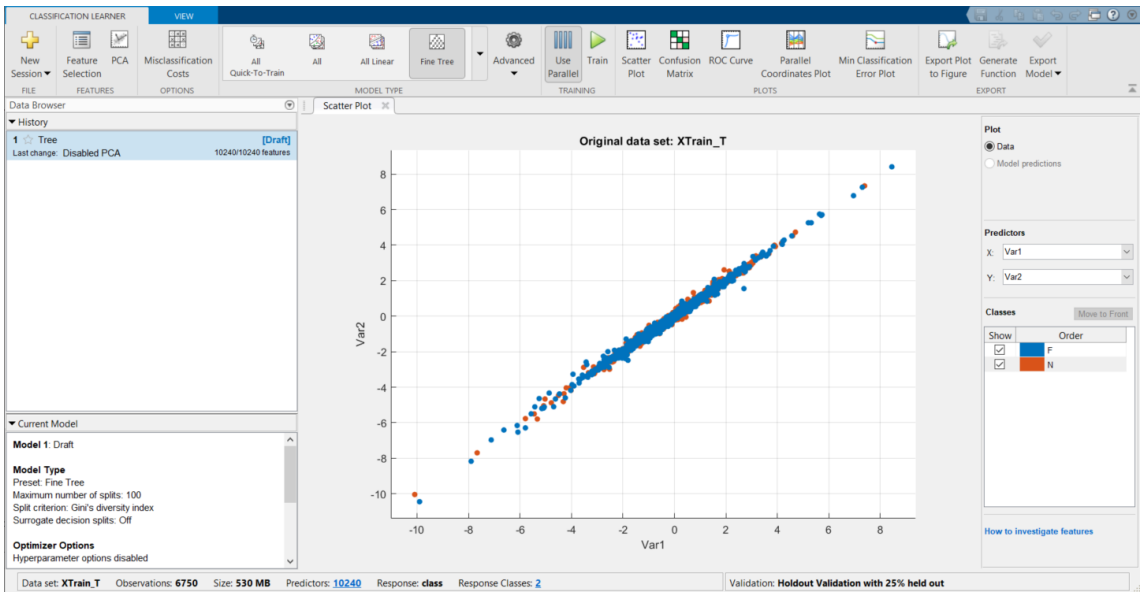


Figura 12 Interfaz de la herramienta Classification Learner.

A continuación, en la Figura 13 se pueden ver algunos de los distintos tipos de modelos de clasificación que se pueden utilizar. La herramienta dispone de una opción realmente útil para probar varios de los modelos en orden y además aprovecha a computación en paralelo de Matlab para mejorar el rendimiento y reducir los tiempos.

En nuestras pruebas se ha decidido utilizar el conjunto entero de señales, pero truncándolas para las primeras 500 muestras en lugar de la secuencia completa, de esta manera se reducen los tiempos de entrenamiento y se pueden probar todos los modelos. Cabe destacar que se probó a realizar una ejecución con uno de los modelos y utilizando las 10240 muestras, pero los resultados no fueron muy distintos de los que se pueden ver en la Figura 13.

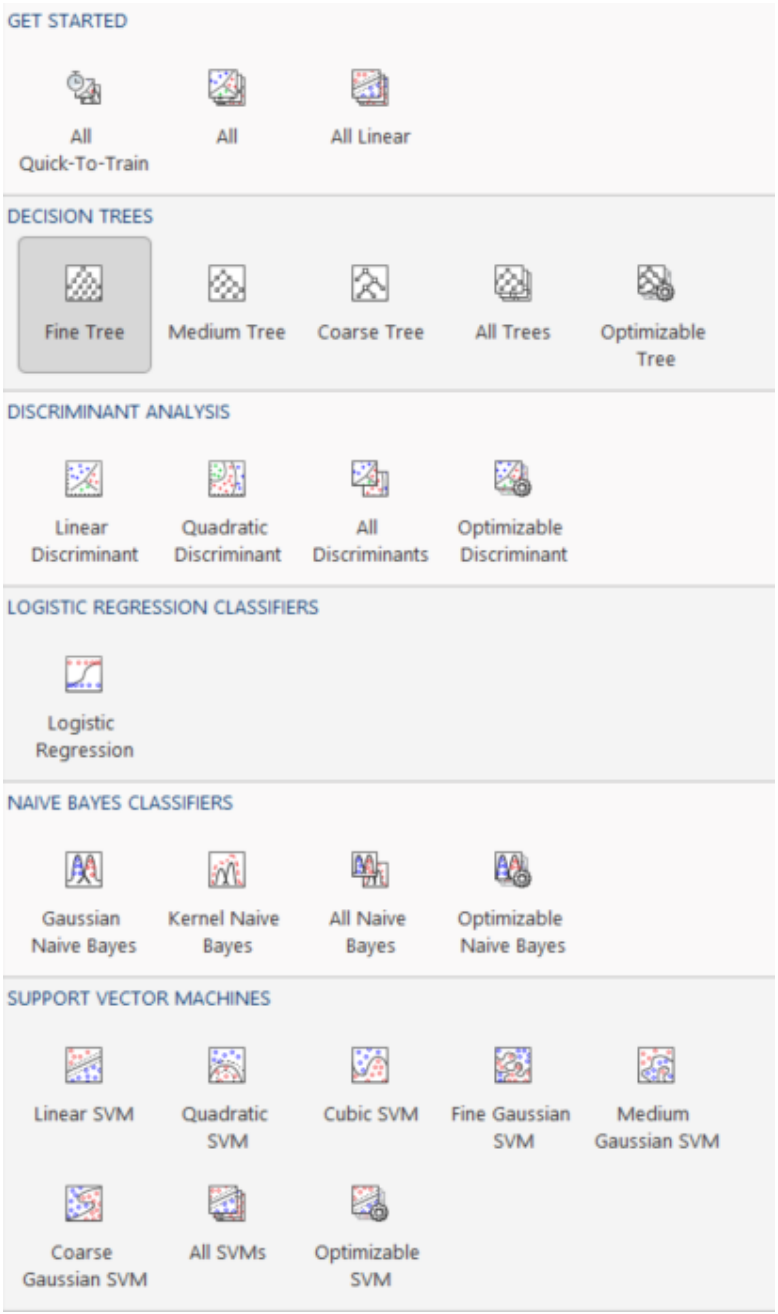


Figura 13 Interfaz de elección del modelo de entrenamiento.

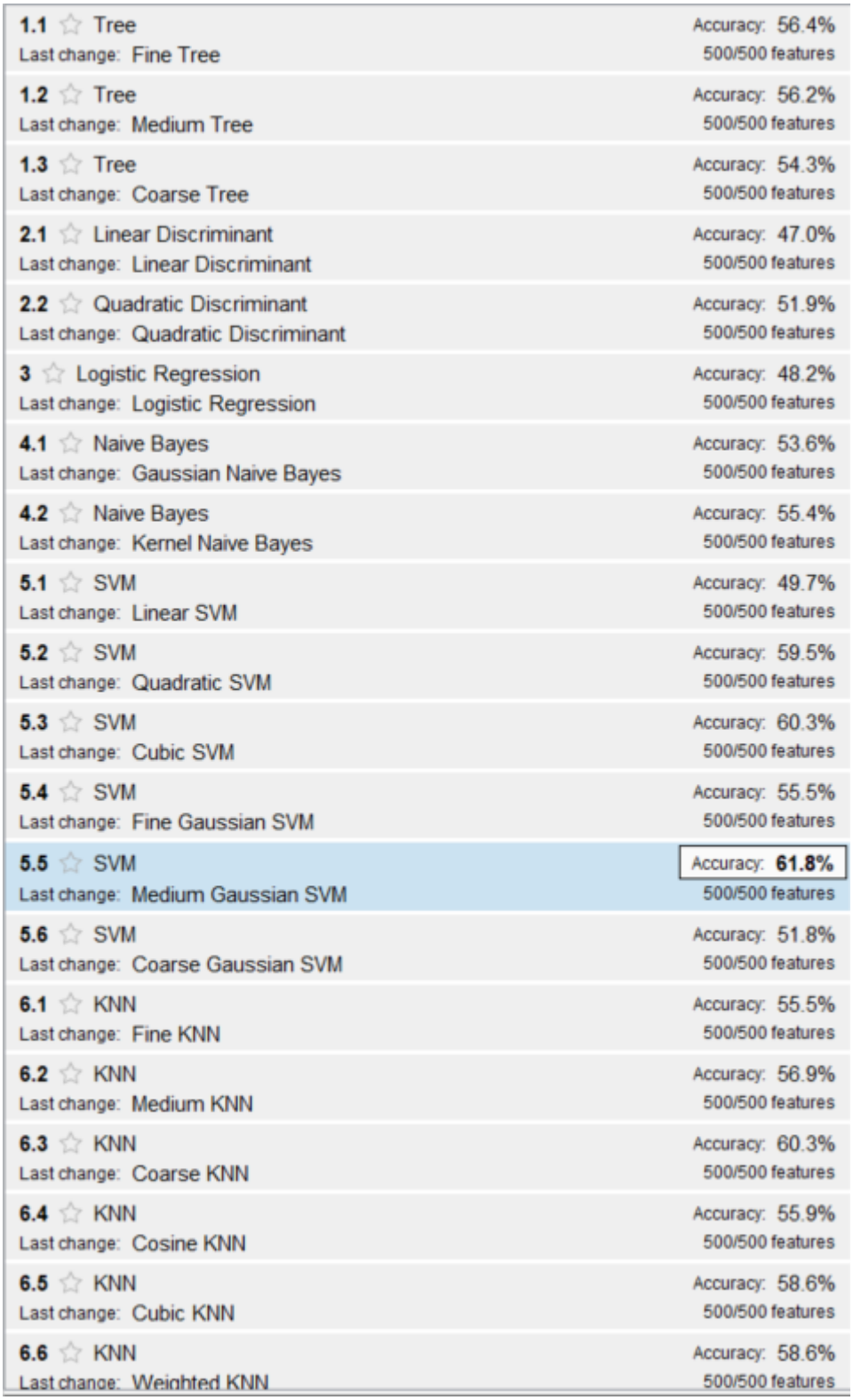


Figura 14 Resultados del entrenamiento con los distintos modelos.

A la vista de estos resultados se puede decir que uno de los mejores modelos que pueden utilizar con este conjunto de señales es el "Medium Gaussian SVM". No se va a entrar en detalles del funcionamiento de SVM, pero sí se comentará que se trata de un algoritmo en el que se separan las clases a dos espacios lo más amplios posibles mediante hiperplanos de separación [5]. Por otra parte, también se puede destacar el artículo de [6], en él se puede encontrar más información relevante sobre la utilidad de estos algoritmos.

Los datos de ese entrenamiento se pueden exportar para utilizarlos con los datos de test y así comprobar que se obtiene una precisión parecida.

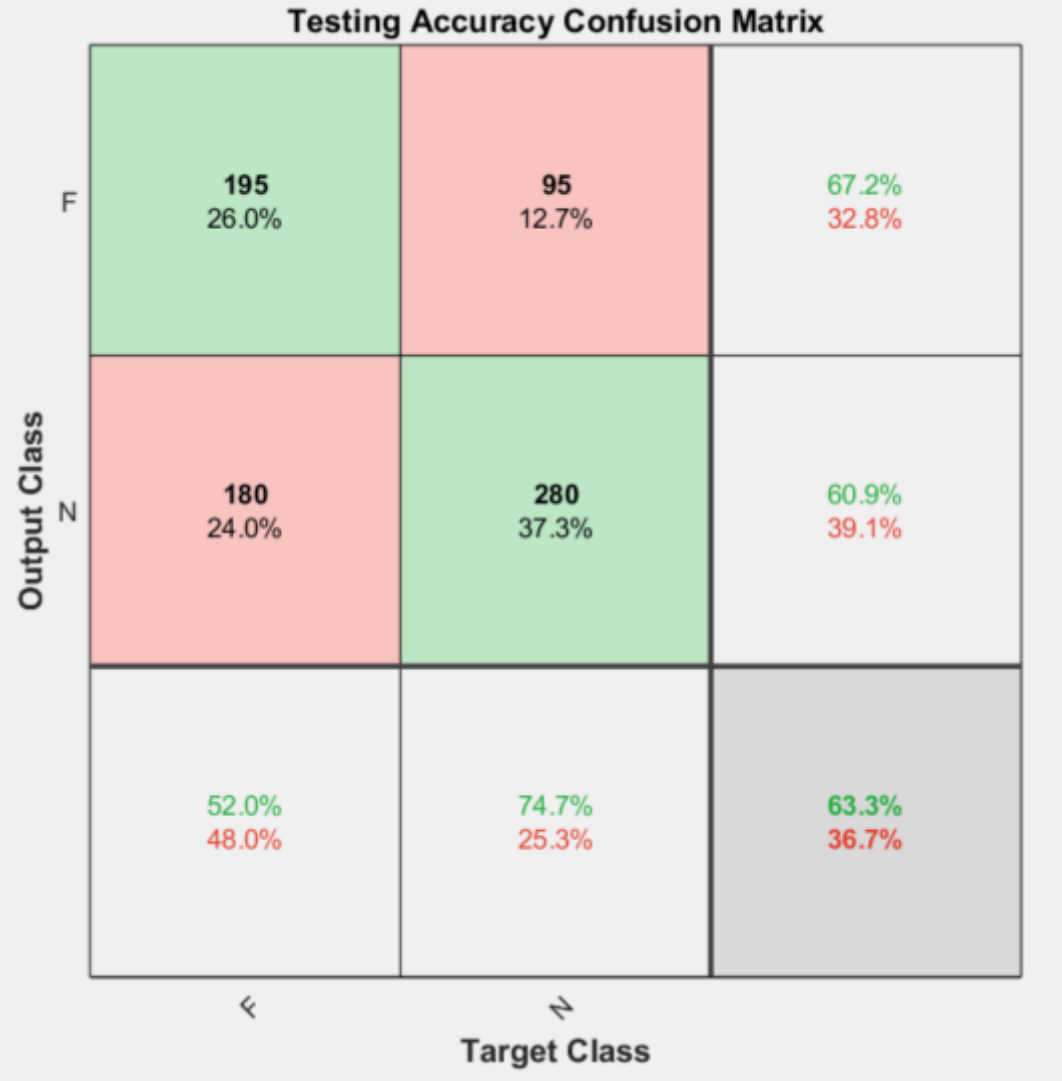


Figura 15 Matriz de confusión de datos de test. Medium Gaussian SVM

En la Figura 14 se puede ver la matriz de confusión con los datos de test. Aunque no se tratan de unos resultados óptimos, un 63 % de acierto es una gran mejoría respecto a lo visto hasta ahora en este trabajo.

Como parece que sería difícil conseguir mejoras por este camino, se ha decidido probar a realizar un cálculo de características sobre cada señal utilizando todas las muestras. Para ello, basándose en el artículo de [7] se utilizó una función facilitada por el tutor que calcula una serie de valores (características) que definen cada señal.

En el artículo se pueden ver las características que se calculan para mejorar los resultados de los algoritmos de machine learning que utilizan. En la Figura 16 se pueden ver las características temporales y en la Figura 17 se pueden ver las frecuenciales.

| Temporal Features | Descriptor |
|---|---|
| Sample entropy (SENT) | $m = 3, r = 0.2$ |
| Mean absolute value (MAV) | $\frac{1}{N}\sum_{i=1}^{N}\lvert X_i\rvert$ |
| Variance (VAR) | $\frac{1}{N-1}\sum_{i=1}^{N}\lvert X_i-\mu\rvert^2$ |
| Root mean square (RMS) | $\sum_{i=1}^{N}\frac{1}{N}X_i^2$ |
| Log detector (LOG) | $e^{\frac{1}{N}\sum_{i=1}^{N}log(\lvert X_i\rvert)}$ |
| Waveform length (WL) | $\sum_{i=1}^{N-1}\lvert X_{i+1}-X_i\rvert$ |
| Standard deviation (STD) | $\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\lvert X_i-\mu\rvert^2}$ |
| Difference Absolute standard deviation (AAC) | $\sqrt{\frac{1}{N-1}\sum_{i=1}^{N-1}(X_{i+1}-X_i)^2}$ |
| Fractal dimension (FD) | Higuchi's algorithms with $m = 5$ |
| Maximum fractal length (MFL) | $log(\sum_{i=1}^{N-1}\lvert X_{i+1}-X_i\rvert)$ |
| Myopulse percentage rate (MYO) | Percentage of time where the signal is bigger than two times the mean |
| Integrated EMG (IEMG) | $\sum_{i=1}^{N}\lvert X_i\rvert$ |
| Simple square EMG (SSI) | $\sum_{i=1}^{N}X_i^2$ |
| Zero crossing (ZC) | The number of times in which the signal crosses its mean |
| Slope sign change (SSC) | The number of times in which the slope of the sign changes |
| Wilson amplitude (WAMP) | $\sum_{i=1}^{N-1}\lvert X_i-X_{i+1}\rvert > \epsilon$ where $\epsilon$ is the mean of the signal |
| Autoregressive coefficients (AR, 4 coefficients) | AR parameter estimation via Yule-Walker method |

Figura 16 Recorte con las características de la señal, temporales, utilizadas. [7].

| Frequency Features | Descriptor |
|---|---|
| Main peak amplitude (Pmax) | Maximum peak |
| Main peak frequency (Fmax) | Frequency of the max peak |
| Mean power (MP) | $\frac{1}{N}\sum_{i=1}^{N}\lvert P_i\rvert$ |
| Total power (TP) | $\sum_{i=1}^{N} P_i$ |
| Mean frequency (MNF) | Estimates the mean normalized frequency of the power spectrum |
| Median frequency (MDF) | Estimates the median normalized frequency of the power spectrum |
| Standard deviation (STD) | $\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\lvert P_i-\mu\rvert^2}$ |
| 1st spectral moment (SM1) | Spectral moments |
| 2nd spectral moment (SM2) | Spectral moments |
| 3rd spectral moment (SM3) | Spectral moments |
| Kurtosis (KUR) | Kurtosis of the power spectrum |
| Skewness (SKW) | Skewness of the power spectrum |
| Autocorrelation (Auto, 3 coefficients) | 3 firsts coefficients of the autocorrelation |

Figura 17 Recorte con las características de la señal, frecuenciales, utilizadas. [7].

La función en cuestión se deja en el Anexo de este trabajo donde se pueden ver todas las características que se calculan con ella. Para la prueba que se va a mostrar a continuación se utilizaron todas las características que se pueden calcular con la función.

De esta forma se consigue que la señal este definida por muchos menos valores y pueda ser más fácil al programa clasificarlas. En la Figura 18 se pueden ver los resultados de esta prueba.

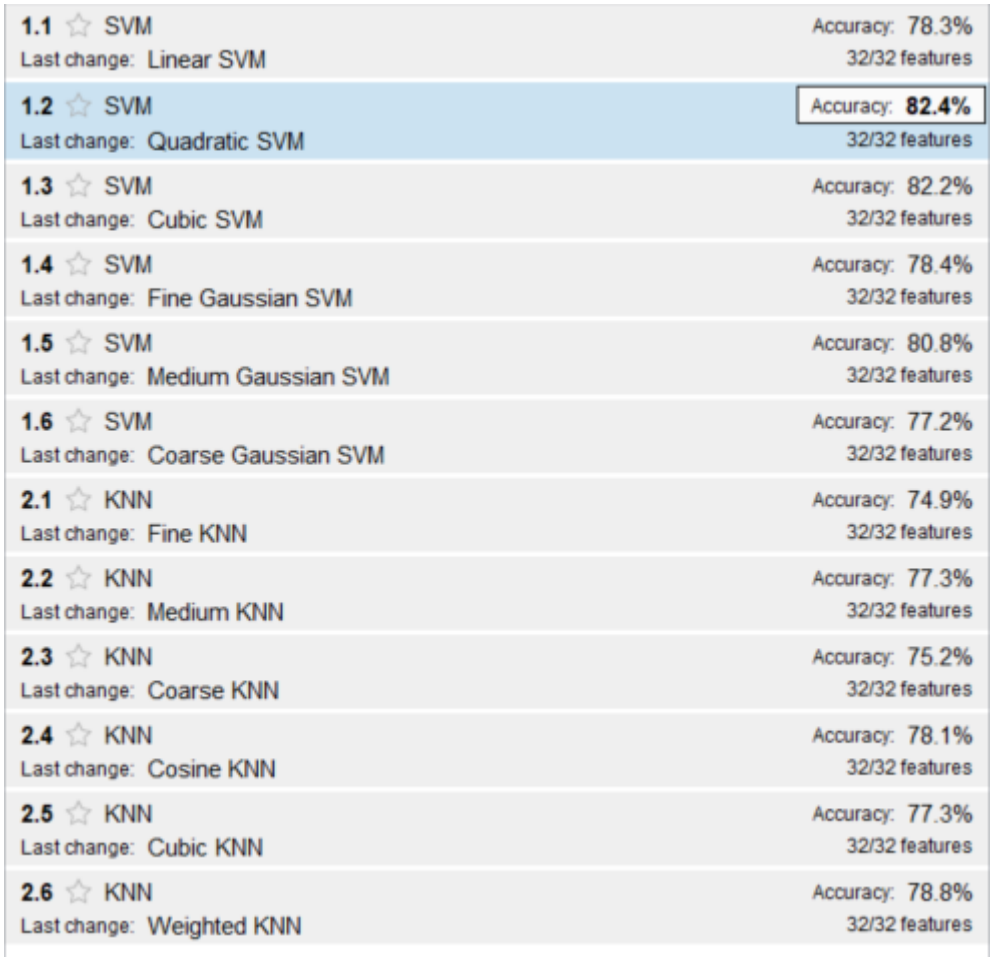


Figura 18 Resultados del entrenamiento con los distintos modelos tras calcular características de la señal.

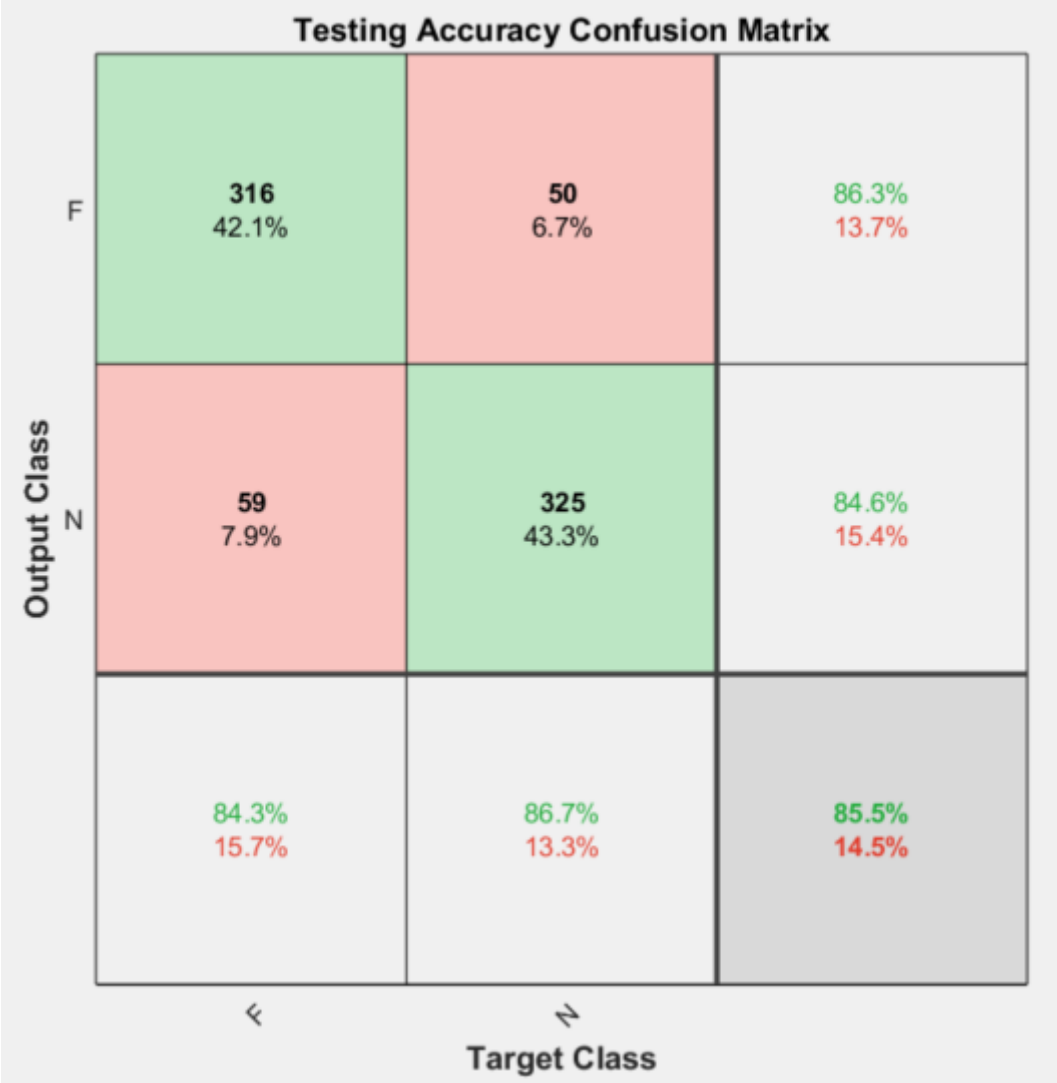


Figura 19 Matriz de confusión de datos de test. Quadratic SVM

Como se puede observar en la Figura 19, ahora se han conseguido unos resultados realmente buenos, llegando a un 85 % de acierto con los datos de test utilizando el Quadratic SVM.

Para esta prueba hay que mencionar que el tiempo invertido en el cálculo de características fue bastante elevado. Se dejó el programa ejecutando varias horas hasta y una vez terminó se procedió a probar los algoritmos.

Por último, estas tasas de acierto podrían ser mejores escogiendo otras características que definan la señal. El número de combinaciones posibles es muy elevado y, al igual que con las configuraciones de LSTM, se trata de ir probando e investigando hasta encontrar una solución mejor.

### 3.6. Optimizar el uso de características.

Para intentar mejorar los resultados anteriores y optimizar programa para solo calcular las características más importantes se procede a utilizar un algoritmo muy útil llamado relief que se puede implementar fácilmente con Matlab. Con él se pueden escoger las características que sirven para diferenciar mejor las señales [8].

Para la siguiente prueba se ha ejecutado relief para ordenar de mejor a peor las características anteriormente calculadas y se va a mostrar una tabla con los resultados de ejecutar SVM Medium Gaussian con un número variable de estas. La razón de utilizar este algoritmo y no el cuadrático que dio un mejor resultado en las pruebas con todas las características a la vez es porque es mucho más rápido de calcular. Las cinco mejores características tras ordenar con relief son: Median Frecuency, Slope Sign Change, Fractal Dimension, Sample Entropy y Skewness.

Hay que mencionar que en la siguiente se están mostrando unos porcentajes de acierto con los datos de entrenamiento, asumimos que para los datos de test no habría demasiadas variaciones.

| Nº de Características | Tasa de acierto |
|---|---|
| 1 | 65 % |
| 2 | 67.9 % |
| 3 | 71.3 % |
| 5 | 78.1 % |
| 10 | 81.1 % |
| 15 | 81.8 % |
| 20 | 79,7 % |

Figura 20 Tasa de acierto en función del número de características utilizadas con el algoritmo SVM Medium Gaussian.

Como se puede ver, la tasa de acierto va mejorando hasta que a partir de 10 características ya se mantiene o incluso empeora. Es por ello por lo que no por utilizar más características la tasa de acierto va a ser mejor, sino que se trata de utilizar las más correctas.

Para finalizar este apartado, se presenta la 0 en la que se puede ver una comparación con diagramas de caja de las cuatro mejores características. Se puede apreciar que existen claras diferencias en la distribución para ambos tipos de señal, es por ello que estas características son realmente útiles para la clasificación.

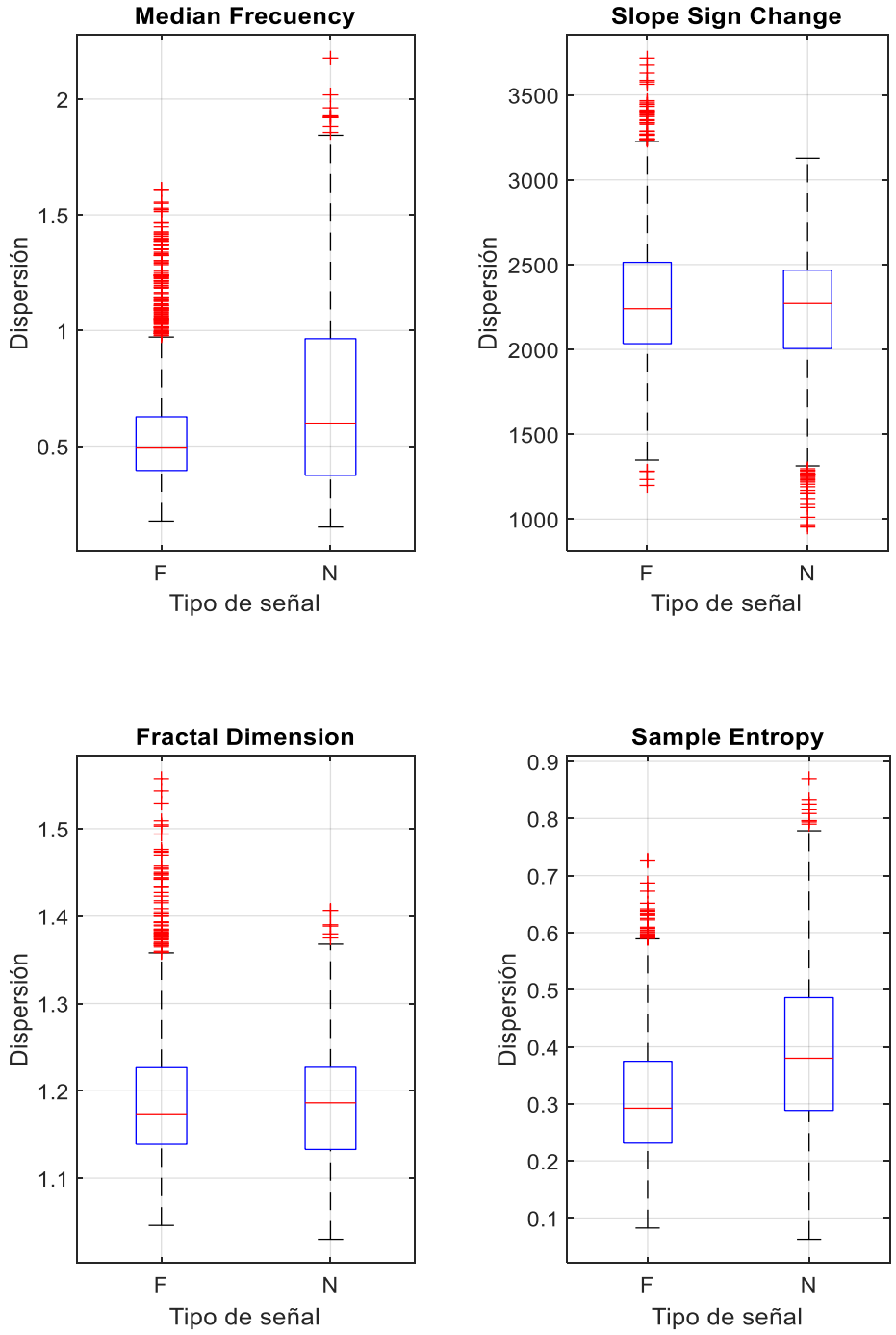


Figura 21 Diagramas de caja de las 4 mejores características según el tipo de señal

### 3.7. LSTM con cálculo de características.

A la vista de los resultados del apartado anterior, se procede a realizar pruebas otra vez con LSTM pero realizando un procesado de los datos. Para este caso no se calculan valores concretos, sino que se seguirán utilizando secuencias.

Estas secuencias se pretenden calcular teniendo en cuenta la frecuencia instantánea y la entropía de la señal. La entropía es una medida del desorden de la señal según el artículo

que puede leerse en [9], puede servir para detectar cambios en la distribución de la señal, concretamente en señales de EEG.

En la Figura 22 se puede ver el espectrograma correspondiente a una señal Focal y a una No focal. No existen evidencias a simple vista que permitan clasificarlas, pero es posible que la red neuronal si sea capaz de encontrar las diferencias.

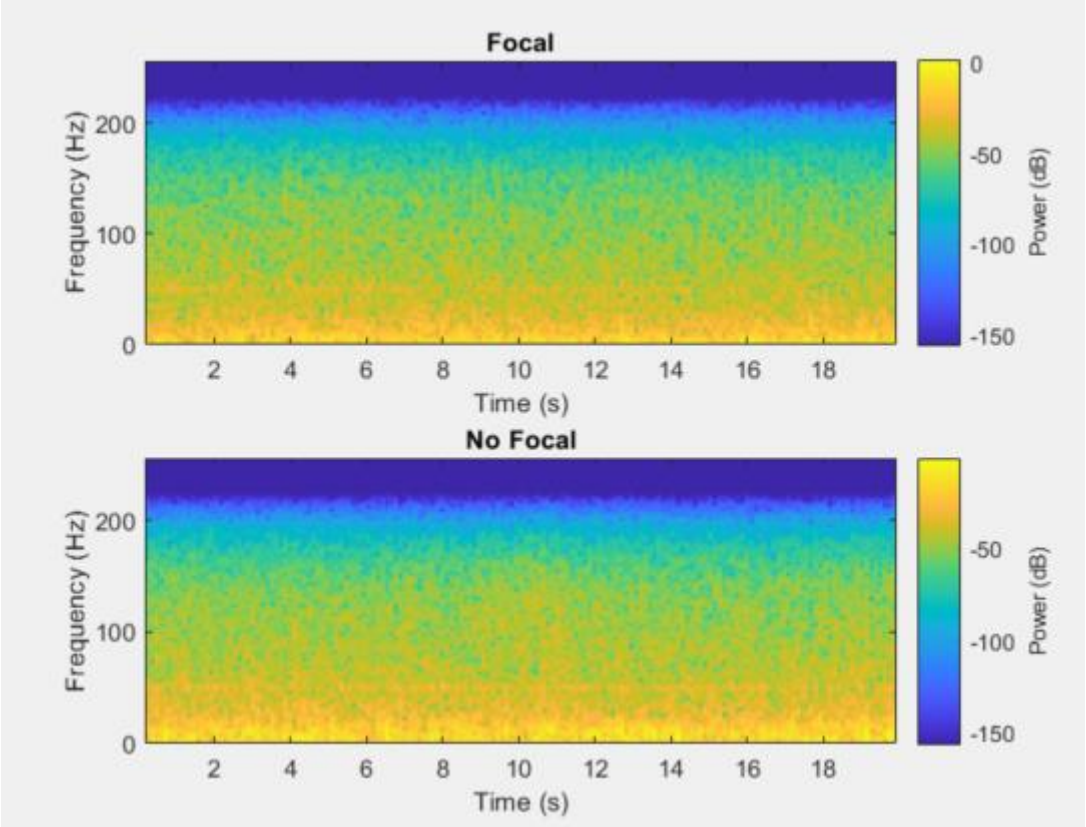


Figura 22 Espectrograma de una señal Focal y otra No focal

También se muestra en las figuras siguientes la frecuencia instantánea y la entropía de esas mismas señales.

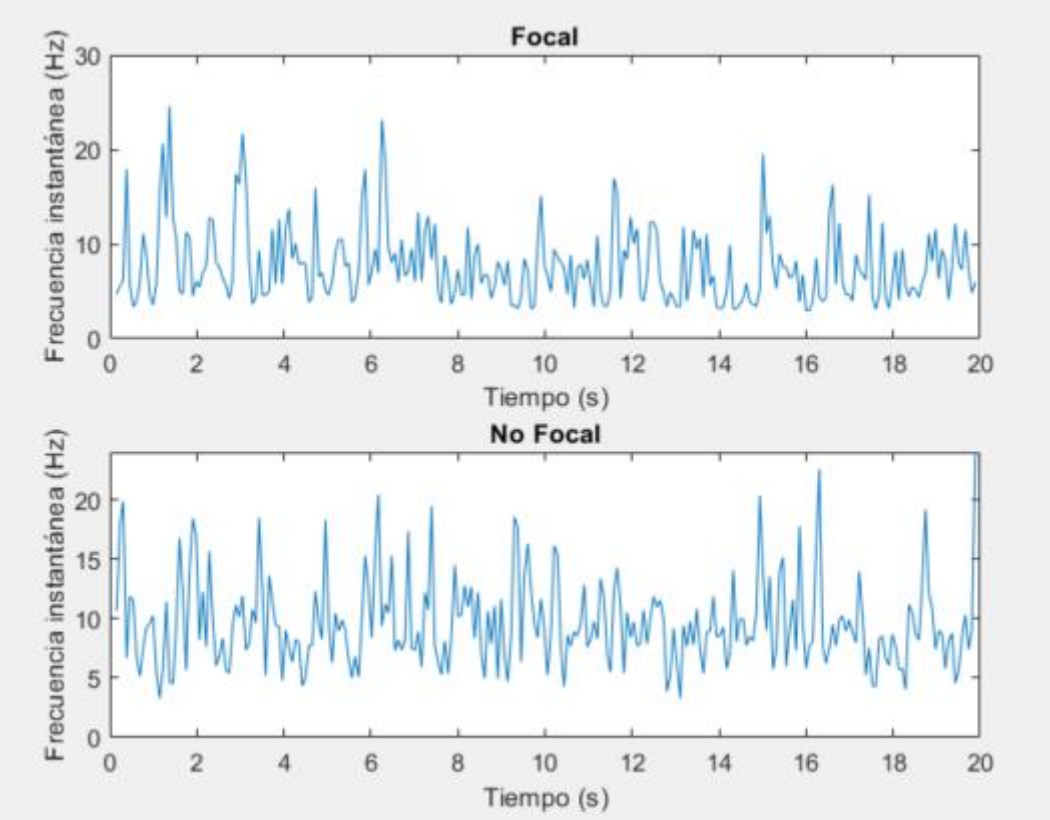


Figura 23 Frecuencia instantánea de una señal Focal y otra No focal

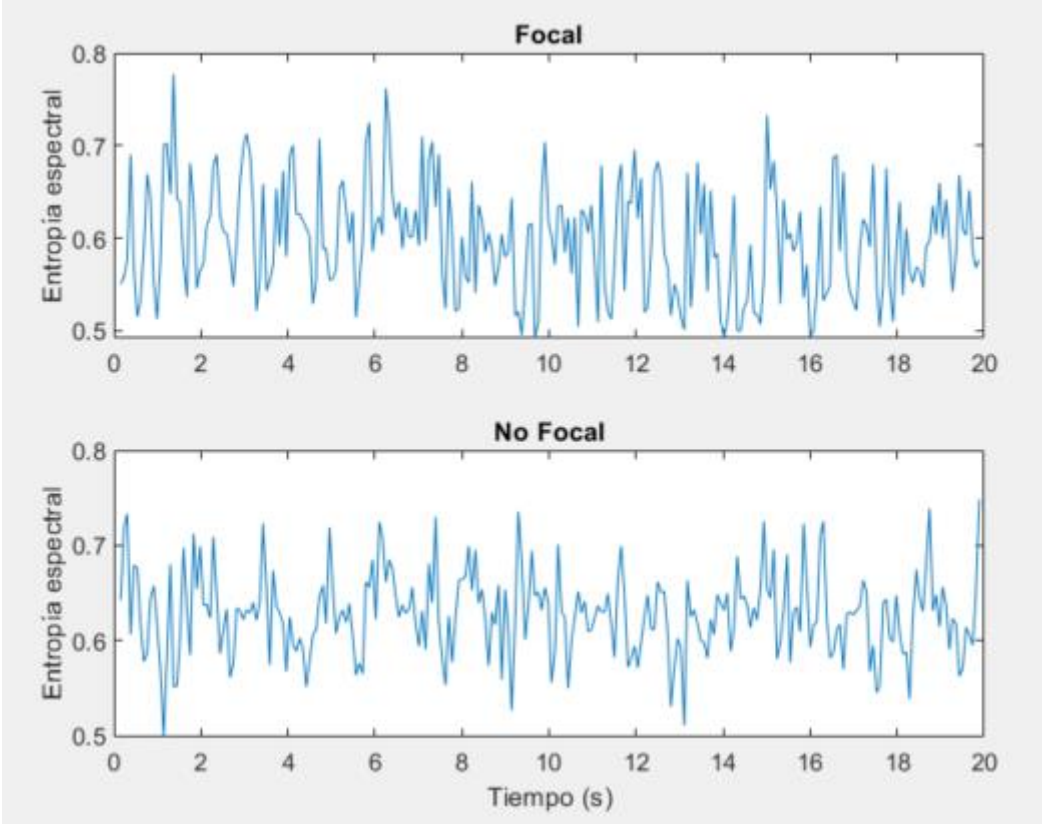


Figura 24 Entropía de una señal Focal y otra No focal

Con estas secuencias, el siguiente paso es ejecutar el algoritmo LSTM con dos entradas. Se utilizará una configuración sencilla como las vistas en el apartado 3.4. En la Figura 25 se puede ver que ahora LSTM si presenta una evolución más ascendente, llegando cerca del 80% con los datos de entrenamiento.

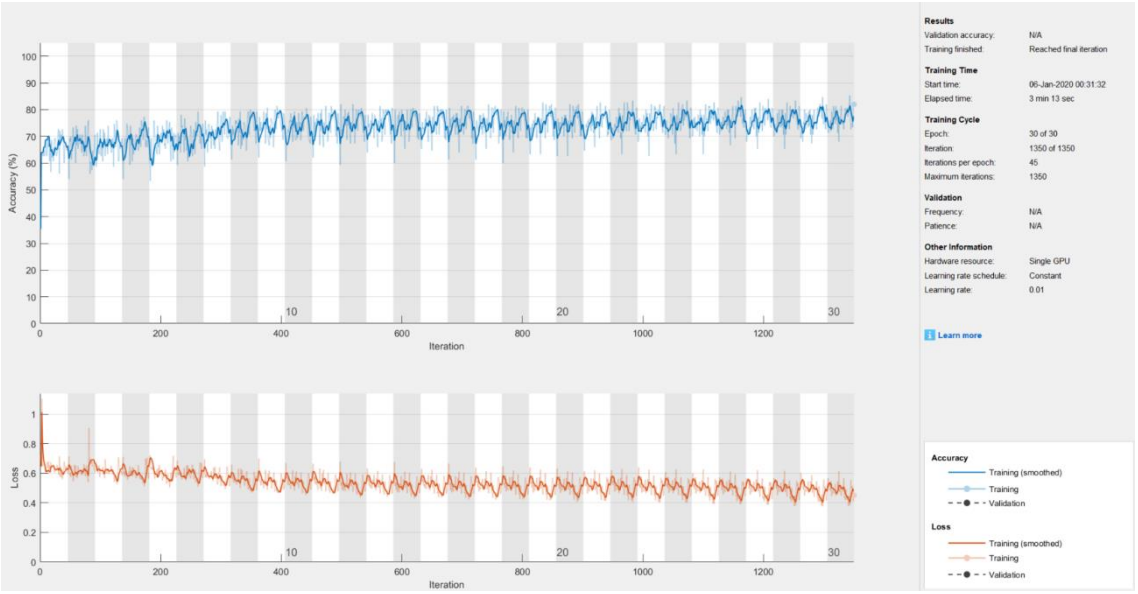


Figura 25 Evolución de la precisión con la entropía y frecuencia instantánea en los datos de entrenamiento. LSTM

En la Figura 26 se puede observar que para los datos de test casi se alcanza ese 80% de acierto. Es inferior a lo obtenido en el apartado 3.5 pero no deja de ser un buen resultado.

Un matiz interesante de esta ejecución es que se realizó con BILSTM usando 100 HiddenUnits sin embargo, también se realizó otra prueba con 200 HiddenUnits y el resultado fue peor en lugar de mejorar. Esto indica que no por utilizar más unidades el resultado va a ser mejor. Lo que se debe hacer es encontrar el número adecuado para cada problema, que en nuestro caso 100 parece ser suficiente.

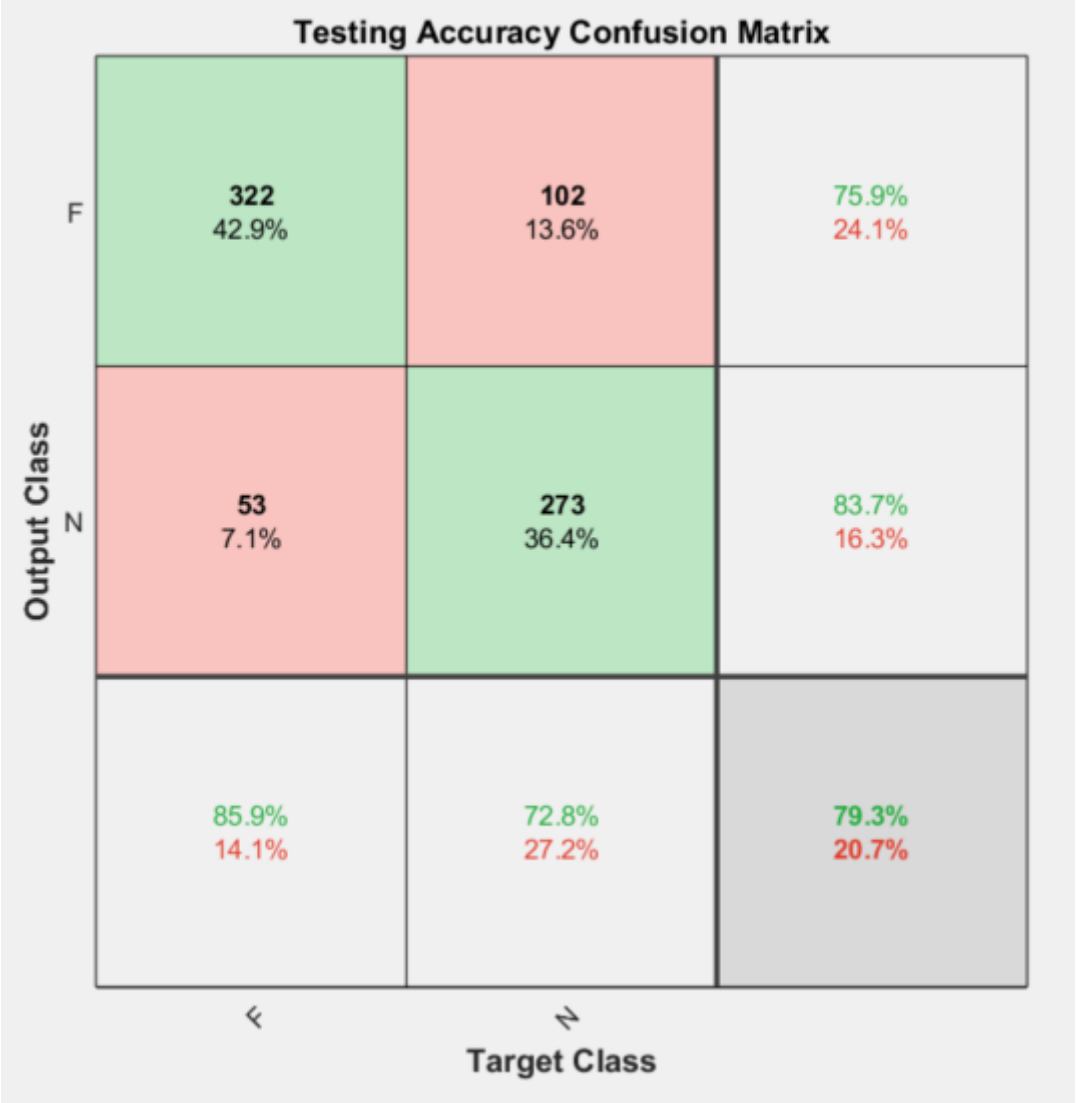


Figura 26 Matriz de confusión de los datos de test con la entropía y frecuencia instantánea. LSTM

Para intentar mejorar estos resultados se prepararon algunas funciones para realizar cálculos de características en ventanas de tiempo de la señal, de forma que se obtuviesen un número determinado de elementos que coincidiese con el de la entropía y frecuencia instantánea. Dicho de otra manera, se divide la secuencia completa en ese número determinado de ventanas y se realizan cálculos como la mediana o el valor cuadrático medio.

También se optó por utilizar la DCT (Transformada de coseno discreto). Con ella se pueden escoger los primeros valores más representativos de la señal para utilizarlos también como entrada al algoritmo.

En la siguiente prueba se utilizaron 5 secuencias como entrada al algoritmo LSTM. Estas son la entropía, la frecuencia instantánea, los primeros elementos más representativos de la DCT y cálculos de la mediana y valor cuadrático medio. Tras utilizar todos estos cálculos se consiguió una mejoría del resultado hasta alcanzar el 81 % tal y como se ve en la Figura 27.

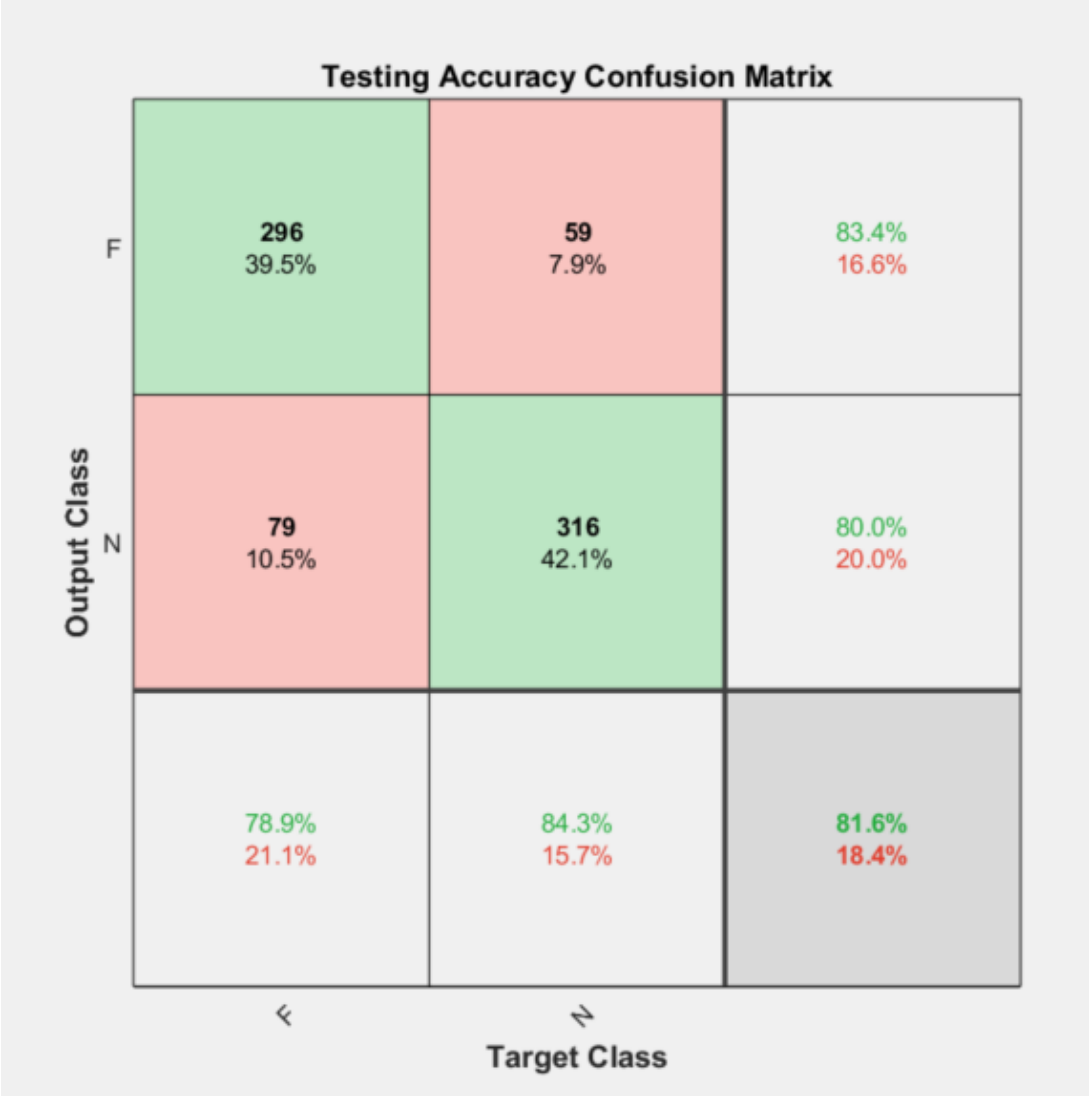


Figura 27 Matriz de confusión de los datos de test con la entropía y frecuencia instantánea. LSTM.

Como en el resto de los apartados, cabe la posibilidad de seguir probando más combinaciones que puedan servir para mejorar los resultados. Sin embargo, aunque no supere el 85 % de las pruebas con SVM, queda demostrado que LSTM es también una opción viable para clasificar este tipo de señales de pacientes con o sin epilepsia.

# 4. Conclusiones

En este apartado se exponen las conclusiones más importantes elaboradas a partir de los resultados de todas las pruebas, experiencia y conceptos aprendidos durante el desarrollo del TFM. De igual manera también se exponen críticas y valoraciones propias tras el esfuerzo invertido.

- Los métodos de aprendizaje automático en general son relativamente fáciles de implementar y se pueden aplicar a muchos tipos de problemas distintos. Sin embargo, saber qué tipo de algoritmo y método utilizar a priori es complejo y tiene mucho peso la experiencia para saber cuál sería el más adecuado según el problema, conjunto de datos de partida, etc.

- Cuando se trabaja con secuencias como las de la base de datos que se utilizó en el trabajo se necesita una potencia de cálculo elevada para que el tiempo de las ejecuciones no sea demasiado elevado. El uso de tarjetas gráficas mejora en gran medida los tiempos de ejecución de los algoritmos y Matlab permite realizar cálculos con las mismas fácilmente.

- Aunque en algunos trabajos se ha visto que con los datos sin procesar se pueden obtener buenos resultados clasificando secuencias con LSTM. En nuestro caso resultó casi imposible que el algoritmo aprendiera correctamente. Aun así, se aprendieron varios conceptos y prácticas útiles como la normalización, filtrado o desorden de las señales antes de introducirlas en la red. Además, el número de opciones para configurar las capas de LSTM deja la posibilidad de que se pueda llegar a conseguir una tasa de acierto que pase del 50% en este caso, aunque lo más probable es que siempre se obtengan mejores resultados con un cálculo de características previo.

- Matlab dispone de herramientas muy útiles para el diseño de sistemas basados en Machine Learning o Deep Learning. Muestra de ello son las herramientas que se han utilizado en este trabajo como el Classification Learner o el Deep Network Designer.

- Preparar los datos para la utilización de estos algoritmos no es una tarea trivial y se debe poner especial cuidado para no cometer errores de etiquetado de las señales. Además, cada herramienta tiene sus peculiaridades y se deben aprovechar las ventajas que tiene Matlab para trabajar con tablas, vectores, matrices o celdas para cumplir los requisitos del método que se vaya a utilizar.

- Según las pruebas realizadas, la mejor opción para clasificar estas señales consiste en utilizar alguno de los algoritmos de SVM que se vieron en el Classification

Learner una vez se han calculado características de la señal. Según qué características se utilicen los resultados serán mejores o peores y plantear otras formas de calcularlas podría ser uno de los caminos a seguir en el futuro si se desea continuar con el proyecto.

- Se ha visto que LSTM es finalmente una opción viable para clasificar las señales, pero solo si se realiza un procesado previo de la señal al igual que con los algoritmos SVM.

- Por último, se ha visto que el entrenamiento y procesado de señales reales con un numero de muestras tan elevado se hace muy lento. Otra opción posible para mejorar los tiempos es utilizar la computación en la nube de la mano de Amazon Web Services (AWS) o Nvidia Cloud. Los dos son compatibles con Matlab, aunque solo Amazon presenta opciones gratuitas por el momento. En la realización de este proyecto se valoró su utilización, pero se descartó por la complejidad añadida tras empezar a seguir los pasos para desplegar el software. Sin embargo, se opina que, para otro proyecto de mayor duración, invertir tiempo en desplegar una solución con AWS gratuita puede ser muy beneficioso.

# 5. Bibliografía


[1] Andrzejak RG, Schindler K, Rummel C (2012). Nonrandomness, nonlinear dependence, and nonstationarity of electroencephalographic recordings from epilepsy patients. Phys. Rev. E, 86, 046206

[2] Classify ECG Signals Using Long Short-Term Memory Networks

[3] How to Scale Data for Long Short-Term Memory Networks in Python - by Jason Brownlee on July 7, 2017 in Deep Learning for Time Series

[4] Classification Learner App - Interactively train, validate, and tune classification models

[5] Hsu, C. W., Chang, C. C., & Lin, C. J. (2003). A practical guide to support vector classification.

[6] Máquinas de Vector Soporte (Support Vector Machines, SVMs) - Joaquín Amat Rodrigo - Abril, 2017

[7] Solé-Casals J, Anchustegui-Echearte I, Marti-Puig P, Calvo PM, Bergareche A, Sánchez-Méndez JI and Lopez-de-Ipina K (2019) Discrete Cosine Transform for the Analysis of Essential Tremor. Front. Physiol. 9:1947. doi: 10.3389/fphys.2018.01947

[8] Relief-Based Feature Selection: Introduction and Review

[9] Detección de eventos en señales de EEG mediante Entropía - Andrea N Bermúdez, Enrique M Spinelli y Carlos M Muravchik

# 6. Anexo

En este capítulo se presenta el código de los programas más importantes escritos en Matlab para la realización de los cálculos y representaciones que se han expuesto en este trabajo. Dichos códigos se encuentran aún en un estado de desarrollo por lo que a la hora de ejecutarlos es posible que se requiera de ajustar alguna variable de entrada en diversos puntos del programa. De la misma manera los comentarios a verde son solo una pequeña ayuda para entender el funcionamiento de las funciones creadas.

```
% Preparación de los datos

Num_files_F = 3750;
Num_files_N = 3750;

fs = 512;

filename_F = 'Data_F_Ind';
filename_N = 'Data_N_Ind';

path = '.\Datos de origen\Data_Complete\';


t=1;
for i = 1:Num_files_F
    if i>=1000
        filename_aux = strcat(filename_F,string(i));
    elseif i>=100
        filename_aux = strcat(filename_F,'0',string(i));
    elseif i>=10
        filename_aux = strcat(filename_F,'00',string(i));
    else
        filename_aux = strcat(filename_F,'000',string(i));
    end
    Data{t,1} = load(strcat(path,filename_aux,'.txt'))';
    Data_Labels{t,1} = 'F';
    t=t+1;
end


for i = 1:Num_files_N
    if i>=1000
        filename_aux = strcat(filename_N,string(i));
    elseif i>=100
        filename_aux = strcat(filename_N,'0',string(i));
    elseif i>=10
        filename_aux = strcat(filename_N,'00',string(i));
    else
        filename_aux = strcat(filename_N,'000',string(i));
    end
    Data{t,1} = load(strcat(path,filename_aux,'.txt'))';
    Data_Labels{t,1} = 'N';
    t=t+1;
end
Data_Labels = categorical(Data_Labels);



%% Divide los datos en dos grupos, uno de entrenamiento y otro de test

FocalX = Data(Data_Labels=='F');

FocalY = Data_Labels(Data_Labels=='F');

NoFocalX = Data(Data_Labels=='N');
NoFocalY = Data_Labels(Data_Labels=='N');

[trainIndF,~,testIndF] = dividerand(length(FocalX),0.9,0.0,0.1);
```

```
[trainIndN,~,testIndN] = dividerand(length(NoFocalX),0.9,0.0,0.1);

XTrainF = FocalX(trainIndF);
YTrainF = FocalY(trainIndF);

XTrainN = NoFocalX(trainIndN);
YTrainN = NoFocalY(trainIndN);

XTestF = FocalX(testIndF);
YTestF = FocalY(testIndF);

XTestN = NoFocalX(testIndN);
YTestN = NoFocalY(testIndN);

XTrain = [XTrainF; XTrainN];
YTrain = [YTrainF; YTrainN];

XTest = [XTestF; XTestN];
YTest = [YTestF; YTestN];

% Reordena los datos de forma aleatoria
ind_perm=randperm(length(XTrain));
XTrain=XTrain(ind_perm);
YTrain=YTrain(ind_perm);

ind_perm=randperm(length(XTest));
XTest=XTest(ind_perm);
YTest=YTest(ind_perm);
```

**%% Función para calcular varias características y generar un vector con todas ellas. (Cortesía de Jordi Solé Casals)**

```
function [feature_final] = generate_feature(AgoS)

%Entropy - 1
% SENT= SampEntropia(3, 0.2*std(AgoS), AgoS);
SENT= SampEntropia(AgoS,3,0.2,'chebychev');

%Mean absolute value -2
MAV=mean(AgoS);

%Variance - 3
VAR=var(AgoS);

%Root mean square - 4
RMS=rms(AgoS);

%Log detector - 5
s=log(abs(AgoS));
s(~isfinite(s))=0;
LOG=exp(1/length(AgoS))*(sum(s));

%Average amplitude change or Waveform Length - 6
WL=sum(diff(AgoS))/length(AgoS);

%Standard deviation - 7
STD=std(AgoS);

%Difference absolute standard deviation - 8
AAC=sqrt(sum(diff(AgoS).^2)/(length(AgoS)-1));

%Fractal Dimension using Higuchi's algorithms. La funci? Wei() calcula la
    %Weiertrass function que t? una dimensi? fractal de 1.5. Tamb? provo amb la
    %Blancmange-Takagi Function que t? un FD=1
    %W = Wei();
    %[B,s] = blancmange(0.5,1); - 9
FD=hfd(AgoS);
    %FD_2=Higuchi_FD(W,5);
    %FDB=hfd(B);
    %FDB_2=Higuchi_FD(B,5);

%Maximum Fractal Length - 10
if sum(diff(AgoS))==0
    MFL=0;
else
    MFL=log(sum(diff(AgoS)));
end
```

```
%Myopulse percentage rate - 11
MYO=sum(AgoS>(2*mean(AgoS)))/length(AgoS);

%Integrated EMG - 12
IEMG=sum(abs(AgoS));

%Simple Square Integral EMG -13
SSI=sum(AgoS.^2);

%Zero Crossing - 14
ZC = sum(abs(diff(AgoS>mean(AgoS))));

%Slope Sign Change - 15
SSC=sum(diff(sign(diff(AgoS)))~=0);

%Wilson Amplitude - 16
th=mean(AgoS);
WAMP=sum(abs(diff(AgoS>th)));

%Autoregressive Coeficcients 17,18,19,20
AR=aryule(AgoS,4);

%PSD usig Wlech method - 21,22
[pxx,f] = pwelch(AgoS,[],[],[],100);
%figure()
%plot(f(1:246),pxx(1:246))
%title('PSD of Accelrometer')
%Peak amplitude & frequency
[Pmax,Fmax]=max(pxx(2:end)); %Excluding DC

%Mean power - 23
MP=mean(pxx);

%Total Power - 24
TP=sum(pxx);

%Mean Frequency and Median Frequency -25, 26
MNF=meanfreq(pxx,f);
MDF=medfreq(pxx,f);
%line([MNF,MNF],[min(pxx) max(pxx)],'Color','red','LineStyle','--');
%line([MDF,MDF],[min(pxx) max(pxx)],'Color','green','LineStyle','-.');
%legend('PSD','Mean Freq','Median Freq')

%The 1st, 2nd and 3rd spectral moments - 27,28,29
SM1=sum(pxx.*f);
SM2=sum(pxx.*f.^2);
SM3=sum(pxx.*f.^3);

%Kurtosis - 30
KUR=kurtosis(AgoS);

%Skewness - 31
SKW=skewness(AgoS);
%Autocorrelation - 32,33,34
feats = zeros(1,3);

minprom = 0.0005;
mindist_xunits = 0.3;
minpkdist = floor(mindist_xunits/(1/100));

% Separate peak analysis for 3 different channels

    [c, lags] = xcorr(AgoS);

    [pks,locs] = findpeaks(c,...
        'minpeakprominence',minprom,...
        'minpeakdistance',minpkdist);

    tc = (1/100)*lags;
    tcl = tc(locs);

    % Feature 1 - peak height at 0
    if(~isempty(tcl))    % else f1 already 0
        feats(1) = pks((end+1)/2);
    end
    % Features 2 and 3 - position and height of first peak
    if(length(tcl) >= 3)    % else f2,f3 already 0
        feats(2) = tcl((end+1)/2+1);
```

```
        feats(3) = pks((end+1)/2+1);
    end


%With Ds
%feature_final=[Ds MAV VAR RMS LOG WL STD AAC FD MFL MYO IEMG SSI ZC SSC WAMP AR(2:end)
Pmax Fmax MP TP MNF MDF SM1 SM2 SM3];
%without Ds

%Treiem fora la Fmax ja que sempre dona el mateix
output=[SENT MAV VAR RMS LOG WL STD AAC FD MFL MYO IEMG SSI ZC SSC WAMP AR(2:end) Pmax
MP TP MNF MDF SM1 SM2 SM3 KUR SKW feats];
%output=table(SENT,MAV,VAR,RMS,LOG,WL,STD,AAC,FD,MFL,MYO,IEMG,SSI,ZC,SSC,WAMP,AR(2),AR(3
),AR(4),AR(5),Pmax,MP,TP,MNF,MDF,SM1,SM2,SM3,KUR,SKW,feats(1),feats(2),feats(3));
%output=table(output,'ColumnNames',{'SENT' 'MAV' 'VAR' 'RMS' 'LOG' 'WL' 'STD' 'AAC' 'FD'
'MFL' 'MYO' 'IEMG' 'SSI' 'ZC' 'SSC' 'WAMP' 'AR1' 'AR2' 'AR3' 'AR4' 'Pmax' 'MP' 'TP'
'MNF' 'MDF' 'SM1' 'SM2' 'SM3' 'KUR' 'SKW' 'feast1' 'feast2' 'feast3'});
feature_final=output;
%feature_final = dataset({output
'MAV','VAR','RMS','LOG','WL','STD','AAC','FD','MFL','MYO','IEMG','SSI','ZC','SSC','WAMP'
,'AR1','AR2','AR3','AR4','Pmax','Fmax','MP','TP','MNF','MDF','SM1','SM2','SM3'});
end
```

**%% Función para el cálculo de los valores de la DCT**

```
function [DCT_Trunc] = DCT_Trunc(Signal,N)
%  Realiza la DCT de la señal y trunca en los N valores más
%  representativos.
Cell_Seq_DCT = dct(Signal);
[XX,ind] = sort(abs(Cell_Seq_DCT),'descend');
% i = 1;
% while norm(Cell_Seq_DCT(ind(1:i)))/norm(Cell_Seq_DCT) < 0.99
%     i = i + 1;
% end
% needed = i;
needed = N;
DCT_Trunc=(ind(1:needed));
end
```

**%% Función para el cálculo de los valores RMS en N ventanas de la señal**

```
function [Random_Signal] = RMS_Test(Signal,V)
%  Realiza el test de mediana de la señal en un número de ventanas
%  marcada por V.

N = floor(length(Signal)/V);
Random_Signal=zeros(1,V);
for i=1:V-1
    ventana = Signal(1+N*(i-1):i*N);
    Random_Signal(i) = rms(ventana);
end

end
```

**%% Ejemplo de código para normalizar y truncar la longitud de las secuencias**

```
for i=1:numel(XTrain(:,1))
    XTrain_C(i,:)=XTrain{i}(1,:);
end
for i=1:numel(XTest(:,1))
    XTest_C(i,:)=XTest{i}(1,:);
end

XTrain_1 = num2cell(XTrain_C,2);
XTest_1 = num2cell(XTest_C,2);


Truncado = [1:500];
[XTrain_Nor,mu,sigma] = zscore(XTrain_C,0,'all');
XTrain_Nor_Trunc = num2cell(XTrain_Nor(:,Truncado),2);

XTest_Nor=(XTest_C-mu)./sigma;
```

```
XTest_Nor_Trunc = num2cell(XTest_Nor(:,Truncado),2);
```

**%% Preparación de los datos en formato tabla para utilizar en el Clasification Learner usando el cálculo de características**

```
XTrain_Feature =
cellfun(@(x)generate_feature(x),XTrain_Nor_Trunc,'UniformOutput',false);
XTest_Feature = cellfun(@(x)generate_feature(x),XTest_Nor_Trunc,'UniformOutput',false);

XTrain_T = array2table(cell2mat(XTrain_Feature(:)));
XTest_T = array2table(cell2mat(XTest_Feature(:)));
XTrain_T.class = YTrain;
XTest_T.class = YTest;
```

**%% Ejemplo de código para entrenar la red LSTM**

```
sequenceLength = numel(XTrain_Nor_Trunc{1,:});

numHiddenUnits = 100;
maxEpochs = 30;
MiniBatchSize = 120;
numFeatures = 1;
numClasses = 2;

layers = [ ...
    sequenceInputLayer(numFeatures)
    bilstmLayer(numHiddenUnits,'OutputMode','last')
    dropoutLayer
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];

options = trainingOptions('adam',...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize', MiniBatchSize, ...
    'InitialLearnRate', 0.01, ...
    'SequenceLength', 1000,...
    'GradientThreshold', 1, ...
    'ExecutionEnvironment',"auto",...
    'plots','training-progress', ...
    'Verbose',false);

net = trainNetwork(XTrain_Nor_Trunc,YTrain,layers,options);

trainPred = classify(net,XTrain_Nor_Trunc,'SequenceLength', 1000);
LSTMAccuracy = sum(trainPred == YTrain)/numel(YTrain)*100
figure
plotconfusion(YTrain',trainPred','Training Accuracy')

testPred = classify(net,XTest_Nor_Trunc,'SequenceLength', 1000);
LSTMAccuracy_Test = sum(testPred == YTest)/numel(YTest)*100
figure
plotconfusion(YTest',testPred','Testing Accuracy')
```