Proyecto No. 2
Clasificadores Multiclase

Valerie Parra Cortés
June 4, 2021

1 Dependencias

Para todo el proyecto desde el preprocesamiento hasta la implementación de los clasificadores se realizó utilizando Python 3.8.3 con las siguientes librerías como dependencias.

1. Open-cv Versión: 4.2.0.32
2. Numpy Versión: 1.18.1
3. Tensorflow Versión: 1.
4. Python Versión: 3.6.1
5. Keras Versión: 2.3.1
6. Sklearn Versión: 0.23.1

2 Obtención y preprocesamiento de los datos

Los utilizados fueron tomados de la base de datos Sleep cassette study and data, de está base de datos se tomaron 286 archivos , cada archivo contenida información de 7 señales fisiológicas:

1. EEG Fpz-Cz: Señal del nodo de la encefalografía ubicado en la posición Fpz (Ver Figura 1)
2. EEG Pz-Oz: Señal del nodo de la encefalografía ubicado en la posición Fpz (Ver Figura 1)
3. EOG horizontal: Señal de los electrodos del electrooculograma ubicado en los ojos
4. Resp oro-nasal: La respiración oronasal es aquella donde se respita por la boca y la nariz pero no se muestra signos de fatiga
5. EMG submental: Mide la actividad eléctrica de los músculos del cuerpo
6. Temp rectal: Temperatura rectal del paciente
7. Event marker: El marcador del evento, la documentación de la base de datos no específica este evento qué es

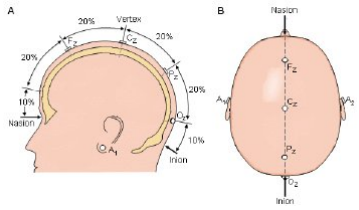

Figure 1: Diagrama de los electrodos. Figura tomada del siguiente Artículo

2.1 Perfilamiento los datos

Las siete señales están muestreadas a dos frecuencias diferentes: cada segundo o cada 0.01 segundo y esto era constante para cada señal fisiológica en todo el set de datos, por lo que se entandarizaron los datos a una misma frecuencia, es decir se tomo un dato por segundo ya que esta era la frecuencia más grande. El código para leer estos datos puede apreciarse en la sección 6.2.Una vez recolectados estos datos se perfilaron, la distribución de los datos puede

encontrada puede verse en la Figura 2, la tabla con estos datos tabulados se puede encontrar en la Tabla 17. En esta gráfica podemos ver que los datos están altamente desbalanceados ya que la mayoría, casi el 70% pertenecen a una sola clase, por lo que el problema es desbalanceado y hay que detallar una estrategia para atacar este problema en los clasificadores que se implementen, más adelante se detalla la estrategia usada por modelo.

Figure 2: Distribución original de los datos

2.2 Transformación de los datos

Para nuestros problema no nos interesan las clases *Movement Time* o *Sleep stage ?* por lo que eliminamos los datos asociados a estas dos clases. Por otro lado, muchas implementaciones estándar piden que las clases puedan representarse como vectores de bits, por lo que además se codificaron y simplificaron las clases de la siguiente manera:

1. *Sleep Stage W:* Representará el estado de que una persona este **despierta**, será representado con la clase '0'.
2. *Sleep Stage 1 & Sleep Stage 2*: Representarán que una persona este en **sueño ligero**, será representado con la clase '1'
3. *Sleep Stage 3 & Sleep Stage 4*: Representarán que una persona este en **sueño profundo**, será representado con la clase '2'
4. *Sleep Stage R:* Representará el estado de que una persona este **en sueño REM**, será representado con la clase '3'.

Una vez se limpiaron las clases que no son de interés para el problema y se unificaron los datos de la estapa 1,2 y de las etapas 3,4 la distribución final de los datos puede verse en la Figura 3.

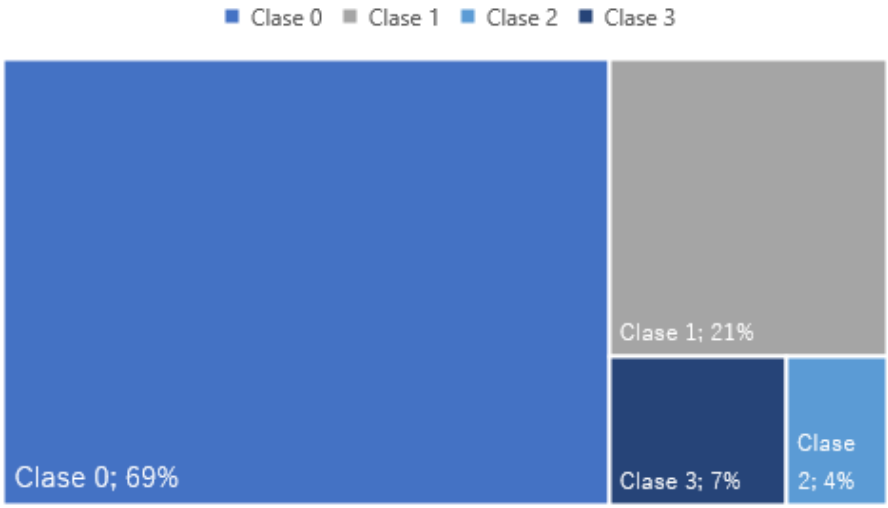

Figure 3: Distribución final de las clases

3 Red neuronal con los descriptores encontrados a mano

3.1 Datos

Como se tenía una gran cantidad de datos (alrededor de 170.000) pero estos estaban extremadamente desbalancedos aprovechamos el hecho de que estuvieran en tal cantidad para hacer sub-sampling, con los datos ya normalizados (y descartados los datos que hacían que los descriptores se indeterminaran como se explicará más adelante) se sub-muestro todas las clases a la cantidad de datos de la clase minoritaria, posteriormente dividimos 5% para validación, 5% para testeo. La división total de los datos es queda como se muestra en la Tabla 1.

	Entrenamiento		Validación		Prueba	
	Nro. datos	Porcentaje	Nro. datos	Porcentaje	Nro. datos	Porcentaje
Clase 0	5531	24.98%	304	24.72%	312	25.34%
Clase 1	5543	25.04%	308	25.04%	300	24.37%
Clase 2	5532	24.99%	315	25.61%	304	24.69%
Clase 3	5533	24.99%	303	24.63%	315	25.60%
Total	22139	100%	1230	100%	951	100%

Table 1: Distribución de los datos usados en la red neuronal

3.2 Descriptores encontrados a mano

Para los descriptores diseñados a mano se utilizaron las características de las señales más frecuentemente usadas para mediciones fisiológicas, se tomaron un total de siete descriptores los cuales se detallan a continuación:

1. Entropía espectral (PSE) Sea $X(\omega_i)$ la señal el espectro de la señal

$$P(\omega_i) = \frac{1}{N}|X(\omega_i)| \tag{1}$$

$$p_i = \frac{P(\omega_i)}{\sum_i P(\omega_i)} \tag{2}$$

$$PSE = -\sum_i p_i log(p_i) \tag{3}$$

2. Máxima amplitud de la señal en dB (MA) Sea $max(X)$ la función que retorna el valor máximo de la señal dada

$$MA = 10log_{10}(max(X)) \tag{4}$$

3. Peak Frecuency (*PF*) La Preak Frecuency de una señal se encuentra de la siguiente manera
   (a) Encuentre la transformada de Fourier de la señal
   (b) Encuentre el valor t_s, dónde se establece la simetría de la señal
   (c) Encuentre el máximo valor de la norma de cada punto de la transformada de Fourier S_m
   (d) Encuentre el máximo tiempo donde la respuesta de la transformada de Fourier de la señal sea t_m

   Sea entonces W la ventana de tiempo que se está analizando

$$PK = \frac{X(t_m - t_s)}{W} \tag{5}$$

4. Máximo valor de la señal $max(X)$
5. Mínimo valor de la señal $min(X)$
6. Promedio de la señal $mean(X)$
7. Mediana de la señal $median(X)$

Para este modelo se eliminó la señal asociada al *event maker* ya que en la documentación de la librería no especifica bien que representa esta señal y es probable que sólo introdujera ruido, por lo que se obtuvieron 42 descriptores en total (seis señales donde a cada una se le extrajo siete descriptores). En el proceso de extracción de estos descriptores había datos que por tener un valor tan pequeño hacían que se indeterminara el logaritmo que se usa para encontrar la amplitud de la señal, los datos que hacían que ese descriptor se indeterminara se eliminaron. Asimismo la desviación estándar de la *Peak Frecuency* daba muy pequeña y daba problemas ya que mandaba a un número muy grande los valores a la hora de hacer la normalización haciendo que *python* lo asignara como un Not a Number (NAN), por lo que este descriptor se eliminó quedándonos con sólo 41 descriptores.

Para el problema de clasificación con estos *features* se uso una red neuronal, se realizó validación cruzada para saber que arquitectura de la red utilizar. Los hiperparámetros fijos como los variables se muestran en las siguientes tablas.

Hiperparámetro	**Valor**
Optimizador	Stochastic descent gradient (SDG)
Función activación capa escondida	Relu
Función de pérdida	*Categorical cross entropy*
Función de activación capa de salida	*Softmax*
Número de iteraciones	1500

Table 2: Hiperparámetros fijos en la red neuronal

Hiperparámetro	**Valor**
Número de neuronas en la capa escondida	[125 150 200 250]
Tasa de aprendizaje	[0.0001 0.001 0.1 1]
Número de capas escondidas	[1-3]

Table 3: Hiperparámetros validados

3.3 Validación Cruzada

Los resultados de todos los modelos probados se pueden ver detalladamente en la Tabla 18. En la figura 4 podemos ver un mapa de calor de las exactitudes finales de los dos hiper-parámetros variados para una capa escondida, vemos que la mayor exactitud lograda está usando 250 neuronas con una tasa de aprendizaje de 0.1. Los peores resultados se obtienen cuando se usa una tasa de aprendizaje de 0.001 donde la exactitud general del modelo se mantiene debajo del 60%. En la Figura 5 tenemos el mismo mapa de calor pero está vez para la red neuronal con 2 capas escondidas, vemos que los resultados están bastante por debajo que los obtenidos con una capa escondida, sin embargo el hecho de que la exactitud baja bastante con la tasa de aprendizaje de 0.001, lo que nos empieza a indicar que este modelo es sensible a la tasa de aprendizaje. Por último, en la Figura 6 vemos los resultados para tres capas escondidas, al igual que en la arquitectura con una capa escondida el mejor desempeño se logra con la configuración de 250 neuronas y una tasa de aprendizaje de 0.1, que a su vez es la mejor de las tres modelos por lo que se seleccionó este como modelo final.

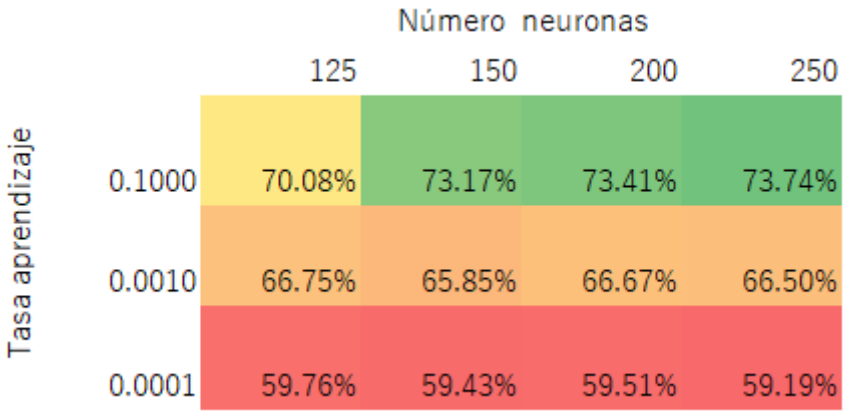

Figure 4: Resultados para una capa escondida

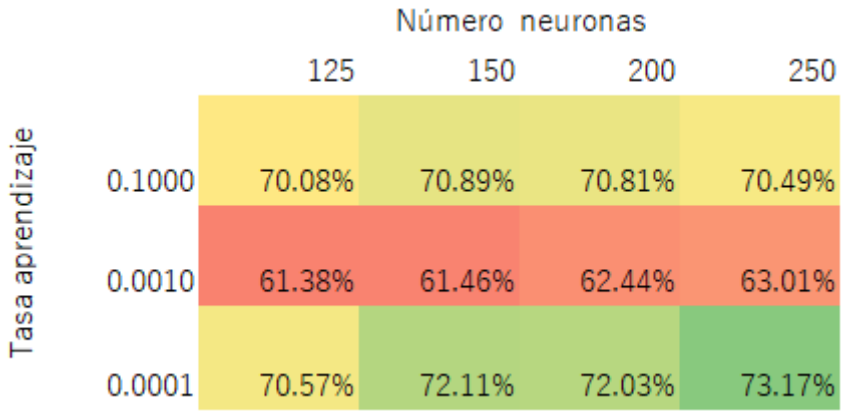

Figure 5: Resultados para dos capas escondida

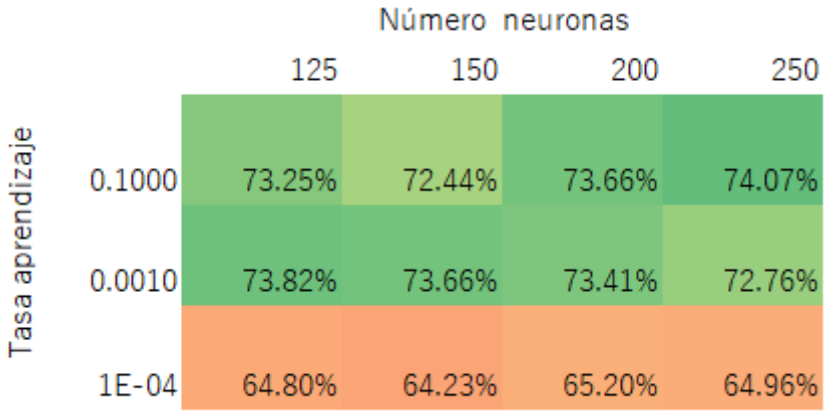

Figure 6: Resultados para tres capas escondida

3.4 Entrenamiento modelo final

Se re-entreno el modelo final con un total de 700 *epochs*, y se sacó la matriz de confusión del problema llegando a la matriz de confusión de la Tabla 13, vemos que en general lo valores de la diagonal son los más grandes como es lo esperado, para evaluar mejor el desempeño del clasificador en cada clase se encontró la sensibilidad y la especificad de cada clase (Tabla 5). Vemos que el modelo tiene mucho mejor desempeño para la clase 0 que tiene los mayores valores de estos dos indicadores. Esto puede deberse a que está clase representa cuando una persona está despierta, mientras que las otras tres clases representan ya estados pero pertenecen en general al estado *dormido*, por lo que tiene sentido que tengamos mejor desempeño en esta clase ya que es la menos relacionada con las otras. Algo similar ocurre con la clase 3, que representa sueño REM, es la que le sigue a la clase 0 en especificidad por lo que es posible que está clase también presente datos muy diferentes a las otras 3 clases (en especial con la señal 3 que represente el movimiento ocular). Las clases 1 y 2 son las que el algoritmo más confunde entre sí.

		Predicción			
		0	1	2	3
Real	0	270	19	9	6
	1	18	158	79	53
	2	6	43	252	14
	3	3	35	29	236

Table 4: Matriz de confusión modelo final red neuronal

	0	1	2	3
Especifidad	90.91%	61.96%	68.29%	76.38%
Sensitividad	88.82%	51.30%	80.00%	77.89%

Table 5: Sensitividad y especifidad para cada clase

4 Red profunda

4.1 Datos

Para la red profunda se busco usar más datos, ya que esta red tenía más parámetros que la anterior. Así que se sub-muestreo pero sin limitarnos a la clase minoritaria, sino que se dejaron un poco más de datos de las otras clases para conseguir más datos pero sin desbalancear el problema. La distribución de los datos se puede ver en la siguiente tabla

	Entrenamiento		Validación		Prueba	
	Nro. datos	Porcentaje	Nro. datos	Porcentaje	Nro. datos	Porcentaje
Clase 0	7210	26%	396	26%	394	25%
Clase 1	7164	26%	402	26%	434	28%
Clase 2	5706	21%	329	22%	312	20%
Clase 3	7232	27%	390	26%	378	27%
Total	27312	100%	1517	100%	1518	100%

Table 6: Distribución de los datos usados en la convolucional

Para la red profunda se dejo de entrada las señales puras, y cómo está entrada es relativamente pequeña (7 señales muestreadas cada segundo por 30s) decidimos también dejar la señal del movimiento, ya que dejaremos que la red aprenda sola las características que le interese. Con este propósito se implemento una CNN en python. Para configurar esta red lo que se hizo fue probar varias arquitecturas las cuales se describen a continuación

1. Modelo 1
   Parte convolucional
   (a) Capa convolucional en 2D con 2 filtros y un tamaño de kernel de 2x1 un border tipo same y función de activación *relu*
   (b) Pooling de 1x2
   (c) Pooling de 1x2
   (d) Dropout a una tasa de 0.2

   Parte neuronal
   (a) Capa con 128 neuronas y función de activación tipo *relu*
   (b) Capa con 50 neuronas y función de activación tipo *relu*
   (c) Capa de salida, 4 neuronas y función de activación tipo *softmax*
2. Modelo 2 Parte convolucional
   (a) Capa convolucional en 2D con 2 filtros y un tamaño de kernel de 2x1 un border tipo same y función de activación *relu*
   (b) Pooling de 1x2
   (c) Pooling de 1x2
   (d) Dropout a una tasa de 0.2

   Parte neuronal
   (a) Capa con 150 neuronas y función de activación tipo *relu*
   (b) Capa de salida, 4 neuronas y función de activación tipo *softmax*
3. Modelo 3
   Parte convolucional
   (a) Capa convolucional en 2D con 2 filtros y un tamaño de kernel de 2x1 un border tipo same y función de activación *relu*
   (b) Pooling de 1x2
   (c) Pooling de 1x2
   (d) Dropout a una tasa de 0.2

   Parte neuronal
   (a) Capa con 150 neuronas y función de activación tipo *relu*

(b) Capa con 150 neuronas y función de activación tipo *relu*

(c) Capa de salida, 4 neuronas y función de activación tipo *softmax*

4. Modelo 4
   Parte convolucional

   (a) Capa convolucional en 2D con 2 filtros y un tamaño de kernel de 2x1 un border tipo same y función de activación *relu*

   (b) Pooling de 1x2

   (c) Pooling de 1x2

   (d) Dropout a una tasa de 0.2

   Parte neuronal

   (a) Capa con 150 neuronas y función de activación tipo *relu*

   (b) Capa con 150 neuronas y función de activación tipo *relu*

   (c) Capa con 150 neuronas y función de activación tipo *relu*

   (d) Capa de salida, 4 neuronas y función de activación tipo *softmax*

5. Modelo 5
   Parte convolucional

   (a) Capa convolucional en 2D con 2 filtros y un tamaño de kernel de 2x1 un border tipo same y función de activación *relu*

   (b) Pooling de 1x2

   (c) Pooling de 1x2

   (d) Dropout a una tasa de 0.2

   Parte neuronal

   (a) Capa con 150 neuronas y función de activación tipo *relu*

   (b) Capa con 150 neuronas y función de activación tipo *relu*

   (c) Capa con 150 neuronas y función de activación tipo *relu*

   (d) Capa de salida, 4 neuronas y función de activación tipo *softmax*

6. Modelo 6
   Parte convolucional

   (a) Capa convolucional en 2D con 2 filtros y un tamaño de kernel de 2x1 un border tipo same y función de activación *relu*

   (b) Pooling de 1x2

   (c) Pooling de 1x2

   (d) Dropout a una tasa de 0.2

   Parte neuronal

   (a) Capa con 150 neuronas y función de activación tipo *relu*

   (b) Capa con 150 neuronas y función de activación tipo *relu*

   (c) Capa con 150 neuronas y función de activación tipo *relu*

   (d) Capa con 200 neuronas y función de activación tipo *relu*

   (e) Capa de salida, 4 neuronas y función de activación tipo *softmax*

4.2 Resultados

Los resultados del entrenamiento de los 6 modelos sugeridos se pueden ver en detalle en la Tabla 7. Aunque todos los modelos implementados tienen resultados más bien parecidos se tienen mucho mejores resultados con el modelo 6 (que sigue un arquitectura en la parte neuronal al mejor modelo obtenido para la red neuronal) con una exactitud global final de 83,7%

	Entrenamiento	Validación
Modelo 1	79.6%	79.0%
Modelo 2	80.3%	80.6%
Modelo 3	80.5%	81.3%
Modelo 4	81.1%	82.1%
Modelo 5	81.8%	82.1%
Modelo 6	83.7%	83.3%

Table 7: Resultados redes profundas entrenadas

4.3 Modelo final

Se re-entreno el modelo final y se evaluó el modelo con los mismos parámetros que el anterior. La matriz de confusión puede verse en la Tabla 8. Como podemos ver tenemos en general un mejor desempeño que en el modelo anterior, en general tenemos una exactitud de 83%, lo que indica que el modelo aumenta en 10% su exactitud. Al igual que el modelo anterior encontramos la especificidad y la sensibilidad de las clase, las cuales aumentaron alrededor de un 9% para cada clase. En la clase 0, la clase más fácil de clasificar por sus diferencias con las otras clases, se empezó a tener una clasificación casi perfecta. Ninguno de los indicadores baja de 72% por lo que en general el aprendizaje profundo resulto ser mucho mejor para este problema de clasificación que un aprendizaje con descriptores diseñados a mano.

		Predicción			
		0	1	2	3
Etiqueta	0	384	11	1	0
	1	3	352	27	20
	2	0	55	257	17
	3	0	52	66	272

Table 8: Matriz de confusión CNN implementada

	0	1	2	3
Especificidad	99.22%	74.89%	73.22%	88.03%
Sensitividad	96.97%	87.56%	78.12%	69.74%

Table 9: Sensitividad y especificidad para cada una de las clases para la CNN

5 Modelo sensitivo a costos

5.1 Costos

El primer acercamiento al modelo de costos que se hizo fue usar la red neuronal que escogimos con validación cruzada en la primera sección y re-entrenarla teniendo en cuenta los costos de las clases. Los costos que se usaron fue '1' para las clases 0,1,3 & 100 para la clase 2 que es la que representa sueño profundo. Posteriormente se probo una SVM para este mismo problema.

5.2 Red neuronal con costos

Para la red neuronal sensible a costos se usaron los siguientes parámetros

Hiperparámetro	Valor
Optimizador	Stochastic descent gradient (SDG)
Función activación capa escondida	Relu
Función de pérdida	*Categorical cross entropy*
Función de activación capa de salida	*Softmax*
Número de iteraciones	1500
Número de capas escondidas	3
Número de neuronas por capa	250

Table 10: Hiperparámetros fijos en la red neuronal sensitiva a costos

Al entrenar la red neuronal obtuvimos la matriz de confusión que podemos ver en la Tabla 11. Lo que podemos ver es que como se está penalizando tan fuertemente el error para la clase (100 veces más fuerte que cometer error en cualquier otra clase) lo que está haciendo la red es decir es simplemente decir que todo pertenece el error para evitar penalización en la función que está minimizando. Esta red obtenida no sirve prácticamente para nada ya que simplemente retorna que todo pertenece a la clase de sueño profundo.

		Predicción			
		0	1	2	3
Etiqueta	0	0	0	304	0
	1	0	0	308	0
	2	0	0	315	0
	3	0	0	303	0
Especificidad		Na	Na	25.6%	Na
Sensibilidad		0	0	1	0

Table 11: Matriz confusión red neuronal con costos

5.3 SVM con costos

Como la red neuronal sensitiva a costos no dio los resultados esperados lo que se hizo fue tratar de crear un nuevo modelo que intentará tener en cuenta estos costos, se escogió una SVM usando la estrategia *one vs one*. Los hiperparámetros de esta SVM se muestran en la Tabla 12. Por otro lado el parámetro que se varió para la validación cruzada fue la constante de regularización que tomo los siguientes valores: [1e-5, 0.1, 1, 25, 100]

Parámetro	Valor
Kernell	Sigmoide
Estrategia	One vs One
Pesos	{0:1 1:1 2:1 3:100}

Table 12: Hiperparámetros fijos de las SVMs implementadas

Finalmente, se calculó la matriz de confusión de los modelos entrenados la cual puede verse en la Figura 7. En esta tabla podemos ver que para un constaste de regularización de $1e-5$ obtenemos exactamente los mismos resultados que en la red neuronal, mientras que para las otras tenemos ligeras diferencias, sin embargo como norma

general todos los modelos intenta hacer lo mismo que hizo la red neuronal del item anterior: clasificar todo como la clase que mayor peso tiene en la red.

		Predicción Modelo C=1E-5				Predicción Modelo 2 C=0.1				Predicción Modelo 3 C=1				Predicción Modelo 4 C=25				Predicción Modelo 5 C=100			
		0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
Etiqueta	0	0	0	304	0	81	1	217	5	48	6	249	1	57	2	244	1	60	3	240	1
	1	0	0	308	0	13	3	288	4	6	5	295	2	6	7	288	7	6	4	292	6
	2	0	0	315	0	1	0	313	1	5	2	308	0	9	3	301	2	9	1	303	2
	3	0	0	303	0	6	9	273	15	4	12	286	1	4	11	279	9	4	12	281	6
Especificidad		Na	Na	Na	Na	80%	23%	29%	60%	76%	20%	27%	25%	75%	30%	27%	47%	76%	20%	27%	40%
Sensibilidad		0	0	1	0	27%	1%	99%	5%	16%	2%	98%	0%	19%	2%	96%	3%	20%	1%	96%	2%
Accuracy		25.6%				33.5%				29.43%				30.40%				30.32%			

Figure 7: Matrices de confusión SVM entrenadas

5.4 Metacost

Dado que no se pudo lograr un buen modelo entrenando la red neuronal o las máquinas vectoriales de soporte se optó por utilizar metacost para re-etiquetar las clases y re-entrenar modelos que pueden ayudarnos a de alguna manera regularizar el entrenamiento. Para está implementación se modificó una implementación ya existente. Se usó el modelo de la red neuronal entrenado en el punto uno de esta entrega, se re-entrenaron 5 redes iguales utilizando bagging con reemplazo. Posteriormente se optó por dos matrices de pesos para este problema. La primera matriz de costos por la que se optó se puede ver en la Tabla 13 penaliza el hecho de clasificar incorrectamente la clase 2, pero no que algo que no sea de la clase 2 sea penalizado. La segunda matriz de costos 14 también penaliza el hecho de que se clasifique como ”sueño profundo” clase 2, algo que no lo es.

		Real			
		0	**1**	**2**	**3**
Predicción	**0**	0	0.01	1	0.01
	1	0.01	0	1	0.01
	2	0.01	0.01	0	0.01
	3	0.01	0.01	1	0

Table 13: Matriz de costos 1

		Real			
		0	**1**	**2**	**3**
Prediccion	**0**	0	0.01	1	0.01
	1	0.01	0	1	0.01
	2	1	1	0	1
	3	0.01	0.01	1	0

Table 14: Segunda Matriz de costos

Adicionalmente, el número de iteraciones que se utilizó para entrenar los 5 modelos se utilizaron 100 iteraciones por cada uno.

5.5 Resultados

Se entrenaron estas dos configuraciones de *Metacost* y se encontró la matriz de confusión tanto para validación como en testeo, (la misma distribución usados en la primera sección).

5.5.1 Matriz de costos 1

La exactitud final del modelo en datos de validación fue de 65%, cuando se probó este modelo en datos de prueba se obtuvo la siguiente matriz de confusión que se muestra en la Tabla 15. Vemos que el algoritmo intenta hacer lo mismo que en las SVM, sin embargo regula mucho mejor los errores. El algoritmo clasifica el 50% de los datos que se le alimenta en esta clase en la clase de interés en un problema balanceado. Por otro lado, el 96% de las etiquetas del sueño profundo son bien clasificadas en la red, pero de todas los datos que el algoritmo indica que es sueño profundo 53% son falsos positivos, esto puede deberse a que no se está penalizando el hecho de que el algoritmo arroje falsos positivos, por lo que se procedió a utilizar la matriz de costos 2 que si penaliza los falsos positivos.

		Predicción			
		0	**1**	**2**	**3**
Etiqueta	**0**	259	8	43	2
	1	14	76	172	38
	2	1	6	292	5
	3	3	33	119	160

Table 15: Matriz de confusión metacost con la matriz de costos 1

5.6 Matriz de costos 2

Con la matriz de costos 2 se obtuvo una exactitud global total de 71.6%, 6% por encima del modelo anterior, adicionalmente el porcentaje de datos correctamente clasificado de esta clase fue de 73%, mucho menor que en el modelo anterior, sin embargo el porcentaje de falsos positivos que arroja el modelo para esta clase es del 26.15%, por lo que podemos confiar un poco más en los resultados de este algoritmo, aún así se procedió a encontrar una métrica que nos permitiera escoger entre estos dos modelos.

		Predicción			
		0	**1**	**2**	**3**
Etiqueta	**0**	294	13	2	3
	1	18	160	54	68
	2	13	40	223	28
	3	5	32	23	255

Table 16: Matriz de confusión apra el modelo de metacost con la matriz de costos 2

5.6.1 Curvas ROC

Como tenemos un dilema entre % de datos bien clasificados y % de falsos positivos se uso la curva ROC ya que esta nos da una medida cuantitativa relacionando estos dos parámetros. La curva de la ROC de los modelos entrenados en Metacost (Figura 8 y la Figura 9). Vemos que la curva para el segundo modelo tiene un área bajo la curva ligeramente mayor, esto quiere decir que en términos generales este modelo es mejor. En general es importante tener cuidado a la hora de definir pesos en problemas tanto desbalanceados como desbalanceados, ya que es probable que toque intoducir alguna regularización extra si se introducen pesos al modelo.

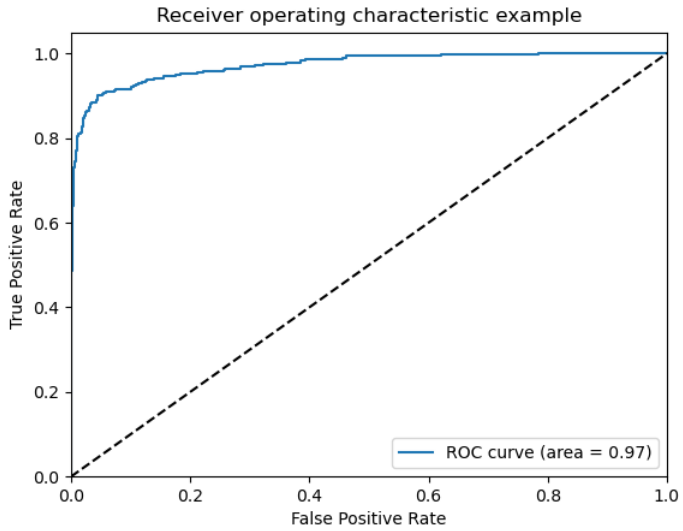

Figure 8: Curva ROC para la clase 2 (sueño profundo) para el modelo entrenado con Metacost y la matriz de costos 1

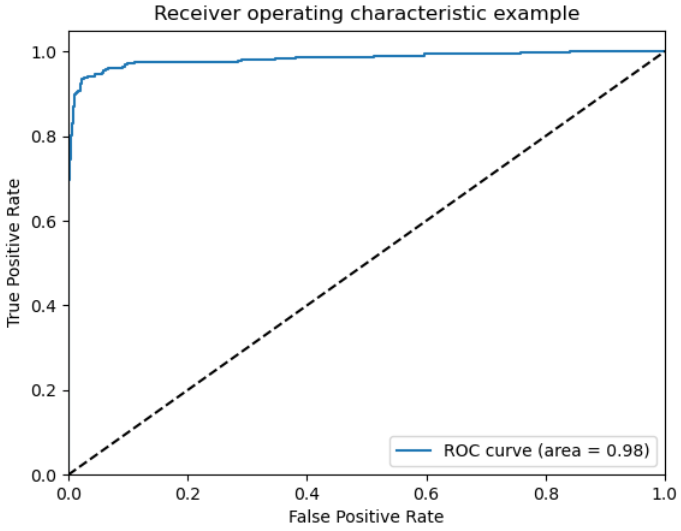

Figure 9: Curva ROC para la clase 2 (sueño profundo) para el modelo entrenado con Metacost y la matriz de costos 2

6 Anexos

6.1 Tablas

Clase	Datos totales	%
Sleep stage R	11,222	6.63%
Sleep stage W	115,455	68.17%
Sleep Stage 1	6,360	3.75%
Sleep Stage 2	29,050	17.15%
Sleep stage 3	4,106	2.42%
Sleep stage 4	2,241	1.32%
Sleep stage ?	863	0.51%
Movement time	78	0.05%
Total	169,375	100.00%

Table 17: Distribución de los datos

6.2 Códigos

Listing 1: Lectura de los archivos

```
import glob
import pyedflib as pyedf
import numpy as np
import csv

# Traer los archivos con los datos y las etiquedas
data1 = glob.glob("./data3/*-PSG.edf")
data2 = glob.glob("./data3/*-Hypnogram.edf")

if len(data1)!=len(data2):
    raise Exception('x should not exceed')

f = open("labels.txt", "w+")
for index in range(len(data1)):

    name1 = data1[index].split("\\")[1][0:6]
    name2 = data2[index].split("\\")[1][0:6]
    if(name1 != name2):
        print(name1, " ", name2)
        raise Exception(
            'El archivo de etiquetas no coincide con el archivo de datos')

    st_FileHypEdf = pyedf.EdfReader(data2[index])
    # vHypDur = Duración de cata étapa
    # v_HypTime = El tiempo
    # v_Hyp = Las etiquetas
    v_HypTime, v_HypDur, v_Hyp = st_FileHypEdf.readAnnotations()
    # Indice del tiempo en el que se va
    import pdb; pdb.set_trace();
    print("Tiempos")
    print(v_HypTime[:-1])
    print("Etiquetas")
```

```
print(v_Hyp)
indexOfTime = 0
st_FileEdf = pyedf.EdfReader(data1[index])
s_SigNum = st_FileEdf.signals_in_file
v_Signal_Labels = st_FileEdf.getSignalLabels()


if s_SigNum != 7:
    raise Exception(
        'Uno de los archivos tiene más señales muestreadas que lo esperado')

# La ventana de tiempo será de 30s
s_WinSizeSec = 30
# s_WinSizeSam = np.round(s_FsHz * s_WinSizeSec)

# Número de datos en la ventana por señal
# 3000
s_fs_30 = []
# 30
s_fs_3 = []

#Guardamos en un arreglo los indices de las señales
#que están muestreadas cada segundo
#Y en otro las que están muestreadas en 1/100 s
for largo in range(7):
    if st_FileEdf.getSampleFrequency(largo)*s_WinSizeSec == 3000:
        s_fs_30.append(largo)
    elif st_FileEdf.getSampleFrequency(largo)*s_WinSizeSec == 30:
        s_fs_3.append(largo)

#Para verificar
if(len(s_fs_30)+len(s_fs_3)) != 7:
    raise Exception('Cardinalidad o frecuencia no esperada')
s_FirstInd_30 = 0
s_FirstInd_3 = 0
j = 0
indiceTiempo = 0

#Leemos todas las señales porque está operación es costosa,
#se cargan todas a RAM de una vez
sign=[]
for p in range(7):
    sign.append(st_FileEdf.readSignal(p))

while 1:
    # Se llevarán dos indices, para las dos frecuencias de muestreo
    s_LastInd_30 = s_FirstInd_30 + 3000
    s_LastInd_3 = s_FirstInd_3 + 30

    sig = np.zeros((30, 7))
    for ind in range(7):
        if j*30 >= v_HypTime[-1]:
            break
```

```
            if ind in s_fs_3:
                sig[:, ind] = sign[ind][s_FirstInd_3:s_LastInd_3]

            if ind in s_fs_30:
                position = 0
                si = []
                while position < 3000:
                    si.append(sign[ind][position])
                    position += 100
                sig[:, ind] = si

        tiempo = j*30
        if(indiceTiempo+1 >= len(v_Hyp)):
            break

        next_step = v_HypTime[indiceTiempo+1]
        etiqueta = ""
        if(tiempo < next_step):
            etiqueta = v_Hyp[indiceTiempo]

        else:
            indiceTiempo = indiceTiempo+1
            etiqueta = v_Hyp[indiceTiempo]

        a = np.asarray(sig)
        n = data1[index].split("\\")[1][0:6]+str(j)+'.csv'
        np.savetxt(n, a, delimiter=',')
        f.write(n+","+etiqueta+"\n")
        f.flush()
        s_FirstInd_30 = s_LastInd_30
        s_FirstInd_3 = s_LastInd_3
        j = j+1
f.close()
```

Listing 2: Transfomación de los datos para la red neuronal

```
import glob
import numpy as np
import csv
import math
import scipy.fftpack as fourier
import matplotlib.pyplot as plt


data = glob.glob("./Datos_Finales/*")
f = open("./Transformacion/resumen1.csv", "w+")
clasePorArchivo = {}

with open("./labels.csv") as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        clasePorArchivo.update({row[0]: row[1]})

u=0
```

```
for datum in data:
    u=u+1
    data = np.genfromtxt(datum, delimiter=',')
    dato = ""
    n = datum.split("\\")[1]
    print(datum)
    isNan=False
    for index in range(6):
        cdatum = data[:, index]
        po = (1/30)*(cdatum*cdatum)
        pi = po/sum(po)
        pse = -sum(pi*np.log(pi))

        if math.isnan(pse):
            isNan=True
            break

        xf = fourier.fft(cdatum)
        xf = fourier.fftshift(xf)
        m = max(xf)
        index = np.where(xf == m)[0][-1]

        if index<15:
            index=30-index
        pf = 2*(index-15)*3.1416/30

        assert(pf>=0)
        maxi = np.max(cdatum)
        log = 10*math.log10(abs(maxi))

        dato += str(pse)+","
        +str(maxi)+","
        +str(pf)+","
        +str(max(cdatum))+","
        +str(min(cdatum))+","
        +str(np.mean(cdatum))+",
␣␣␣␣␣␣␣␣"+str(np.median(cdatum))+","
    if not isNan:
        clase = clasePorArchivo[datum.split("\\")[1]]+"\n"
        if clase == 'Sleep␣stage␣W\n':
            clase = "0"
        elif clase == "Sleep␣stage␣1\n" or clase =="Sleep␣stage␣2\n":
            clase = "1"
        elif clase == "Sleep␣stage␣3\n" or clase =="Sleep␣stage␣4\n":
            clase = "2"
        elif clase == "Sleep␣stage␣R\n":
            clase = "4"
        else:
            print(clase)
            continue
        f.write(dato+clase+"\n")
f.close()
```

Listing 3: Implementacion SVM

```
from sklearn.svm import SVC
from numpy import genfromtxt
from sklearn import preprocessing
import joblib
C = [0.00001]

X_train = genfromtxt('./datos_normalizados/X_train.csv', delimiter=',')
y_train = genfromtxt('./datos_normalizados/y_train.csv', delimiter=',')
X_validation = genfromtxt(
    './datos_normalizados/X_validation.csv', delimiter=',')
y_v = genfromtxt('./datos_normalizados/y_validation.csv', delimiter=',')


for c in C:
    print('pass')
    weights = {0: 1, 1: 1, 2: 1, 3: 5}
    model = SVC(kernel='sigmoid', probability=True, C=c,
                decision_function_shape='ovo', verbose=1, class_weight=weights)
    model.fit(X_train, y_train)
    predictions = model.predict(X_validation)
    print(predictions)
    filename = 'sigmoid_'+str(c)+'s.sav'
    joblib.dump(model, filename)
```

Listing 4: Implementación de la red convolucional

```
from numpy import genfromtxt, newaxis, savetxt, reshape, argmax
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Convolution2D
from keras.layers.convolutional import MaxPooling2D
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
import os
import glob
import csv
from numpy import genfromtxt
from keras.utils import to_categorical
import statistics
from sklearn import preprocessing


X_train_s = genfromtxt('./datos_normalizados_cnn/X_train.csv', delimiter=',')
y_train = genfromtxt('./datos_normalizados_cnn/y_train.csv', delimiter=',')
X_validation = genfromtxt('./datos_normalizados_cnn/X_validation.csv', delimiter=',')
y_v = genfromtxt('./datos_normalizados_cnn/y_validation.csv', delimiter=',')
```

```
X_train=reshape(X_train_s,(len(X_train_s),7,30,1))
X_validation=reshape(X_validation,(len(X_validation),7,30,1))

lb = preprocessing.LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_validation = lb.fit_transform(y_v)


model = Sequential()
model.add(Convolution2D(3, 3, 1, border_mode='valid',
input_shape=(7, 30, 1),activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
#model.add(MaxPooling2D(pool_size=(1, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(150, activation='relu'))
model.add(Dense(150, activation='relu'))
model.add(Dense(150, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(4, activation='softmax'))
  # Compile model
model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

model.fit(X_train, y_train, batch_size=128,
epochs=100, validation_data=(X_validation, y_validation))
y_p = model.predict(X_validation,batch_size=128)
y_pred = argmax(y_p, axis=1)
mc=confusion_matrix(y_v,y_pred)
print(mc)
model.save("cnn.h5")
```

6.3 Resultados

Tasa aprendizaje	# Neuronas por capa	Función pérdida	Exactitud total	Número capas escondidas
0.1	125	0.76853481	70%	1
0.1	150	0.77129684	73%	1
0.1	200	0.76475693	73%	1
0.1	250	0.84807317	74%	1
0.0001	125	0.92969949	60%	1
0.0001	150	0.92355872	59%	1
0.0001	200	0.93081923	60%	1
0.0001	250	0.92549814	59%	1
0.001	125	0.74256539	67%	1
0.001	150	0.74779473	66%	1
0.001	200	0.74832246	67%	1
0.001	250	0.74774394	67%	1
0.001	125	0.68570519	70%	2
0.001	150	0.68563137	71%	2
0.001	200	0.67452249	71%	2
0.001	250	0.66937352	70%	2
0.1	125	1.55358865	71%	2
0.1	150	1.78412354	72%	2
0.1	200	1.72169746	72%	2
0.1	250	1.91240732	73%	2
0.0001	125	0.88537592	61%	2
0.0001	150	0.89421842	61%	2
0.0001	200	0.88359395	62%	2
0.0001	250	0.873343	63%	2
0.1	125	2.73896208	73%	3
0.1	150	3.36209821	72%	3
0.1	200	3.27779314	74%	3
0.1	250	3.04173394	74%	3
0.0001	125	0.81274862	65%	3
0.0001	150	0.80657085	64%	3
0.0001	200	0.79618478	65%	3
0.0001	250	0.80741939	65%	3
0.001	125	0.65226128	74%	3
0.001	150	0.66102263	74%	3
0.001	200	0.65085894	73%	3
0.001	250	0.65807508	73%	3

Table 18: Resultados de la validación cruzada para la red neuronal implementada