Departamento de Ingeniería Eléctrica
Facultad de Ciencias Físicas y Matemáticas
Universidad de Chile
EL5202-2 Laboratorio de Sistemas Digitales

Informe Módulo 4

Programación de Support Vector Machine

Integrantes: Diego Campanini
Eduardo Hormazábal
Profesor: Helmuth Thiemer
Auxiliar: Carlos Navarro
Ayudante: Rodrigo Maureira
Fecha: 26 de noviembre de 2015

Índice

Índice de figuras

Índice de códigos

1. Introducción

El objetivo de este módulo es implementar el algoritmo de aprendizaje supervisado *Support Vector Machine* (SVM), el cuál se encarga de buscar un hiperplano, que separe elementos de dos clases distintas, para esto se puede recurrir a un aumento de la dimensionalidad del conjunto original, lo anterior se realiza a través de una función llamada Kernel, la cual suele ser una función polinomial, tangente hiperbólica o gaussiana.

Producto que SVM separa dos clases, se utilizará una cascada de este clasificador, ya que, se buscan identificar 6 clases. La idea es implementar este algoritmo en un Arduino Due, mencionar que se ha escogido este clasificador porque resulta más rápido que por ejemplo una red neuronal.

Para implementar el SVM en Arduino se realizará el entrenamiento de este clasificador en un computador, lo anterior para determinar el hiperplano de separación de forma previa y luego ingresar el clasificador ya entrenado a la plataforma Arduino. El correcto funcionamiento del SVM será comprobado mediante la comparación con el *toolbox* que Matlab posee para este algoritmo.

2. Marco Teórico

2.1. Explicación Support Vector Machine

En el contexto de aprendizaje el *Suppor Vector Machine* es un método que permite la clasificación de clases. En términos generales el SVM encuentra un hiperplano que separa dos clases, tal que el margen de separación se maximice. Lo anterior se aplica a casos linealmente separables y se extiende para los que no lo son. A continuación se describe el método para ambos casos.

2.1.1. Patrones Linealmente Separables

Para separar las dos clases se puede definir la ecuación del hiperplano que las separa, la cual se expresa como sigue:

$$\mathbf{w}^t \cdot \mathbf{x} + b = 0 \tag{1}$$

Donde $\mathbf{w}$ es un vector normal al hiperplano (vector de pesos), $\mathbf{x}$ es un vector de entrada y b es el bias que corresponde a un valor constante. Para un cierto $\mathbf{w}$ y b, la separación entre el hiperplano y el conjunto de datos más cercano se denomina *margen*. Cuando los datos son no separables se pueden seleccionar dos planos tal que entre ellos no se encuentre ningún dato, para seleccionar toda la porción de los datos de una clase o la otra, estos dos planos se pueden definir como:

$$\mathbf{w}^t \cdot x + b \geq 1 \tag{2}$$

$$\mathbf{w}^t \cdot x + b \geq -1 \tag{3}$$

Luego por geometría se puede determinar que la distancia entre los dos planos definidos por las igualdades de las ecuaciones (2) y (3) está dada por $\frac{2}{||w||}$, por lo tanto, para maximizar el margen se tiene que minimizar $\mathbf{w}$, además se puede obtener que la distancia desde el origen al hiperplano óptimo, dado por la ecuación (1), es $\frac{b}{||w||}$. En la figura (1) se pueden observar los hiperplanos con línea segmentada que definen el margen y el hiperplano óptimo, para un problema de dos clases.

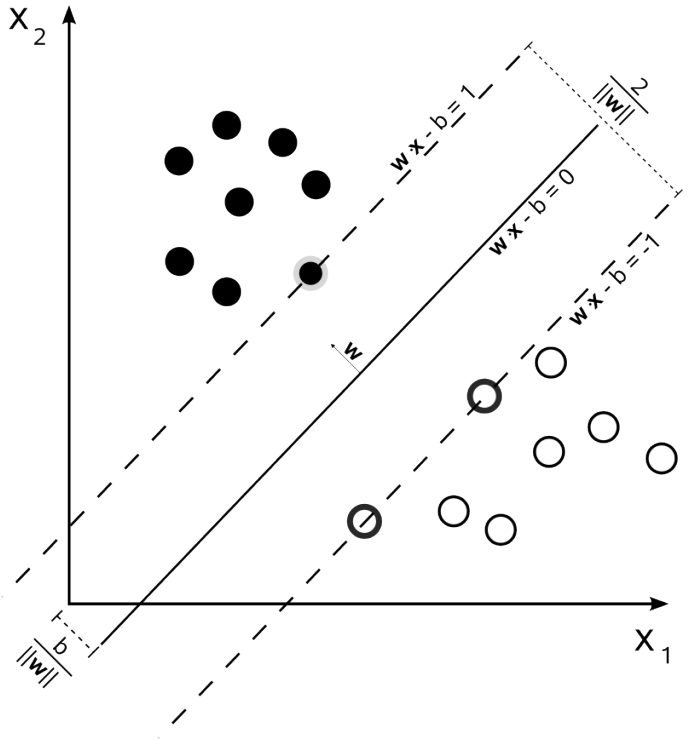


Figura 1: Separación de dos clases (círculos y circunferencias) mediante hiperplanos

En el problema de minimización antes expuesto se puede cambiar sin alterar el resultado $||w||$ por $\frac{||w||^2}{2}$, donde el $\frac{1}{2}$ se agrega por conveniencia matemática, además la minimización de la expresión $\frac{||w||^2}{2}$ está restringida a cumplir con $y_i(\mathbf{w}^t\mathbf{x}_i + b) \geq 1$, donde y_i puede tomar valores de +1 o -1 dependiendo a qué clase pertenezca. En virtud de lo antes expresado este es un problema de optimización con restricciones y se puede resolver utilizando los multiplicadores de Lagrange, entonces la función lagrangeana para el problema, se puede expresar como:

$$\mathscr{L}(\mathbf{w}, b, \alpha_i) = \frac{\mathbf{w}^t\mathbf{w}}{2} - \sum_{i=1}^{N} \alpha_i[d_i(\mathbf{w}^t\mathbf{x}_i + b) - 1] \tag{4}$$

Luego aplicando las condiciones de optimalidad se pueden obtener los parámetros del clasificador ($\mathbf{w}$ y b) de la siguiente forma:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i d_i \mathbf{x}_i \;\; con \;\; \alpha_i \neq 0 \tag{5}$$

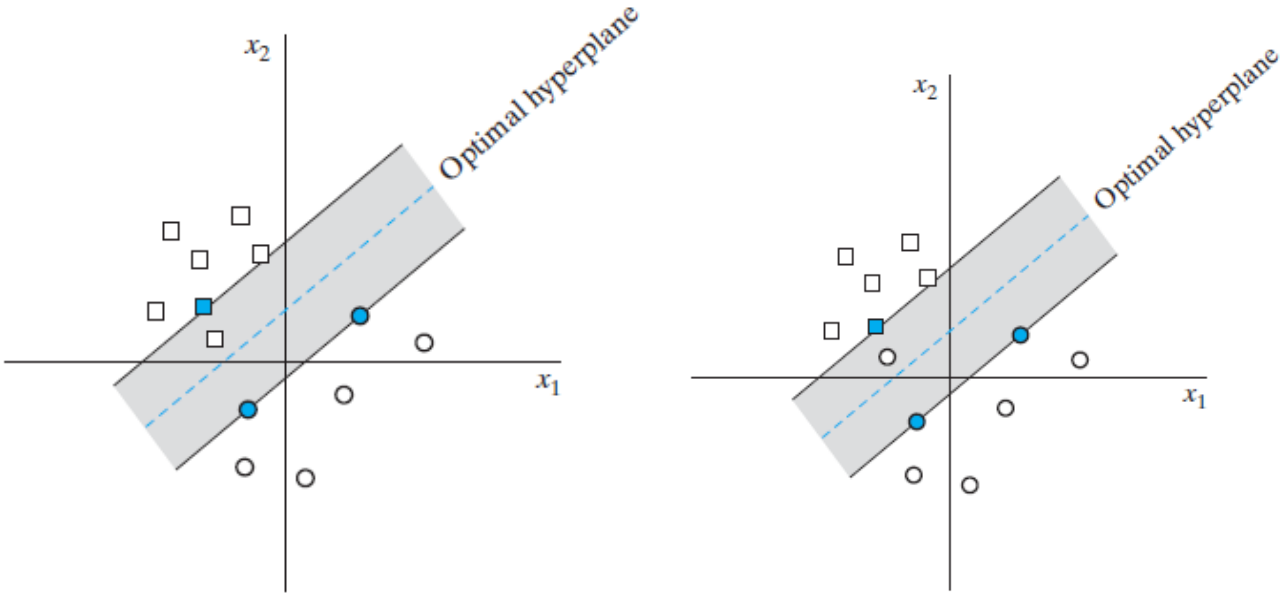
$$b = 1 - \sum_{i=1}^{N} \alpha_i d_i \mathbf{x}_i^t \mathbf{x} \tag{6}$$

2.1.2. Patrones No Linealmente Separables

En el caso en que los patrones no sean linealmente separable, no se podrá encontrar un hiperplano que separe de forma exacta los datos, por lo tanto, en este caso se busca encontrar un hiperplano tal que se minimice la probabilidad de clasificación errónea. Para tratar con estos datos mal clasificados, se introduce en la ecuación del hiperplano de decisión, una variable escalar no negativa denominada ξ , que recibe el nombre de *slack variables*, entonces la ecuación de la superficie de decisión se expresa como:

$$d_i(\mathbf{w^t x_i} + b) \geq 1 - \xi_i \qquad i = 1, 2, ..., N \tag{7}$$

La varaible ξ_i mide el grado de clasificación errónea, para $0 < \xi_i \leq 1$ los datos caen dentro de la región de separación y en el lado correcto, como se aprecia en la figura (2) (a), por el contrario cuando $\xi > 1$ los datos caen en el lado incorrecto de la separación, como se observa en la figura (2) (b)



(a) Datos mal clasificados dentro del margen pero en el lado correcto

(b) Datos mal clasificados dentro del margen pero en el lado incorrecto

Figura 2: Región de decisión caso no separable

En este casos los vectores de soporte son aquellos que satisfacen la ecuación (7), incluso pueden existir vectores que cumplan con $\xi_i = 0$. El funcioanal que se desea minimizar con respecto a $\mathbf{w}$ se puede escribir como sigue:

$$\Phi(\mathbf{w}, \xi_i) = \frac{\mathbf{w^t w}}{2} + C \sum_{i=1}^{N} \xi_i \tag{8}$$

En la ecuación (8) el parámetro C controla la compensación entre la maximización del margen y la correcta clasificación de los datos. Si C se escoge como un valor grande quiere decir que existe una mayor penalización de los errores, mencionar que el segundo término de la ecuación (8) se relaciona con el error

de clasificación. La forma dual del problema se puede expresar de la siguiente forma:

$$Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j \mathbf{x_i^t x_j} \tag{9}$$

$$\text{Sujeto a:} \sum_{i=1}^{N} \alpha_i d_i = 0 \tag{10}$$

$$0 \leq \alpha_i \leq C \tag{11}$$

En el problema dual anterior, lo que se busca determinar son los multiplicadores de Lagrange $\{\alpha_i\}_{i=1}^{N}$, tal que se maximice la función objetivo $Q(\alpha)$, el conjunto de α_i definen el vector de soporte buscado.

2.1.3. Aumento de dimensionalidad utilizando un kernel

Destacar otro aspecto presente en el SVM el cual se relaciona con la idea que el espacio original de características puede ser trasladado a un espacio de mayor dimensión, donde el problema sí sea separable, esta traspaso de una dimensión a otra se realiza por una función denominada *Kernel*, dentro de las más comunes destacan la polinomial, gaussiana y tangente hiperbólica.

Si $\mathbf{x}$ es el vector que representa los datos de entradas al clasificador, que pertenecen a un espacio de dimensión m_0, entonces se denota $\varphi(\mathbf{x})$ como una función no lineal que aumenta la dimensión del espacio de entrada. Dada esta transformación se puede definir un hiperplano actuando como la superficie de decisión, lo cual se escribe de la siguiente forma:

$$\sum_{j=1}^{\infty} w_j \varphi_j(\mathbf{x}) = 0 \tag{12}$$

El término $\{w_j\}_{j=1}^{\infty}$ en la ecuación(12), denota un conhjunto de vectores pesos infinitamente largos, que transforman las características del espacio de entrada al espacio de salida. La ecuación (12) se puede escribir matricialmente como:

$$\mathbf{w}^T \phi(\mathbf{x}) = 0 \tag{13}$$

La ecuación (5) se puede escribir de la siguiente forma:

$$\mathbf{w} = \sum_{i=1}^{N_s} \alpha_i d_i \phi(\mathbf{x}_i) \tag{14}$$

En la ecuación anterior N_s es el número de vectores de soporte y el vector de características es expresado como $\phi(\mathbf{x}_i) = [\varphi_1(\mathbf{x}_1), \varphi_2(\mathbf{x}_2), ...]^T$. Usando las ecuaciones (13) y (14), se obtiene lo siguiente:

$$\sum_{i=1}^{N_s} \alpha_i d_i \phi^T(\mathbf{x}_i) \phi(\mathbf{x}_i) = 0 \tag{15}$$

El término $\phi^T(\mathbf{x}_i)\phi(\mathbf{x}_i$ define un producto punto, además este es el denominado Kernel de SVM, en virtud de los anterior, se tiene que el Kernel está dado por:

$$k(\mathbf{x}, \mathbf{x}_i) = \phi^T(\mathbf{x}_i)\phi(\mathbf{x}_i) \tag{16}$$

$$= \sum_{j=1}^{\infty} \varphi(\mathbf{x}_i)\varphi(\mathbf{x}) \quad \text{con } i = 1, 2, ..., N_s \tag{17}$$

Finalmente para encontrar los elementos del vector de soporte que permite maximizar el margen de separación entre clases, se obtiene de resolver el problema dual de la sección anterior, dado por las ecuaciones (9), (10) y (11). La única diferencia es que en la ecuación (9) el término $\mathbf{x}_i^t \mathbf{x}_j$, se reemplaza por $\mathbf{k}(\mathbf{x}_i, \mathbf{x}_j)$.

Para ilustrar el beneficio de un aumento de la dimensión de los datos de entrada, se muestra la figura 3, en la cual en primera instancia los datos de las dos clases (puntos rojos y azules) se ubican sobre una recta , donde no es posible separar los datos por un hiperplano lineal, sin embargo, al trasladarse a un espacio de dos dimensiones, mediante un kernel cuadrático, se obtiene que las clases son separables fácilmente por un hiperplano lineal.

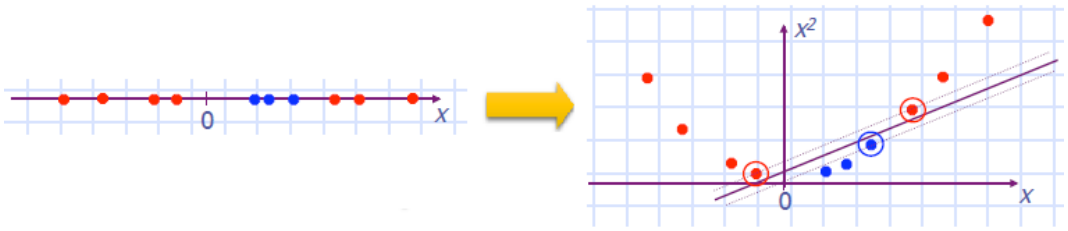


Figura 3: Efecto del aumento de dimensionalidad para lograr trabajar en un espacio donde las clases sean fácilmente separables

Para la implementación de SVM, se determinarán los vectores de soporte en un computador, debido a la capacidad de cálculo, es decir, el proceso de entrenamiento no se realizará en el Arduino Due. Una vez obtenidos los vectores de soporte se implementará el clasificador en el Arduino. Se pretende utilizar un kernel gaussiano para el aumento de dimensionalidad, debido a su efectividad, para trabajar con señales provenientes de electromiogramas. La base de datos para realizar el entrenamiento se obtendrá con el circuito implementado en el módulo 2.

3. Implementación de SVM

Para generar un correcto sistema de clasificación mediante SVM, se desarrolló en Matlab un *script* que recibe las series de tiempo de diferentes movimientos, le calcula las características consideradas previamente, construye el clasificador y retorna sus parámetros. Esto se testeó con la base de datos utilizados en [1].

En la Figura 4 se visualizan dos muestras correspondientes a diferentes movimientos a clasificar, pero que pertenecen al par de movimientos más similar entre todos (movimiento cilíndrico y esférico). Cabe destacar que a simple vista no es posible distinguir una gran diferencia entre ambos. Sin embargo al extraer las características de éstos y junto con su posterior separación mediante SVM se obtiene un error de clasificación del 6,7 %, para un total de 60 muestras.

Los parámetros generados por el clasificador corresponden a un Kernel Gaussiano de la forma:

$$k_\sigma(\mathbf{x},\mathbf{y}) \quad = \quad e^{-\frac{(\mathbf{x}-\mathbf{y})(\mathbf{x}-\mathbf{y})^T}{2\sigma^2}} \tag{18}$$

donde el ancho de banda encontrado es $\sigma = 2{,}1659$. Además para una mejor generalización, las características se normalizaron de forma de tener media 0 y varianza unitaria.

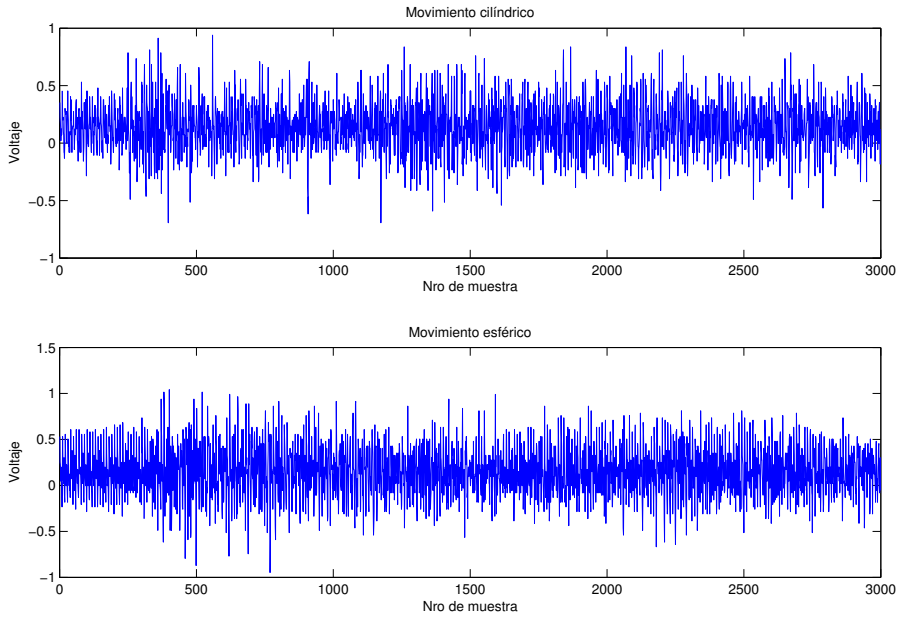


Figura 4: Muestra de movimientos de la base de datos utilizada en [1]

El proceso de entrenamiento y verificación se muestra en el Código fuente 1, el que sería extendible para cualquier base de datos que siga la estructura utilizada. En particular, se podrán entregar los datos muestreados con el Arduino para construir el clasificador de acuerdo a los movimientos reales a utilizar.

Código fuente 1: Entrenamiento y Clasificación mediante SVM.

```
%% Cargar datos
male1 = load('male_1.mat');
%% Movimientos
mov1 = male1.cyl_ch1;
mov2 = male1.spher_ch1;
%% Obtener caracteristicas
c_mov1 = get_caract(mov1);
c_mov2 = get_caract(mov2);
%% Juntar muestras y asignar clases (1: mov1, -1: mov2)
data = [c_mov1;c_mov2];
class = ones(size(data,1),1);
class(size(c_mov1,1)+1:size(data,1)) = -1;
%% Entrenar modelo
model = fitcsvm(data,class,'KernelFunction','rbf',...
    'BoxConstraint',Inf,'ClassNames',[1,-1],...
    'Standardize',true,'KernelScale','auto');
%% Resultados clasificacion fallida
CVmodel = crossval(model);
misclass = kfoldLoss(CVmodel);
misclass
%% Clasificar una muestra
muestra = data(11,:);
score = get_score(muestra,model.SupportVectors,...
    model.SupportVectorLabels,model.KernelParameters.Scale,...
    model.Alpha,model.Sigma,model.Mu,model.Bias);
clase = sign(score)
```

Como el clasificador logra un porcentaje de clasificaciones correctas superior al 90 % para las clases más similares, se infiere que es posible construir múltiples clasificadores conectados en cascada para decidir a cuál de todos los movimientos corresponde la serie de tiempo. Esto se implementará en el código Arduino final.

4. SVM con datos de Arduino

Se muestrearon 4 movimientos diferentes, con 15 muestras para cada uno. La tasa de muestreo es de 500 muestras por segundo, con un total de 1024 muestras. Los movimientos son los siguientes:

- Mano quieta.
- Apretar puño.
- Sostener peso con brazo horizontal.
- Apretar cilindro.

En la Figura 5 se observa una muestra para cada movimiento. De la misma forma que en la sección anterior, si bien existen movimientos distinguibles a simple vista (cerrar puño), la mayoría se confunde entre ellas

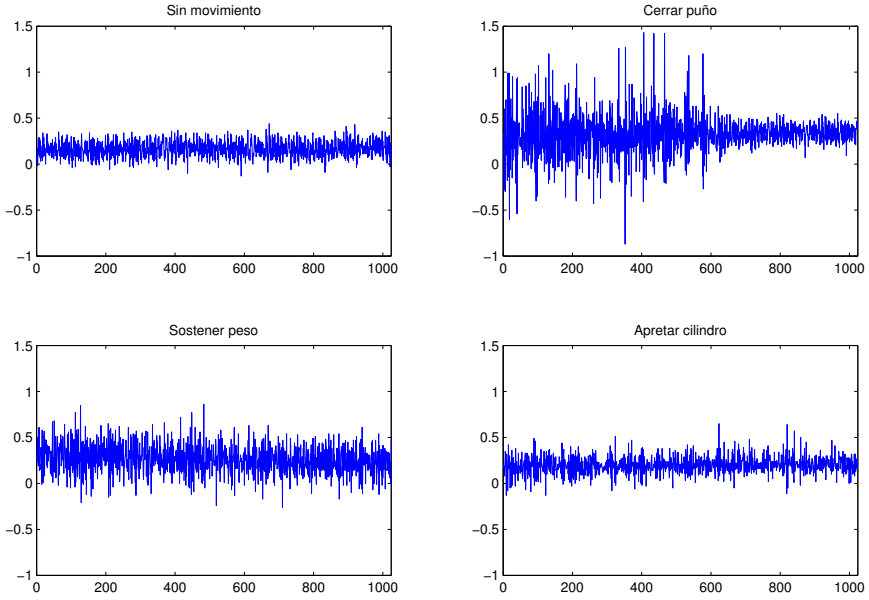


Figura 5: Muestra de movimientos obtenidos con Arduino

Al clasificar cada movimiento uno-contra-uno se obtuvieron los errores mostrados en el Cuadro 1. Como se observa el mayor error ocurre entre los movimientos .ªpretar puñoz "Sostener peso", el que alcanza un 20 % de clasificaciones erróneas. Sin embargo se debe recordar que solamente se está utilizando 1 canal, por lo que aún no se han extraído las características totales a utilizar en el proyecto.

Por otro lado, se observa que las demás tasas de error de clasificación no superan el 10 %, lo que es un buen indicador para el único canal utilizado. Debido a esto y a los resultados de la sección anterior, se verifica que conectar clasificadores uno-contra-uno en cascada es una solución factible para el problema

de múltiples clases, que es lo que se buscaba en este módulo.

Finalmente, sólo basta transcribir el código implementado en Matlab al Arduino. Esto se hace directamente pues el Arduino solamente recibe los parámetros del clasificador (generados en Matlab), sin la necesidad de usar alguna librería adicional.

Cuadro 1: Tasas de error de clasificación entre clases

	Mano quieta	**Apretar puño**	**Sostener peso**	**Apretar cilindro**
Mano quieta	0	0.0690	0.0230	0.0294
Apretar puño	0.0690	0	0.2000	0.0303
Sostener peso	0.0230	0.2000	0	0.0571
Apretar cilindro	0.0294	0.0303	0.0571	0

5. Conclusión

Algunas consideraciones importantes sobre el algoritmo SVM, es la cantidad de memoria que se requiere para almacenarlo, ya que, se tiene que considerar la cantidad de memoria para guardar los vectores de soporte encontrados por este algoritmo, la señal se entrada y sus características. Se utilizará para la implementación Arduino Due, que cuenta con 512KB de memoria para almacenamiento de código

Con respecto a la calidad del clasificador, nos encontramos satisfechos pues solamente se está utilizando 1 canal de adquisición de datos, y un selecto conjunto de características. Si bien el clasificador se podría hacer aún más riguroso considerando más características y canales, el construido en este módulo resulta ser más rápido y de menor almacenamiento, que es uno de los objetivos a perseguir.

Otra consideración tiene relación con la rapidez de respuesta del clasificador, puesto que, idealmente se requiere una respuesta, desde que ingresa la señal desde los electrodos, unas fracciones de segundos después. Para esto se deberá procurar optimizar el código, de tal forma de operar eficientemente.

Así mismo, cabe destacar que se están realizando clasificaciones sobre un conjunto de 1024 datos, muestreados a 500 muestras por segundo. Esto restringe el tiempo de reacción del Arduino, pues tomará decisiones cada poco más de 2 segundos. Sin embargo a futuro es posible utilizar ventanas deslizantes para analizar pequeños intervalos de tiempo y así aumentar la resolución del tiempo, teniendo una respuesta más rápida de la mano biónica.

Bibliografía

[1] Sapsanis, C.; Georgoulas, G.; Tzes, A., EMG based classification of basic hand movements based on time-frequency features,in Control & Automation (MED), 2013 21st Mediterranean Conference on , vol., no., pp.716-722, 25-28 June 2013

[2] Rahman K K, M.; Nasor, M.,Multipurpose Low Cost Bio-Daq System for Real Time Biomedical Applications, 2015 International Conference on Information and Communication Technology Research (ICTRC2015)

[3] Support Vector and Kernel Machines. Nello Cristianni.

[4] Haykin,S.(2009).Neuronal Networks and Learning Machines.