BCI_TP3

October 1, 2022

Diplomatura en Ciencia de Datos, Aprendizaje Automático y sus Aplicaciones

Edición 2022

Mentoría: Data Science aplicado a BCI

Grupo 2

Integrantes: Gastón Briozzo, Pablo Ventura

Profesor de Práctico: Juan Manuel Lopez

1 Extracción de Características y Armado de Datasets

1.1 A) Introducción:

1.1.1 a) Familiarizarse con la clase BCIDataset.

Para la resolución de este problema, haremos uso de la clase BCIDataset. Ésta nos permite seleccionar la información correspondiente al paciente, la sesión y el canal deseados, y que transformaciones aplicarle a dicha información (transformada de Fourier, filtro de frecuencias, etc.). Una vez realizado este proceso, la clase nos facilita la extracción de la información mas relevante a través de los "features", valores característicos y representativos de los datos analizados que nos facilitan el análisis y la comprensión de estos.

De esta forma, la clase BCIDataset resulta una herramienta sumamente útil para el estudio de este problema.

1.1.2 b) Estudiar cómo varía el número de ejemplos en el dataset y la dimensión de cada dato según la variación de la ventana de tiempo seleccionada y el criterio de solapamiento.

La clase BCIDataset toma la serie temporal con todos los datos seleccionados y la divide para su análisis en segméntos de un mismo largo. Este largo es llamado tamaño de ventana, o windows size.

Al mismo tiempo, el segmento siguiente no comienza inmediatamente después de la finalización del anterior, sino que estos se solapan, compartiendo información. El porcentaje de información que comparten ambos segmentos es llamado fracción de solapamiento, o overlapping fraction.

Analicemos que influencia tienen estos parámetros. Lo que buscamos es maximizar tanto la cantidad como la calidad de la información que podemos extraer de los datos.

Para maximizar la cantidad, debemos aumentar el número de muestras, es decir, reducir el windows size e incrementar la overlaping fraction. El problema con esto es que un windows size pequeño

contiene muy pocos datos, por lo que no contendrá información útil. A su vez, al reducir la overlaping fraction sube la correlación entre segmentos, es decir, contienen la misma información, por lo que tener muchos deja de ser útil.

Para maximizar la calidad de los datos, debemos aumentar el tamaño de cada muestra, incrementando el windows size, y reducir la overlaping fraction. El inconveniente de hacer esto es que toda la información para a estar contenida en pocos segmentos, perdiendo variedad, con lo que podríamos pasar por alto detalles mas finos.

La solución está, como siempre, en el balance. Debemos encontrar los valores de windows size y overlaping fraction que maximicen la información resultante.

También debemos considerar que la elección de estos parámetros tendrá un impacto en el tamaño del dataset resultante.

[ ]:
```
csvs_path = 'data'

dataset1 = BCIDataset(csvs_path, subject = "AA", session = "0")

print('Dimensiones de ventanas de dataset1:', dataset1.get_X_signal().shape)
print('Dimensiones de features de dataset1:', dataset1.get_X_features().shape)

dataset2 = BCIDataset(csvs_path, subject = "AA", session = "0", window_size =␣
↪450)

print('Dimensiones de ventanas de dataset2:', dataset2.get_X_signal().shape)
print('Dimensiones de features de dataset2:', dataset2.get_X_features().shape)

dataset3 = BCIDataset(csvs_path, subject = "AA", session = "0",␣
↪overlapping_fraction=1/4)

print('Dimensiones de ventanas de dataset3:', dataset3.get_X_signal().shape)
print('Dimensiones de features de dataset3:', dataset3.get_X_features().shape)

dataset4 = BCIDataset(csvs_path, subject = "AA", session = "0",␣
↪window_size=1800, overlapping_fraction=1/15)

print('Dimensiones de ventanas de dataset4:', dataset4.get_X_signal().shape)
print('Dimensiones de features de dataset4:', dataset4.get_X_features().shape)
```

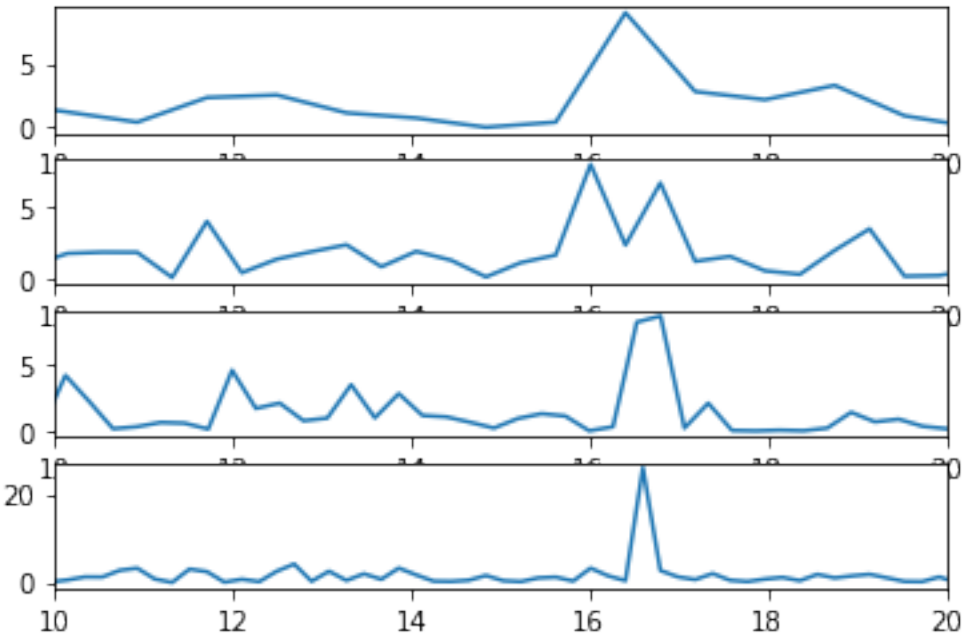
```
Processing subject: AA - session: 0...
Dimensiones de ventanas de dataset1: (151, 3600)
Dimensiones de features de dataset1: (151, 1804)
Processing subject: AA - session: 0...
Dimensiones de ventanas de dataset2: (304, 1800)
Dimensiones de features de dataset2: (304, 904)
Processing subject: AA - session: 0...
Dimensiones de ventanas de dataset3: (201, 3600)
Dimensiones de features de dataset3: (201, 1804)
```

```
Processing subject: AA - session: 0...
Dimensiones de ventanas de dataset4: (361, 7200)
Dimensiones de features de dataset4: (361, 3604)
```

1.2 B) Características Temporales:

1.2.1 a) Influencia del tamaño de la ventana en el dominio de tiempo en la resolución en frecuencia del espectrograma de potencia.

Veamos que efecto tiene el tamaño del windows size en la resolución resultante del espectrograma. En la siguiente figura, el windows size incrementa mientras mas abajo se encuentre la imagen. Las imágenes corresponden, de arriba a abajo, a windows size de 256, 512, 750 y 1024.



Esta claro que la resolución de frecuencias en el espectrograma de potencias aumenta conforme lo hace el tamaño de la ventana.

Esto se debe al hecho de que, al realizar la transformada de Fourier sobre un conjunto discreto de datos, la información total se conserva. De esta forma, si nuestro conjunto original contaba con N datos, su transformada de Fourier contará también con N datos.

Ahora bien, la transformada de Fourier de una función en el conjunto de los números reales da como resultado una función compleja (parte real y parte imaginaria), por lo que solo la mitad de los datos nos serán de interés (solo estudiaremos la parte real). Por otro lado, sabemos que para la frecuencia 0 las contribuciones real e imaginaria son idénticas.

Por lo tanto, siendo N par (impar) el número original de datos, la parte real de la transformada contará con N/2 + 1 (N/2+1/2) datos, equiespaciados entre la frecuencia 0 y la frecuencia de Niquist.

Es decir, la resolución del espectrograma crece linealmente con el tamaño de la ventana.

1.2.2 b) ¿Cuál considera que es el número adecuado de muestras temporales que puede recortar conservando la mayor cantidad de información útil en el dominio de la frecuencia?

Intentaremos encontrar ahora el tamaño de ventana óptimo para nuestro problema.

La frecuencia de muestreo es de $200Hz$ (1/200 segundos).

Las estipulaciones tienen una duración aproximada de 10 segundos.

De esta forma, cada estipulación cubre aproximadamente 2000 muestras.

Si consideramos un windows size superior a 2000, estaríamos señales provenientes de distintos estímulos, por lo que 2000 es una cota superior del windows size.

Por otro lado, si consideramos un windows size similar a 2000, las muestras que no estén sincronizadas con el estímulo también estarán mezclando información. De esta forma, la cota superior del windows size debe ser algún número claramente inferior a 2000.

Respecto a la resolución, las frecuencias que obtenemos están equiespaciadas entre $0Hz$ y $100Hz$, siendo aproximadamente $N/2$, por lo que la resolución será de $100Hz/(N/2) = 200/NHz$.

Dado que debemos poder distinguir claramente entre los $12.5Hz$ y los $16.5Hz$, una cota inferior para la resolución sería $(16.5Hz - 12.5Hz)/10 = 0.4Hz$, lo que nos deja una cota inferior para el windows size de $N = 500$.

Con esto en mente, consideramos que el windows size óptimo esta al rededor de 1024

1.2.3 c) Defina una estrategia de extracción de atributos en el dominio de tiempo que opere sobre la serie cruda.

La serie temporal cruda no brinda por si sola información que permita identificar cual es la estimulación a la que esta siendo expuesto el paciente. Para resolver este problema, debemos procesar los datos y extraer información relevante.

Con este objetivo, proponemos extraer las propiedades estadísticas de la serie temporal tales como valores mínimo y máximo, media y desviación estándar. Estos valores serán empleados mas adelante para intentar predecir el estado del paciente

[ ]:
```
def features_stats(signal_matrix):
    st_fila = []
    for fila in signal_matrix:
        mi  = min(fila)
        ma  = max(fila)
        me = statistics.mean(fila)
        sd = statistics.stdev(fila)
        s = [mi, ma, me, sd]
        st_fila.append(s)
    return st_fila
```

1.2.4 d) Guarde los datasets generados de la forma que considere conveniente.

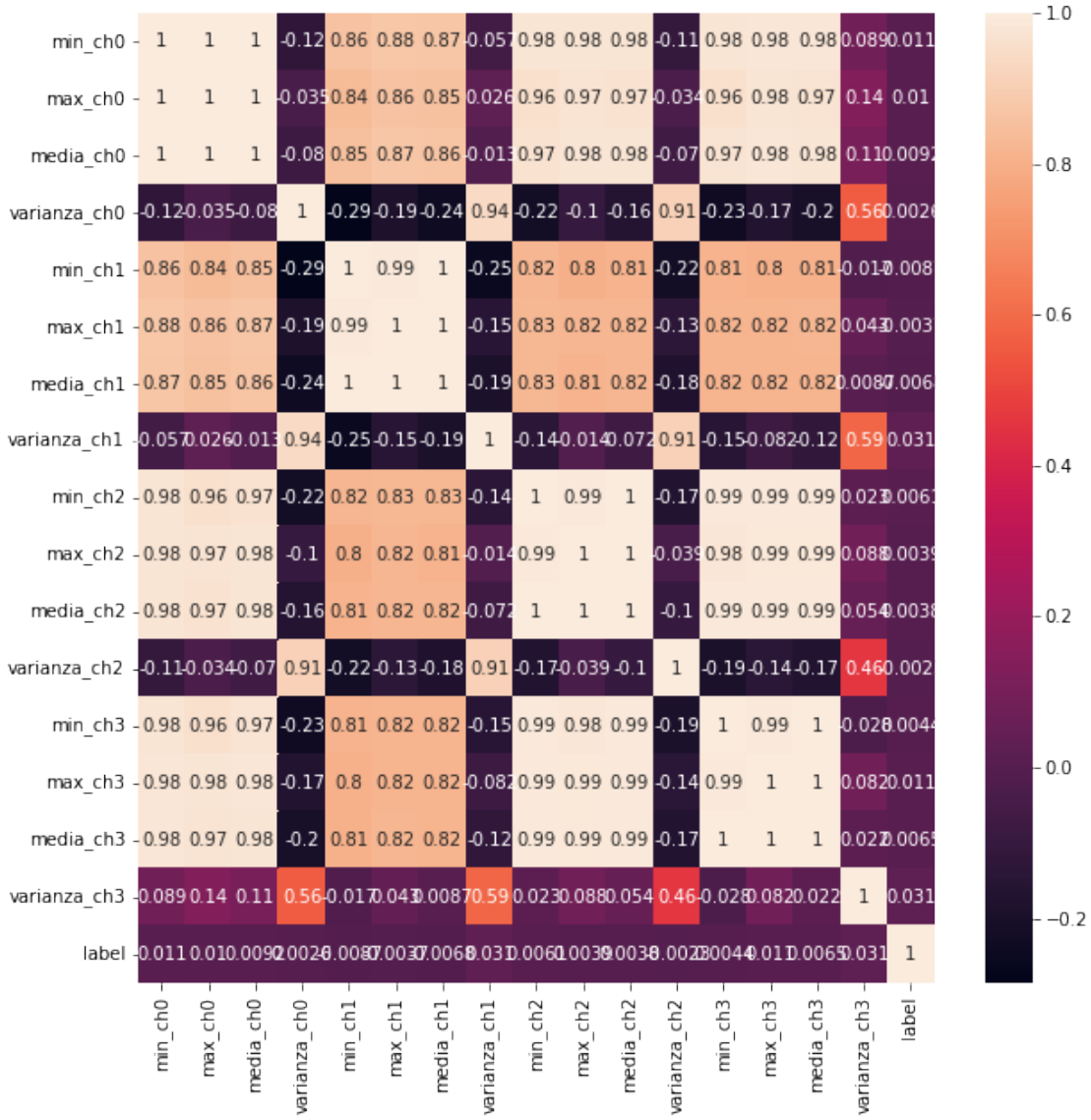
[ ]:
```
def ds_to_csv(ds,path):
  df = pd.DataFrame(data=np.hstack((ds.get_X_features(),ds.get_metadata(),ds.
↪get_Y())),
                    index=None,
                    columns=["min_ch0", "max_ch0", "media_ch0", "varianza_ch0",
                             "min_ch1", "max_ch1", "media_ch1", "varianza_ch1",
                             "min_ch2", "max_ch2", "media_ch2", "varianza_ch2",
                             "min_ch3", "max_ch3", "media_ch3", "varianza_ch3",
                             "pureza", "tiempo0", "tiempof", "sujeto",␣
↪"sesion",
                             "label"]
                    )
  objects = {"pureza", "tiempo0", "tiempof", "sujeto", "sesion"}
  convert_dict=dict()
  for k in df.columns:
    if k not in objects:
      convert_dict[k] = float
    else:
      convert_dict[k] = object

  df = df.astype(convert_dict)

  df.to_csv(path)
  return df
```

1.2.5 e) Analice la contribución de información de cada feature. Estudie la correlación entre features. Estudie la correlación entre features y etiquetas.

En la siguiente tabla se puede ver la correlación de los estadísticos extraídos, tanto entre si como con la etiqueta (label) de la señal, que indica el estado del paciente.



Podemos ver que, si bien los estadísticos presentan una buena correlación entre sí, no tienen buena correlación con el label, por lo que difícilmente puedan emplearse para la tarea de predicción.

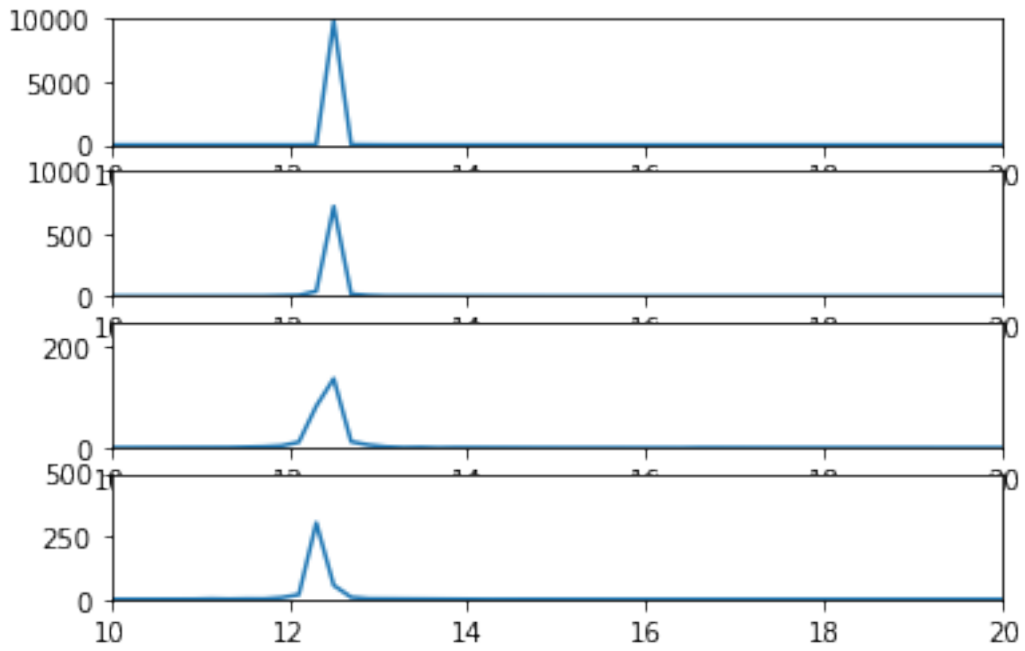
De esta forma, y tomando en cuenta los resultados obtenidos en el informe anterior, concluimos que los features extraídos de la serie temporal cruda resultan insuficientes para la predicción del estado del paciente.

1.3 C) Características Espectrales:

1.3.1 a) Espectrograma de potencia.

Como vimos en el informe anterior, la transformada de Fourier de la señal temporal presenta picos característicos en las frecuencias de estimulación, por lo que ésta resulta sumamente prometedora en la tarea de predicción.

A continuación mostramos el aspecto característico de la transformada.



Viendo estas imágenes resulta evidente que tiene un potencial predictivo mucho mayor al de la señal temporal cruda.

Es conveniente separar los datos correspondientes a distintos canales para facilitar su análisis.

1.3.2 b) Generar una estrategia de extracción de features en el dominio de la frecuencia.

Para mejorar la calidad de los features resultantes y mejorar su análisis, resulta conveniente realizar un preprocesamiento de los datos.

Para empezar, normalizaremos los datos. Tomaremos la señal temporal, le restaremos su valor medio y la dividiremos por su varianza. De esta forma, todos los conjuntos de datos están normalizados, por lo que arrojarán valores con magnitudes similares. Restar el valor medio sirve además para deshacernos de las frecuencias bajas, que no resultan de interés. Al dividir por la varianza, las potencias resultantes tendrán valores similares y será mucho mas fácil analizarlas.

Una vez normalizados los datos temporales, les realizaremos la transformada de Fourier. Para esto, aplicaremos un filtro de frecuencias, deshaciéndonos de aquellas que estén por debajo o por encima de la región de interés. De esta forma, solo consideraremos el espectro entre los 10 y los 19Hz.

[ ]:
```
def signal_filter(signal_matrix):

    for i in range(len(signal_matrix)):
        signal_matrix[i] = ( signal_matrix[i] - np.mean(signal_matrix[i]) )  /␣
↪np.std(signal_matrix[i])
```

```
    N = signal_matrix.shape[1]
    dt = 1/200
    T = N*dt
    sf = 200
    Q = 30
    f_notch = 50
    b_notch, a_notch = sgn.iirnotch(w0=f_notch, Q=Q, fs=sf)
    sig_notch = sgn.filtfilt(b_notch, a_notch, signal_matrix, axis=1)

    #Ahora creamos el filtro pasabanda Butterworth
    f_nq = sf/2
    f_low = 10
    f_high = 19
    order = 4
    b_band, a_band = sgn.iirfilter(
        N=order, Wn=[f_low/f_nq, f_high/f_nq], btype="bandpass", ftype="butter"
    )
    sig_filt = sgn.filtfilt(b_band, a_band, sig_notch, axis=1)

    fft = np.fft.rfft(sig_filt)
    Sxx = np.real(((2*dt**2)/T)*fft*fft.conj())

    return Sxx
```

Una vez que la señal este filtrada, procederemos a la extracción de features.

El primer feature que tomaremos será la frecuencia con mayor participación en la señal resultante (es decir, la frecuencia a la cual se da la máxima potencia). Llamaremos a esta frecuencia f_{max}. En general, esta está fuertemente correlacionada con el estado del paciente, indicando la frecuencia de estimulación.

[ ]:
```
def max_freq(signal_matrix):

    N = signal_matrix.shape[1]
    dt = 1/200

    Sxx = signal_filter(signal_matrix)
    freqs = np.fft.rfftfreq(n=N, d=dt)

    freqs = np.fft.rfftfreq(n=N, d=dt)
    fmax = freqs[Sxx.argmax()]

    return fmax
```

Sin embargo, la presencia de ruido puede generar que f_{max} no coincida con la frecuencia de estimulación, por lo que por si sola no ofrece suficiente información.

Definimos el peso ponderado de una frecuencia de interés f_i como la integral de las potencias $P(f)$ alrededor de dicha frecuencia, moduladas por una gaussiana centrada en f_i,

$p(f_i) = \frac{1}{p_{max}} \int P(f) exp(-2(f - f_i)^2) df.$

De esta forma, los pesos ponderados contienen información no solo de la potencia en frecuencia de interés $P(f_i)$, sino tambien información acerca de las potencias en torno a esta, disminuyendo la importancia de estas potencias conforme aumenta la distancia entre las frecuencias.

Para normalizar estos datos, dividimos los valores resultantes por el peso ponderado al rededor de la frecuencia de máxima participación.

$p_{max} = \int P(f) exp(-2(f - f_{max})^2) df.$

[ ]:
```
def p_freq(Sxx,freqs):

    p_125 = 0
    p_165 = 0
    p     = 0

    fmax = freqs[Sxx.argmax()]

    for i in range(len(freqs)):
      f = freqs[i]
      s = Sxx[i]
      p_125 += s*math.exp(-2*(12.5-f)**2)
      p_165 += s*math.exp(-2*(16.5-f)**2)
      p     += s*math.exp(-2*(fmax-f)**2)

    p_125 = p_125/p
    p_165 = p_165/p

    return p_125, p_165
```

Finalmente, definimos los valores relativos de una frecuencia de interés como la potencia de dicha frecuencia dividido por la potencia de la frecuencia de máxima participación.

$v(f_i) = \frac{P(f_i)}{P(f_{max})}.$

[ ]:
```
def v_freq(Sxx,freqs):

    v_125 = 0
    v_165 = 0
    v_max = 0

    for i in range(len(freqs)-1):
        f  = freqs[i]
        s  = Sxx[i]
        fn = freqs[i+1]
        sn = Sxx  [i+1]
        if (f == 12.5):
            v_125 = s
        if (f<12.5 and 12.5<fn):
```

```
            v_125 = ( (fn-12.5)*s + (12.5-f)*sn ) / (fn - f)
        if (f == 16.5):
            v_165 = s
        if (f<16.5 and 16.5<fn):
            v_165 = ( (fn-16.5)*s + (16.5-f)*sn ) / (fn - f)

    v_max = max(Sxx)

    v_125 = v_125/v_max
    v_165 = v_165/v_max

    return v_125, v_165
```

Una vez calculados los features, los extraemos todos mediante una misma función.

[ ]:
```
def pro_features(signal_matrix):

    N = signal_matrix.shape[1]
    dt = 1/200

    Sxx = signal_filter(signal_matrix)
    freqs = np.fft.rfftfreq(n=N, d=dt)

    vpm = []

    for sxx in Sxx:
        fmax = freqs[sxx.argmax()]
        p_125, p_165 = p_freq(sxx,freqs)
        v_125, v_165 = v_freq(sxx,freqs)

        vpm.append([v_125, v_165, p_125, p_165, fmax])

    return vpm
```

Esperamos que estos features sean suficientes para realizar con éxito la tarea de predicción.

1.3.3 c) Guarde los datasets generados de la forma que considere conveniente.

Los features generados son guardados en datasets para facilitar su utilización mas adelante.

[ ]:
```
def ds_to_csv(ds,path):
  df = pd.DataFrame(data=np.hstack((ds.get_X_features(),ds.get_metadata(),ds.
↪get_Y())),
                    index=None,
                    columns=["val_12.5_ch0", "val_16.5_ch0", "p_12.5_ch0",␣
↪"p_16.5_ch0", "f_max_ch0",
                             "val_12.5_ch1", "val_16.5_ch1", "p_12.5_ch1",␣
↪"p_16.5_ch1", "f_max_ch1",
```

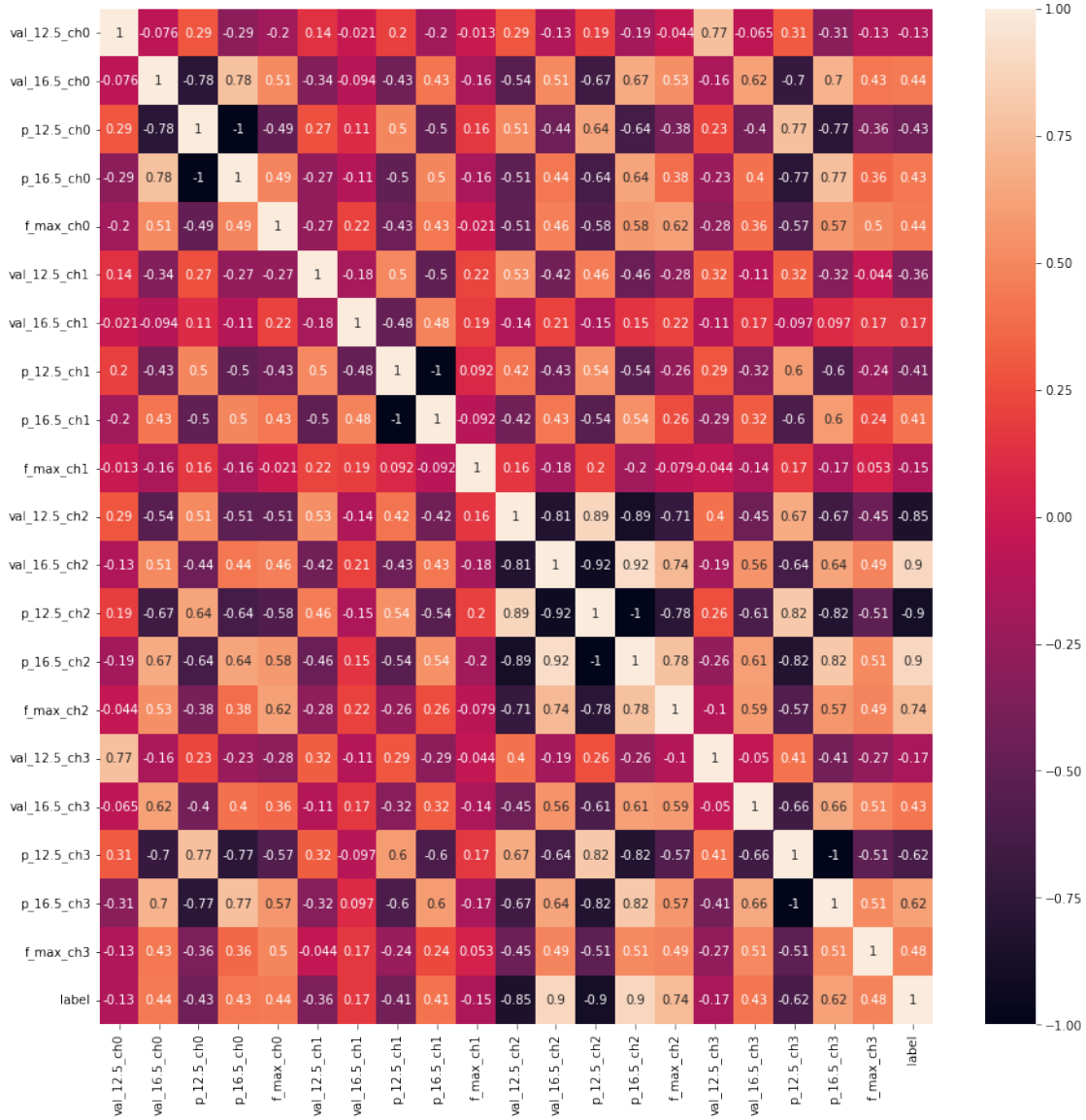
```
                                "val_12.5_ch2", "val_16.5_ch2", "p_12.5_ch2",␣
↪"p_16.5_ch2", "f_max_ch2",
                                "val_12.5_ch3", "val_16.5_ch3", "p_12.5_ch3",␣
↪"p_16.5_ch3", "f_max_ch3",
                                "pureza", "tiempo0", "tiempof", "sujeto",␣
↪"sesion",
                                "label"]
                    )
 objects = {"pureza", "tiempo0", "tiempof", "sujeto", "sesion"}
 convert_dict=dict()
 for k in df.columns:
   if k not in objects:
     convert_dict[k] = float
   else:
     convert_dict[k] = object

 df = df.astype(convert_dict)

 df.to_csv(path)
 return df
```

1.3.4 d) Correlación entre features y labels.

Veamos ahora como se correlacionan los featurs extraídos con el estado del paciente.



En la tabla anterior podemos ver que los features no solo presentan una muy buena correlación entre sí, sino que ademas tienen una alta correlación con el label. Esto sugiere fuertemente que son buenos atributos para llevar adelante la tarea de predicción.

1.3.5 e) Analice la distribución del dataset, número de ejemplos por clase, balance entre clases, número de ejemplos "puros" y ejemplos "impuros".

Al analizar el dataset resultante, vemos que existe una buena proporción en los datos generados entre distintas etiquetas y en la pureza de estas.

[ ]:
```
ds6_df.label.value_counts()
```

[ ]:
```
2.0     31
1.0     30
99.0    26
```

```
Name: label, dtype: int64
```

[ ]:
```
ds6_df.pureza.value_counts()
```

[ ]:
```
True     46
False    41
Name: pureza, dtype: int64
```

1.4 D) Particionado del dataset:

[ ]:
```
data_all = BCIDataset(csvs_path, overlapping_fraction=1/4, window_size= 1300,␣
↪feature_extractor=pro_features)
```

1.4.1 a) Esquema de particionado.

Una vez generado el dataset, procedemos a dividirlo en conjuntos de entrenamiento, evaluación y validación.

Decidimos emplear un 80% de los datos en el conjunto de entrenamiento, mientras que los datos restantes se dividiran en partes iguales de un 10% para los conjuntos de evaluación y validación. Para esta desición nos basamos en las proporciones estandar que se suelen tomar en este tipo de problemas.

Es posible ver que cada conjunto contiene proporciones similares de datos correspondientes a cada clase y propociones similares de pureza, con lo que concluimos que los datos están bien valaceados.

La influencia del solapamiento ya fue discutida anteriormente. Este genera que exista cierta correlación entre los datos, pero permite también incrementar el número de muestras.

Consideraremos para el dataset de predicción los datos correspondientes a todos los pasientes, en todas sus sesiones y de todos sus canales, para contar con el conjunto mas amplio y heterogeneo posible.

2 Aprendizaje automático

El objetivo de este trabajo práctico no es evaluar y comparar diferentes algoritmos de clasificación, sino entender el problema de clasificación en su conjunto y definir el esquema general para trabajar en él. En ese sentido:

2.1 Entendimiento del problema

2.1.1 Está claro que hasta aquí hablamos de un problema de clasificación supervisada, pero ¿a qué subclase dentro de ellos pertenece? (multilabel? multiclase?)

Se trata de un problema multiclase, ya que se clasifican en clases disjuntas dependiendo del tipo de estimulacion.

2.1.2 En palabras, describa cómo podría plantearse un problema de regresión con los datos estudiados.

Por ejemplo, se podria tomar el feature que implica la probabilidad de que la estimulacion sea de tipo 1 y tipo 2 y tomarlos como puntos de un plano y aplicar una regresion lineal para buscar dividir los puntos en las dos clases.

2.1.3 De la misma forma, ¿cómo podría pensarse un problema de clasificación no supervisada a partir de los datos disponibles?

En este caso se pueden tomar todos los features y analizarlas utilizando el algoritmo K-Means para dividirlas en dos grupos, buscando que se correspondan con la clase 1 y 2 de estimulacion.

2.2 En el problema de clasificación supervisada:

2.2.1 ¿Qué métricas considera que son apropiadas para evaluar el desempeño de algún algoritmo de clasificación?

Para evaluar el desempeño de los algoritmos utilizamos la matriz de confusion y el f1-score.

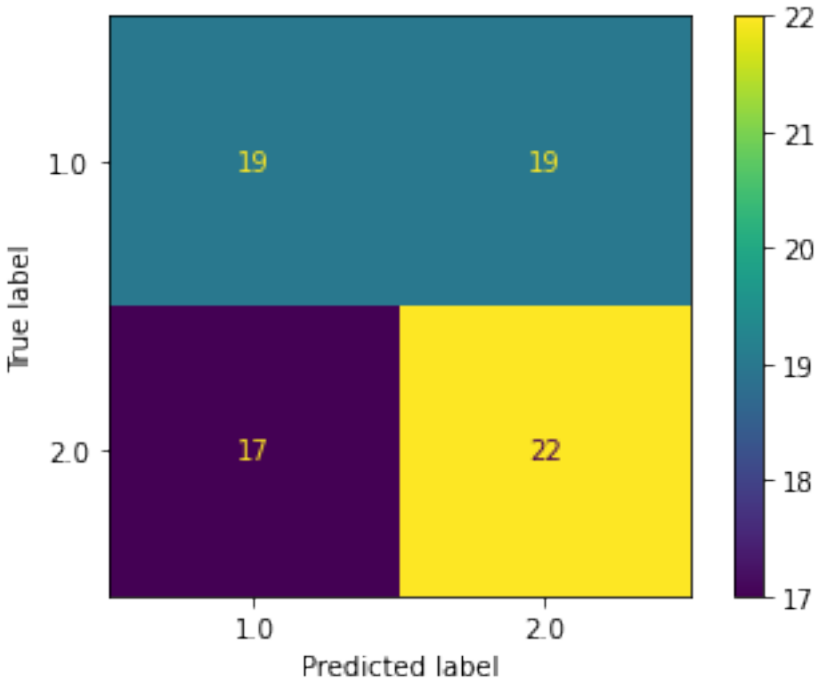
2.2.2 ¿Qué funciones de costo consideran apropiadas para entrenar un algoritmo para este problema?

Dado que cada funcion de costo, se comportan distinto en cada algoritmo, utilizamos las funciones de costo estandar para cada algortmo.

2.3 Seleccione un algoritmo de aprendizaje supervisado estudiado en la Diplomatura.

2.3.1 Elija uno de los datasets generados previamente.

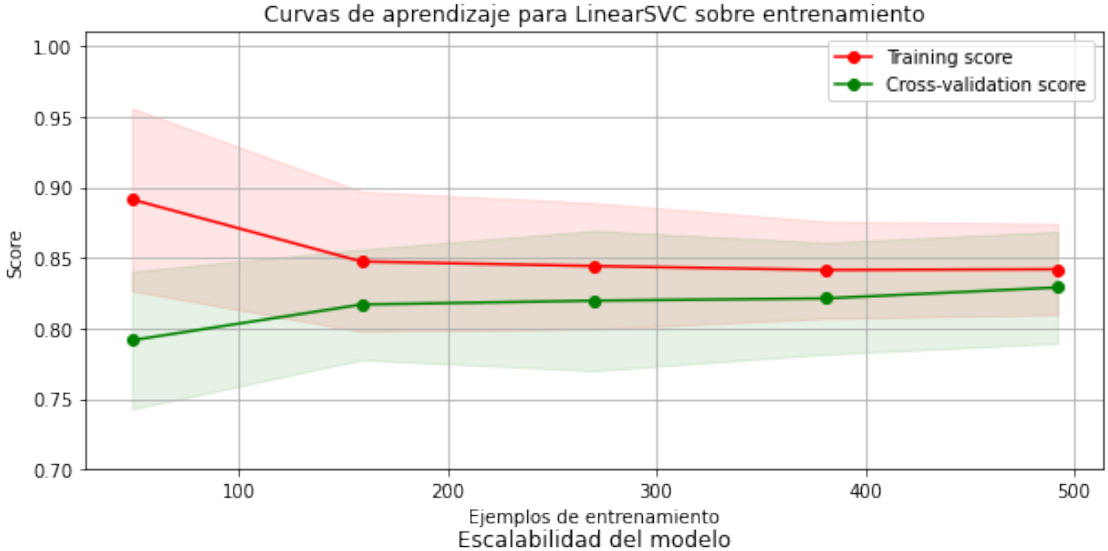
2.3.2 Determine un benchmark de performance usando una asignación aleatoria de etiquetas.

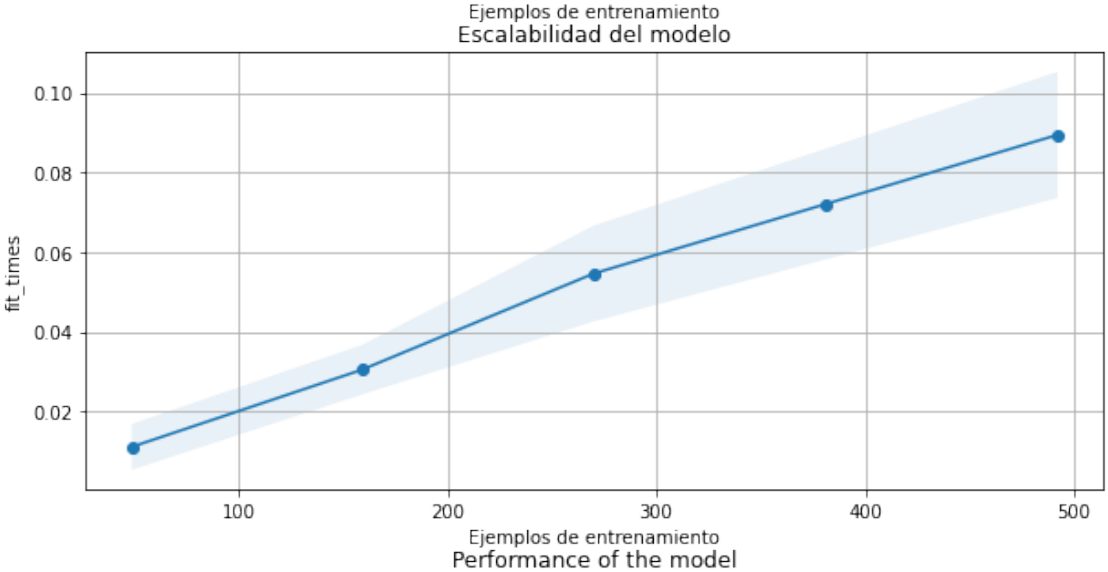


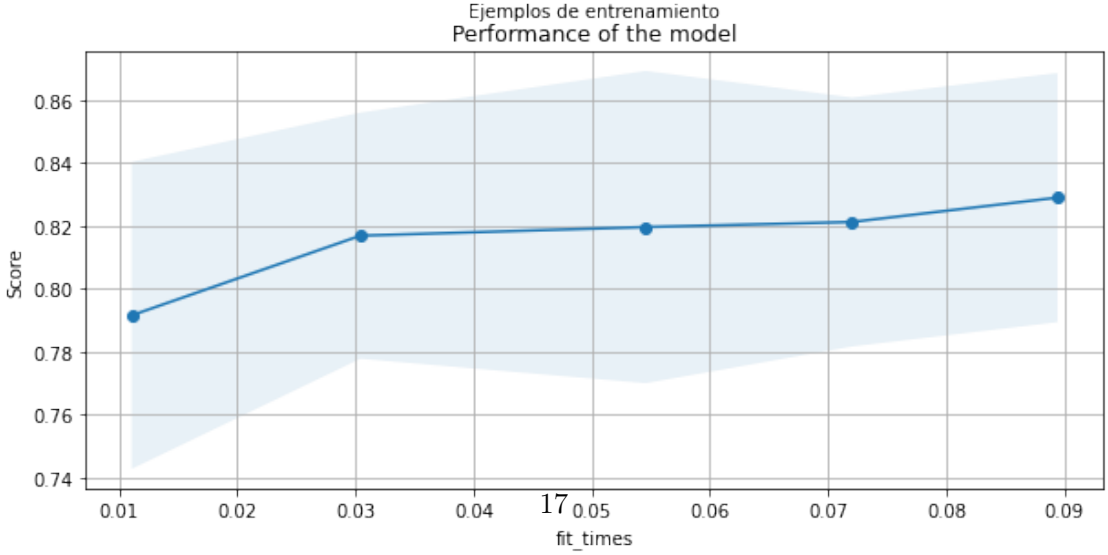
	precision	recall	f1-score	support
1.0	0.50	0.42	0.46	38
2.0	0.51	0.59	0.55	39
accuracy			0.51	77
macro avg	0.51	0.51	0.50	77
weighted avg	0.51	0.51	0.50	77

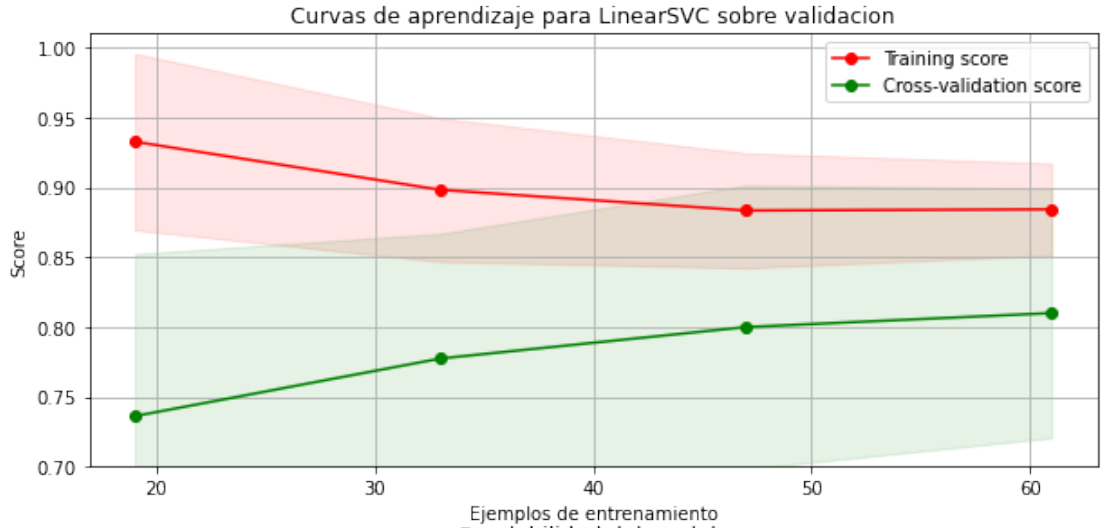
2.3.3 Entrene el algoritmo seleccionado, (no invertir mucho tiempo en el ajuste de hiperparámetros).

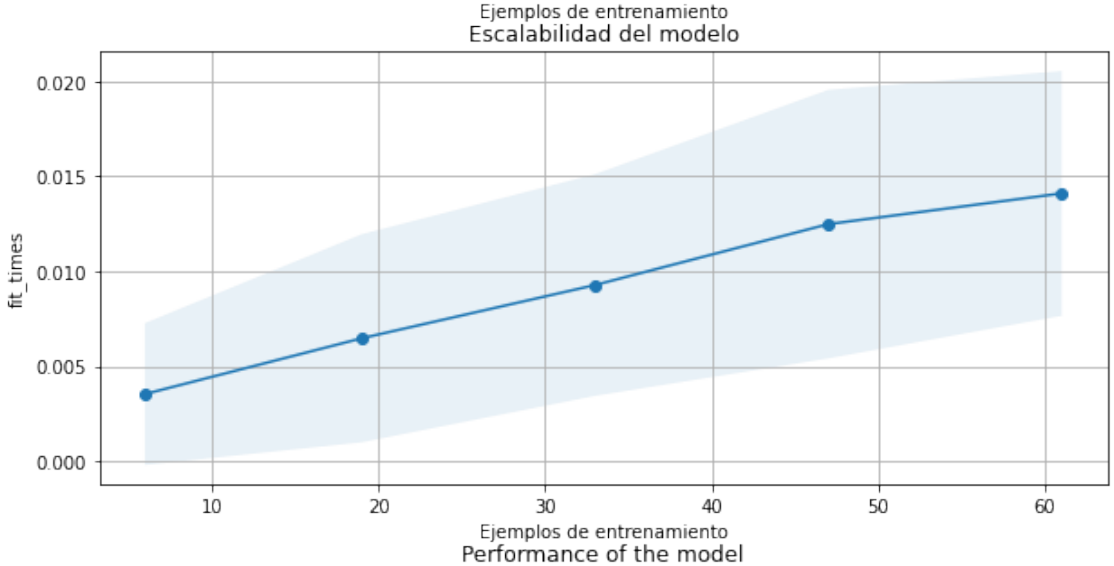
2.3.4 Genere curvas de la evolución de las métricas relevantes a lo largo del entrenamiento (desempeño y loss), tanto para el conjunto de entrenamiento como de validación. Analice la presencia de los fenómenos de overfitting, underfitting, y su relación con el bias y variance.

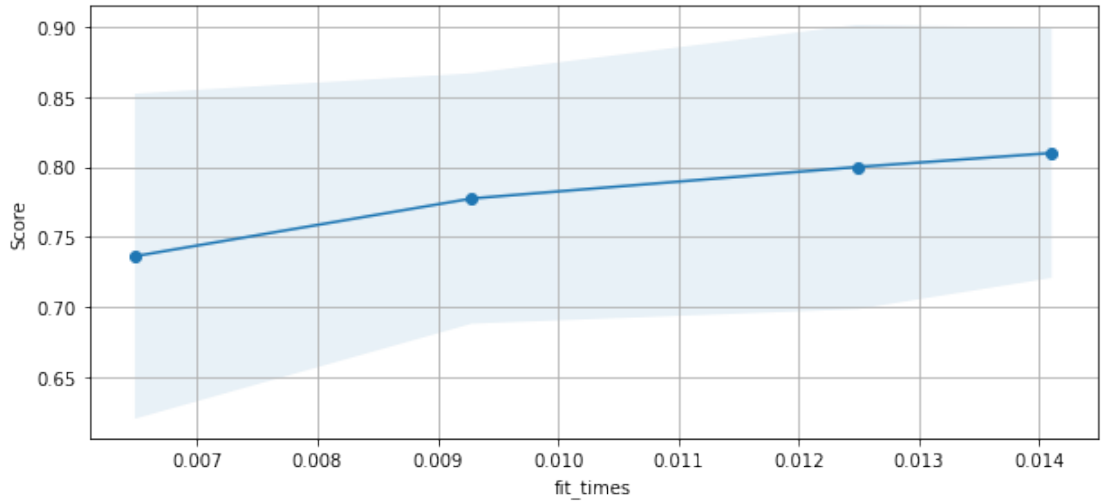


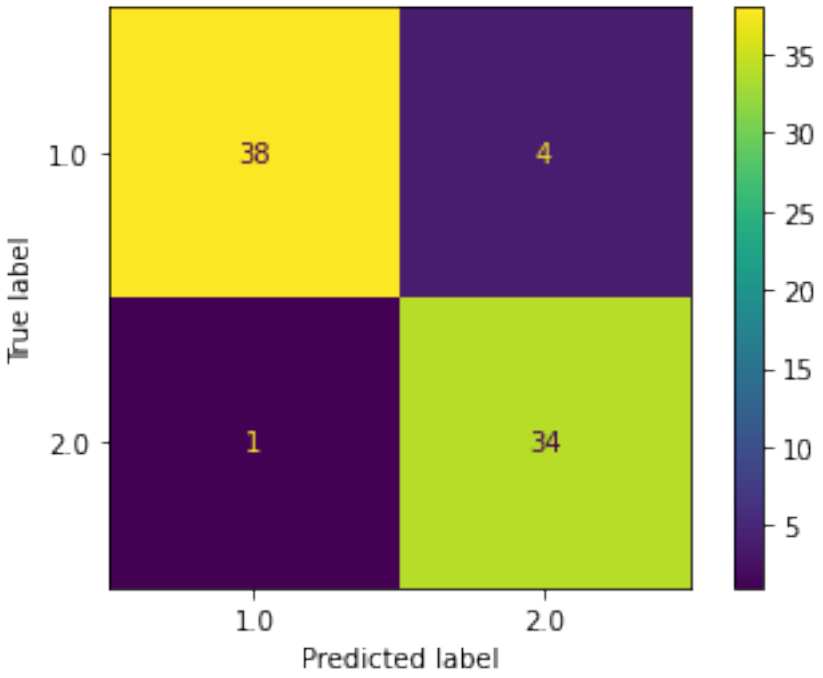





Curvas de aprendizaje para LinearSVC sobre validacion
Training score
Cross-validation score
Score
Ejemplos de entrenamiento

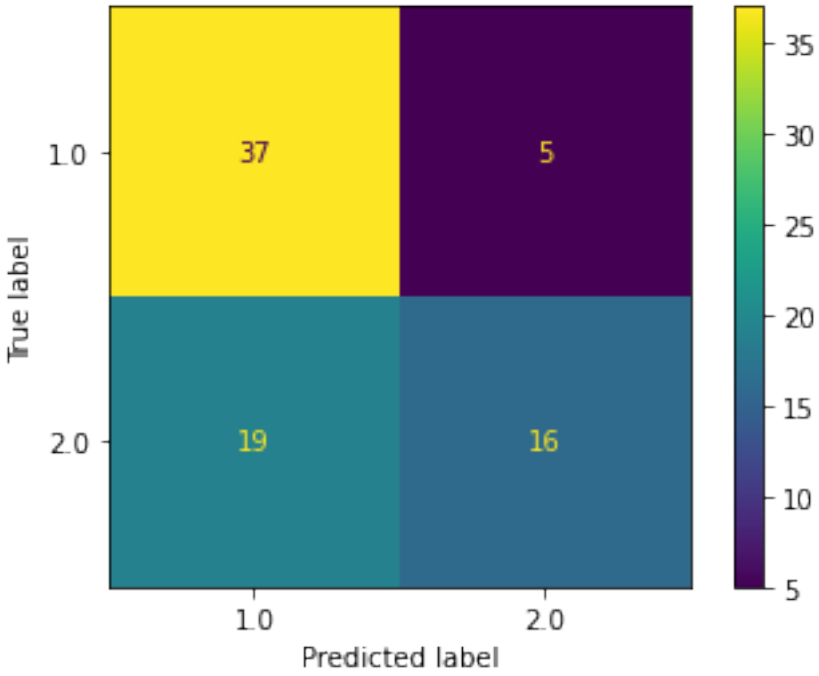

Escalabilidad del modelo
fit_times
Ejemplos de entrenamiento


Performance of the model
Score
fit_times



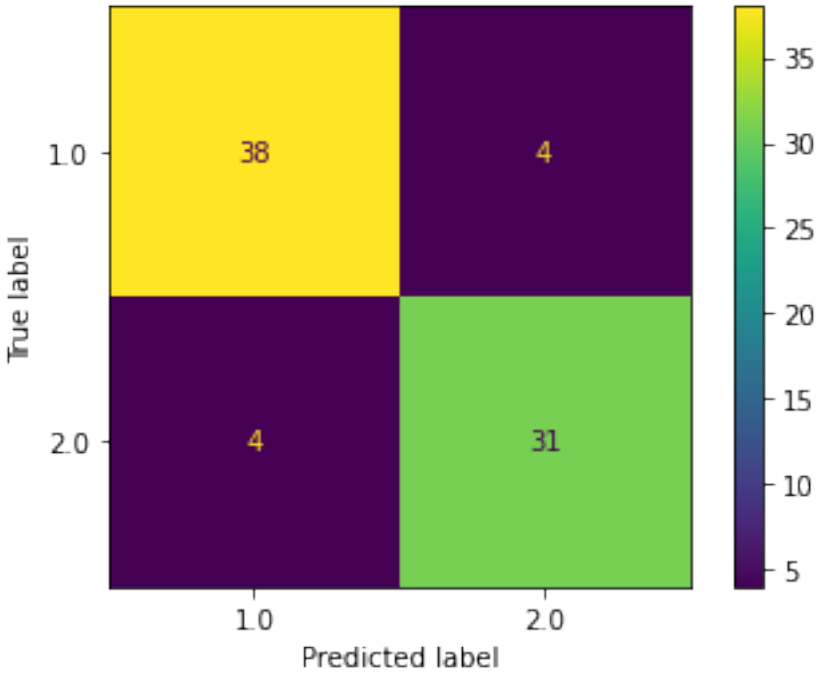
	precision	recall	f1-score	support
1.0	0.97	0.90	0.94	42
2.0	0.89	0.97	0.93	35
accuracy			0.94	77
macro avg	0.93	0.94	0.93	77
weighted avg	0.94	0.94	0.94	77

2.3.5 K-means



	precision	recall	f1-score	support
1.0	0.66	0.88	0.76	42
2.0	0.76	0.46	0.57	35
accuracy			0.69	77
macro avg	0.71	0.67	0.66	77
weighted avg	0.71	0.69	0.67	77

2.3.6 Random forest GBDT



	precision	recall	f1-score	support
1.0	0.90	0.90	0.90	42
2.0	0.89	0.89	0.89	35
accuracy			0.90	77
macro avg	0.90	0.90	0.90	77
weighted avg	0.90	0.90	0.90	77

2.3.7 Sobre el conjunto de test, utilice las métricas definidas en el apartado B-a) para determinar el desempeño del algoritmo. Compárelo con el benchmark de desempeño.

Es evidente que el algoritmo LinearSVC es mucho mejor que nuestro benchmark ya que el f1-score en el algoritmo Dummy da 0.51 mientras que con LinearSVC da 0.90. Por otro lado en las dos matrices de confusion se nota claramente que el algoritmo Dummy acierta el 50% de las veces mientras que LinearSVC acierta en la gran mayoria de las veces.

2.3.8 Teniendo en cuenta lo observado en el apartado C), entrene el mismo algoritmo en todos los datasets generados previamente (5 en total). Use siempre el mismo esquema de particiones (los 5 datasets deberían ser diferentes features de los mismos ejemplos en el dominio del tiempo).

2.3.9 Compare el desempeño del algoritmo sobre el conjunto de test.

2.3.10 ¿Qué influencia tienen los features elegidos sobre el algoritmo utilizado?

Los features a elegir tienen mucha influencia ya que necesitamos encontrar features que esten fuertemente correlacionados con las clases a clasificar.