Optimización de hiperparámetros en Deep Learning

En su aplicación a clasificación de EEG

Javier León Palomares

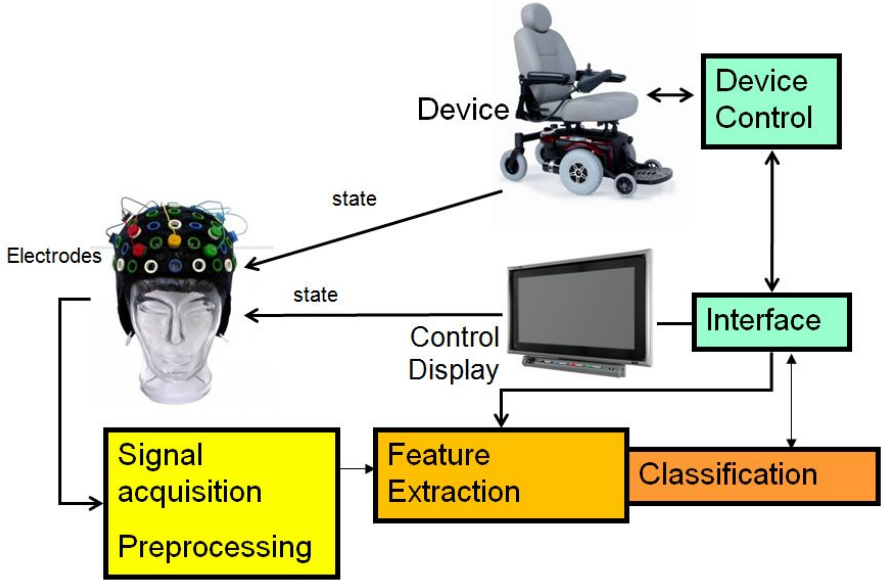
Contenidos

1. Introducción y motivación
2. Objetivos
3. Un poco de trasfondo
4. Metodología
5. Resultados
6. Conclusiones y trabajo futuro


1 — Introducción y motivación

1.1 — Introducción

- BCI: comunicación directa entre el cerebro y un dispositivo externo.
- EEG: un tipo de BCI no invasivo.
- Red neuronal: modelo complejo de aprendizaje automático.
- Algoritmo genético: metaheurística bioinspirada.



1.2 — Motivación

- El campo de las redes neuronales está en auge.
- Conocimientos útiles para un ingeniero informático hoy en día.
- Experiencia con un problema real.
- Buenos resultados abren nuevas vías de investigación.
- Interés personal.

2 — Objetivos

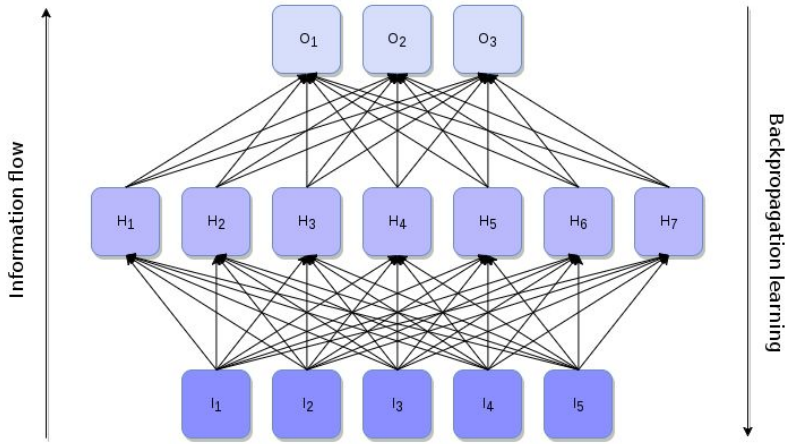
2 — Objetivos

- Adquirir experiencia práctica en el uso de las redes neuronales.
- Aprender acerca de técnicas de optimización aplicables.
- Conseguir los mejores resultados posibles en el caso de estudio.
- Crear una base de código que pueda ser reutilizada y extendida.

3 — Un poco de trasfondo

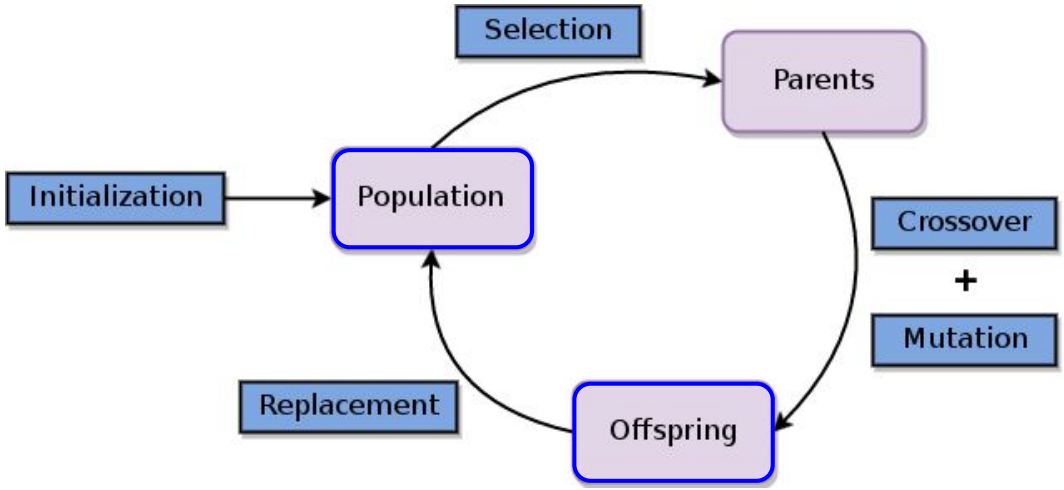
3.1 — Redes neuronales

- Modelos capaces de predecir a partir de unos datos.
- Compuestas por:
  - ➢ Capa de entradas.
  - ➢ Capas ocultas.
  - ➢ Capa de salida.
- A optimizar: número de capas, tamaño de las mismas, tasa de aprendizaje, función de activación...



3.2 — Algoritmos genéticos

- Población de soluciones actualizada en cada generación según *fitness*.
- No garantizan máximos globales, pero suelen ser muy efectivos.



3.3 — Selección de características

- No todas las características tienen por qué ser importantes.
- Reducir su número tiene ventajas:
  - ➢ Entrenamiento más rápido.
  - ➢ Menor *overfitting*.
  - ➢ Mayor interpretabilidad (a veces).
- Es un problema NP-duro, por lo que necesitamos heurísticas.

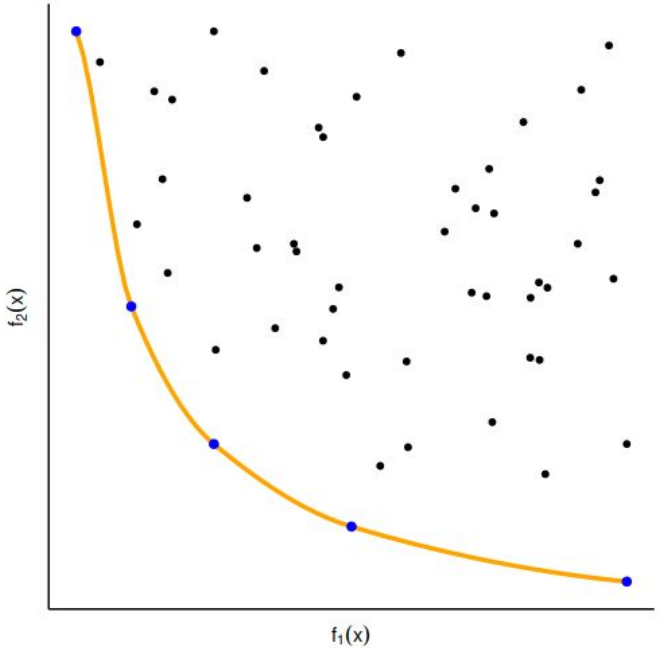
4 — Metodología

4.1 — Conjunto de datos

- Patrones de EEG (178 de entrenamiento y 178 de test).
- Tres sujetos de prueba (104, 107 y 110).
- 3600 características inicialmente.
- Tres clases:
  - ➢ Mover la mano izquierda.
  - ➢ Mover la mano derecha.
  - ➢ Mover los pies.

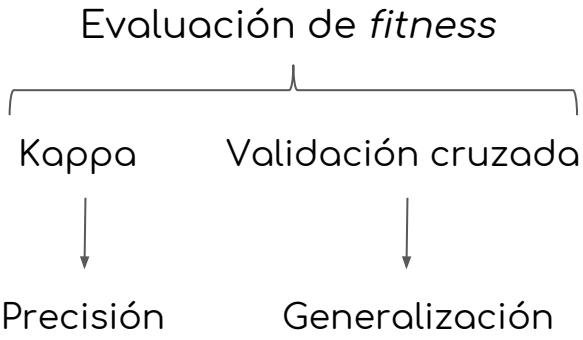
4.2 — Algoritmo genético multiobjetivo I

- Basado en los frentes de Pareto, soluciones que no mejoran a las demás en todas las medidas.
- Generalmente, nos interesa el primero de ellos (marcado en amarillo en la gráfica).
- Las más separadas tienen más prioridad.
- Queda a nuestra elección cuál de las soluciones nos conviene más.



4.2 — Algoritmo genético multiobjetivo II

- Evaluación
- Ordenación
- Reemplazo generacional



Algorithm 1: NSGA-II

Procedure *NSGA-II*
  Input: population size, generations, data, max features
  Output: final population
  population ⟵ Initialize(population size, max features);
  evaluation ⟵ Evaluate(population, data);
  population ⟵ NDSort(population, evaluation);
  for *gen* = 0 **to** *max generations* **do**
    parents ⟵ Selection(population);
    offspring ⟵ CreateOffspring(parents);
    shared population ⟵ population ∪ offspring;
    evaluation ⟵ Evaluate(shared population, data);
    shared population ⟵ NDSort(shared population, evaluation);
    population ⟵ Replace(shared population, population size);
  end
  return population;
end

4.2 — Algoritmo genético multiobjetivo II

- Evaluación
- Ordenación
- Reemplazo generacional

Algorithm 1: NSGA-II

Procedure *NSGA-II*
- **Input**: population size, generations, data, max features
- **Output**: final population
- population ⟵ Initialize(population size, max features);
- evaluation ⟵ Evaluate(population, data);
- population ⟵ NDSort(population, evaluation);
- **for** $gen = 0$ **to** *max generations* **do**
  - parents ⟵ Selection(population);
  - offspring ⟵ CreateOffspring(parents);
  - shared population ⟵ population ∪ offspring;
  - evaluation ⟵ Evaluate(shared population, data);
  - shared population ⟵ NDSort(shared population, evaluation);
  - population ⟵ Replace(shared population, population size);
- **end**
- **return** population;

end

4.2 — Algoritmo genético multiobjetivo II

- Evaluación
- Ordenación
- Reemplazo generacional

Población anterior
+
Descendencia
↓
Ordenar y truncar
↓
Población nueva

Algorithm 1: NSGA-II

Procedure *NSGA-II*
  Input: population size, generations, data, max features
  Output: final population
  population ⟵ Initialize(population size, max features);
  evaluation ⟵ Evaluate(population, data);
  population ⟵ NDSort(population, evaluation);
  for *gen* = 0 **to** *max generations* **do**
    parents ⟵ Selection(population);
    offspring ⟵ CreateOffspring(parents);
    shared population ⟵ population ∪ offspring;
    evaluation ⟵ Evaluate(shared population, data);
    shared population ⟵ NDSort(shared population, evaluation);
    population ⟵ Replace(shared population, population size);
  end
  return population;
end

4.3 — Pasos de optimización

1. Selección de características.
2. Optimización de estructuras de las redes.
3. Optimización de parámetros de aprendizaje.

El proceso es incremental: en cada paso se usan los resultados de los anteriores.

5 — Resultados


5.1 — Configuración experimental

5.1 — Configuración experimental

Software:

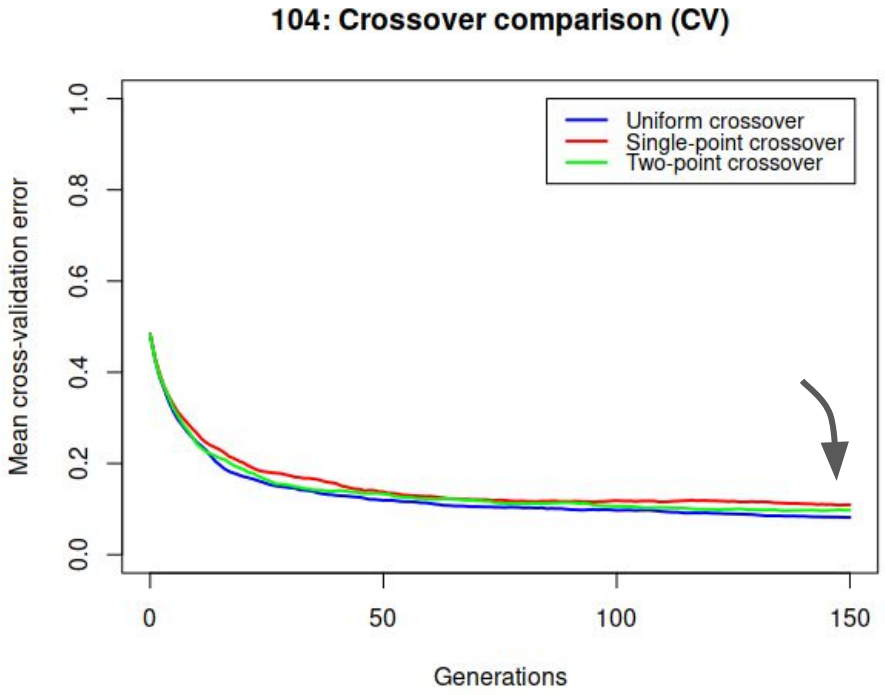
- *Python*, *Keras* (con *TensorFlow* como backend), *Scikit-Learn* y *NumPy* para obtener resultados experimentales.
- *R* para crear gráficas.

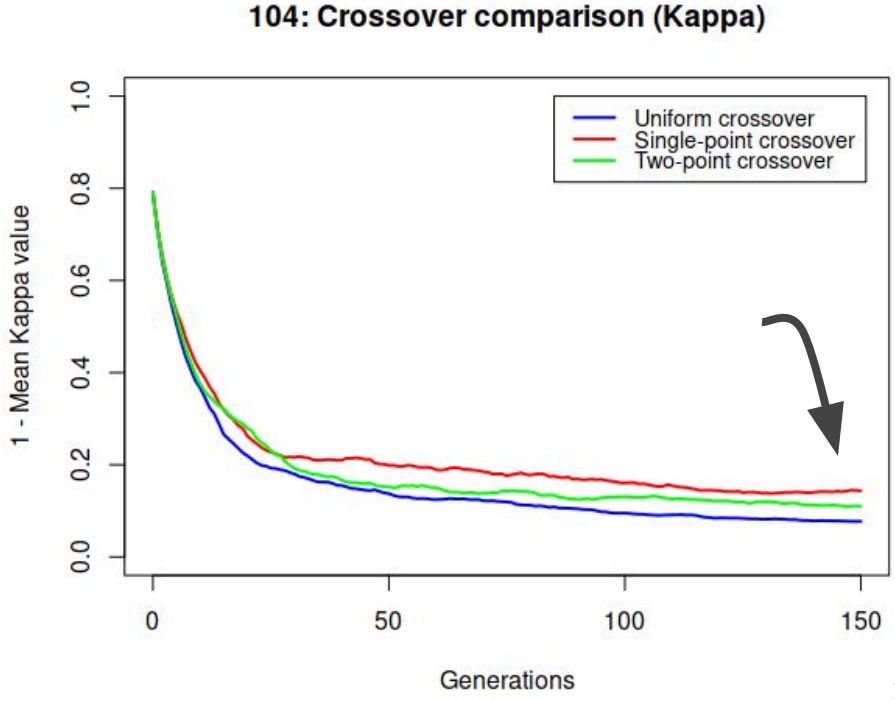
Hardware:

- Primer servidor dedicado:
  - ➢ Dos **Intel® Xeon® E5-2620 v2 @ 2.10GHz, 32GB DDR3.**
  - ➢ **NVIDIA Tesla® K20c, 5GB GDDR5.**
- Segundo servidor dedicado:
  - ➢ **Intel® Xeon® E5-2620 v4 @ 2.10GHz, 32GB DDR4.**
  - ➢ **NVIDIA Tesla® K40m, 12GB GDDR5.**

5.2 — Selección de características

5.2.1 — Selección de características: crossover I





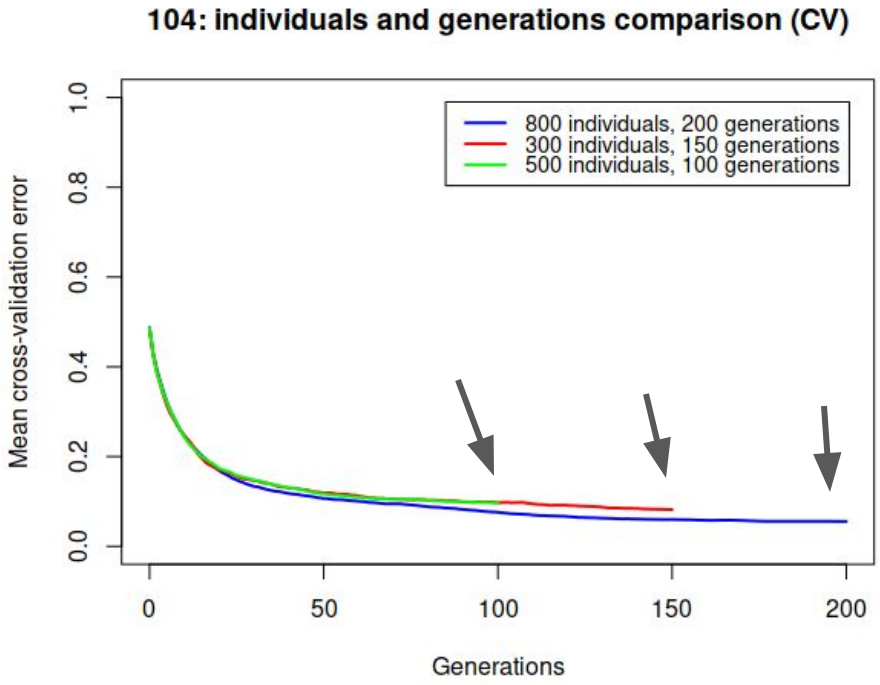
5.2.1 — Selección de características: crossover II

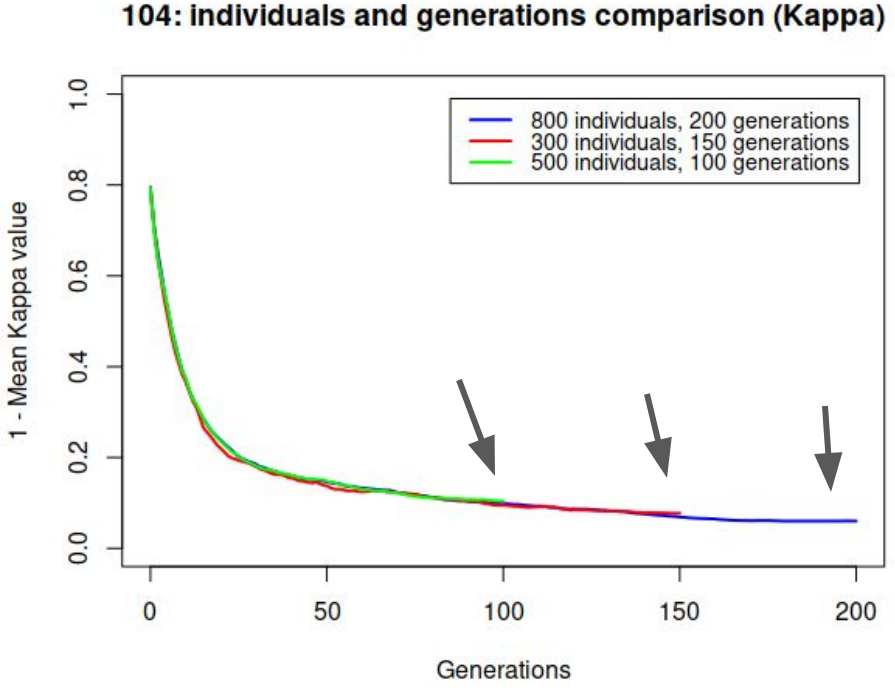
104	Single-point	Two-point
Uniform	$p = 0.000027$	$p = 0.000835$
Single-point		$p = 0.056282$

107	Single-point	Two-point
Uniform	$p = 0.000023$	$p = 0.000104$
Single-point		$p = 0.678133$

110	Single-point	Two-point
Uniform	$p = 0.000454$	$p = 0.006561$
Single-point		$p = 0.299489$

5.2.2 — Selección de características: tamaño I





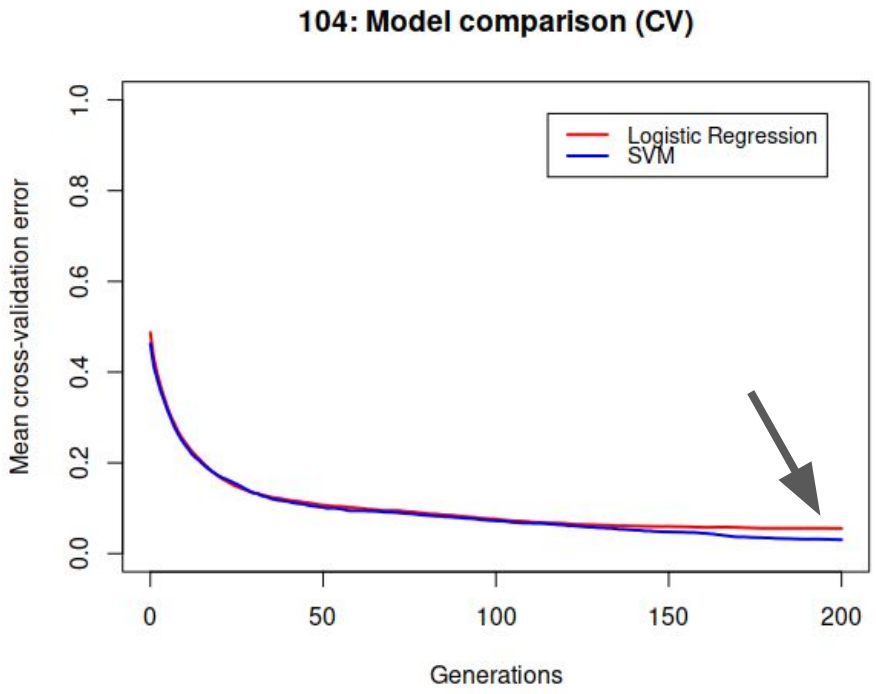
5.2.2 — Selección de características: tamaño II

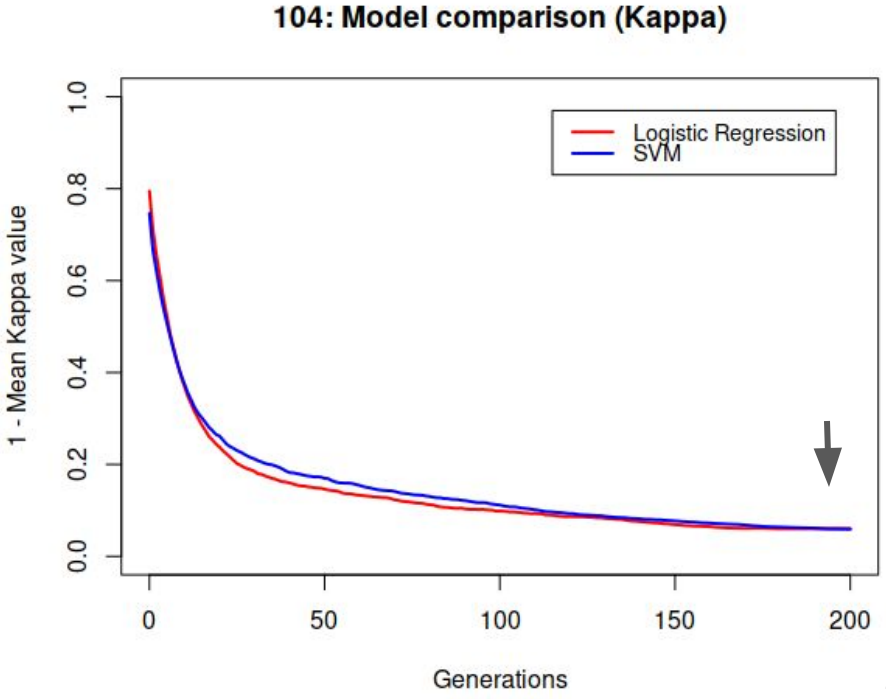
104	300-150	500-100
800-200	$p = 0.000144$	$p = 0.000301$
300-150		$p = 0.884484$

107	300-150	500-100
800-200	$p = 0.000016$	$p = 0.000005$
300-150		$p = 0.002441$

110	300-150	500-100
800-200	$p = 0.003392$	$p = 0.000005$
300-150		$p = 0.034037$

5.2.3 — Selección de características: modelo





5.2.3 — Selección de características: modelo

Logistic Regression against SVM		
104	**107**	**110**
$p = 0.917098$	$p = 0.000025$	$p = 0.000060$

5.2.4 — Selección de características: resultados

Los mejores resultados hasta el momento son:

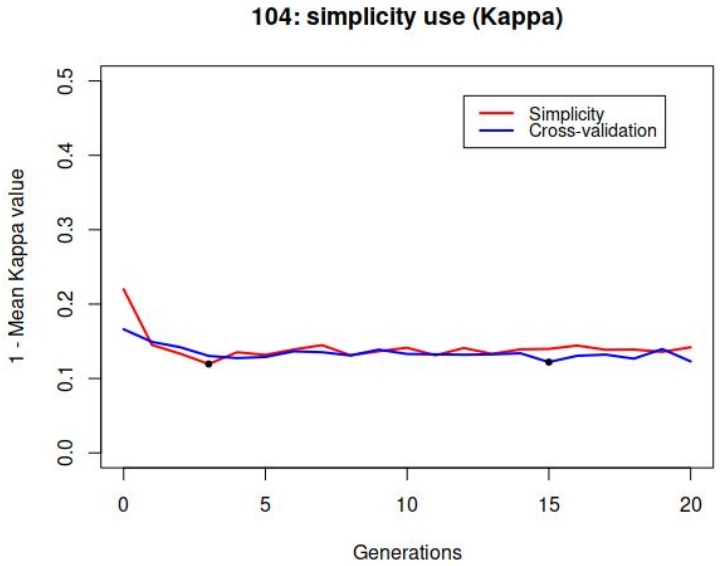
	Feature selection		No feature selection	
Subject	Log Reg	SVM	Log Reg	SVM
104	0.042246	0.033781	0.279954	0.305215
107	0.101059	0.075823	0.327885	0.328009
110	0.117981	0.101107	0.404468	0.404545

(1 - Coeficiente Kappa sobre el conjunto de test)


5.3 — Optimización de estructuras

5.3.1 — Optimización de estructuras: *fitness*

Cambiamos validación cruzada por una medida de simplicidad.



$$T_s = T_{cv} - \left(\frac{k}{k+1}\right) T_{cv} = \left(\frac{1}{k+1}\right) T_{cv}$$

Subject	Cross-validation	Simplicity
104	0.09289	0.08456
107	0.16840	0.19395
110	0.15169	0.17690

5.3.2 — Optimización de estructuras: resultados

Los resultados promedio y pico de la optimización de estructuras son:

	Average	Best
104	0.09037 ± 0.0089	0.07598
107	0.21734 ± 0.0328	0.14319
110	0.20895 ± 0.0147	0.18543

(1 - Coeficiente Kappa sobre el conjunto de test)

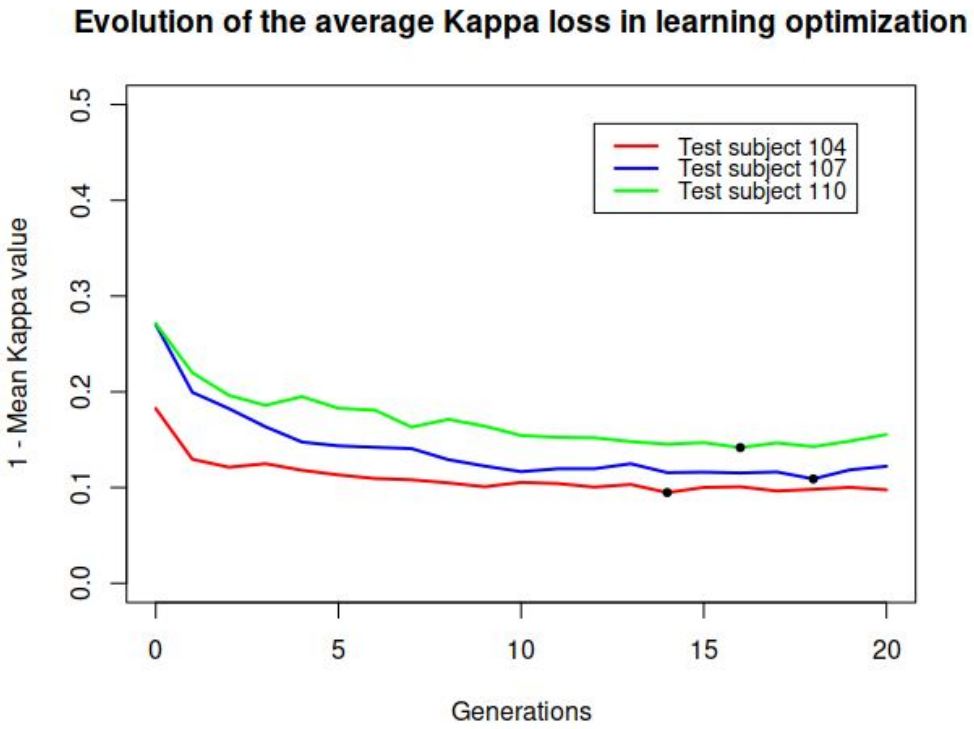
5.4 — Optimización de aprendizaje

5.4.1 — Optimización de aprendizaje: descripción

Buscamos optimizar tres hiperparámetros en concreto:

- Épocas de entrenamiento: en el rango [1,→).
- Tasa de aprendizaje: en el rango (0,1].
- Tasa de *dropout*: en el rango [0,1).

5.4.2 — Optimización de aprendizaje: evolución



5.4.3 — Optimización de aprendizaje: comparativa

	Average	Best
104	0.06756 ± 0.0119	0.05907
107	0.08636 ± 0.0106	0.07582
110	0.11794 ± 0.0069	0.10953

Optimización de redes neuronales

Selección de características

	Feature selection		No feature selection	
Subject	Log Reg	SVM	Log Reg	SVM
104	0.042246	0.033781	0.279954	0.305215
107	0.101059	0.075823	0.327885	0.328009
110	0.117981	0.101107	0.404468	0.404545

5.4.4 — Optimización de aprendizaje: modelos

Aquí se muestran las mejores redes obtenidas para los tres sujetos de prueba:

- **104**: **50** características, **1** capa oculta (**110** neuronas), **283** épocas, **0.027** tasa de aprendizaje, **0%** dropout, función de activación **ELU**.
- **107**: **46** características, **2** capas ocultas (**89** y **102** neuronas), **239** épocas, **0.048** tasa de aprendizaje, **5%** dropout, función de activación **ELU**.
- **110**: **50** características, **1** capa oculta (**34** neuronas), **286** épocas, **0.066** tasa de aprendizaje, **5%** dropout, función de activación **ELU**.

5.5 — Eficiencia

5.5.1 — Eficiencia: selección de características

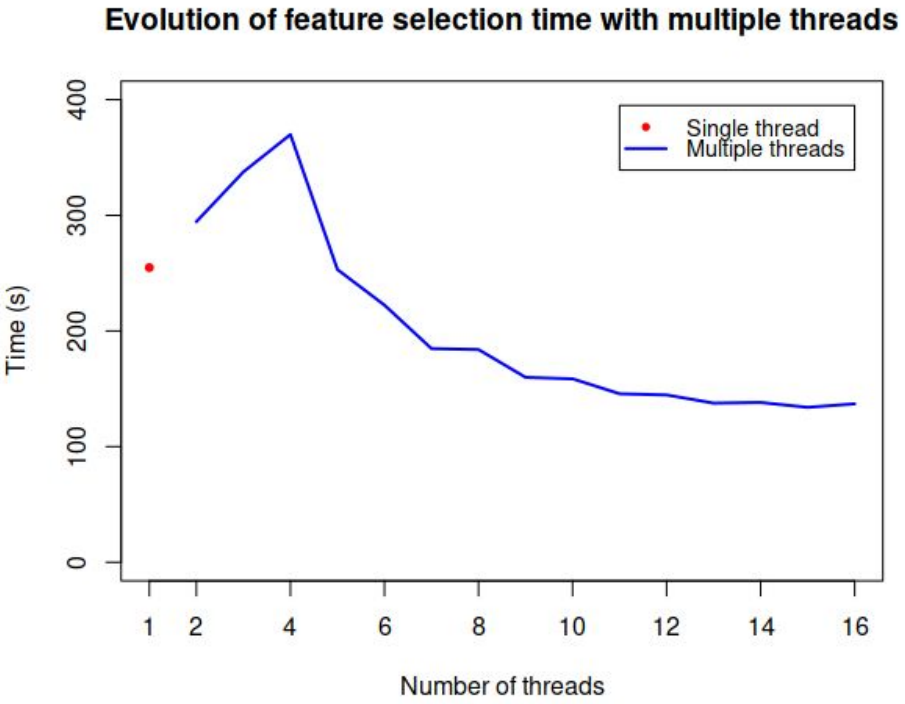
Si comparamos los modelos que evalúan las características:

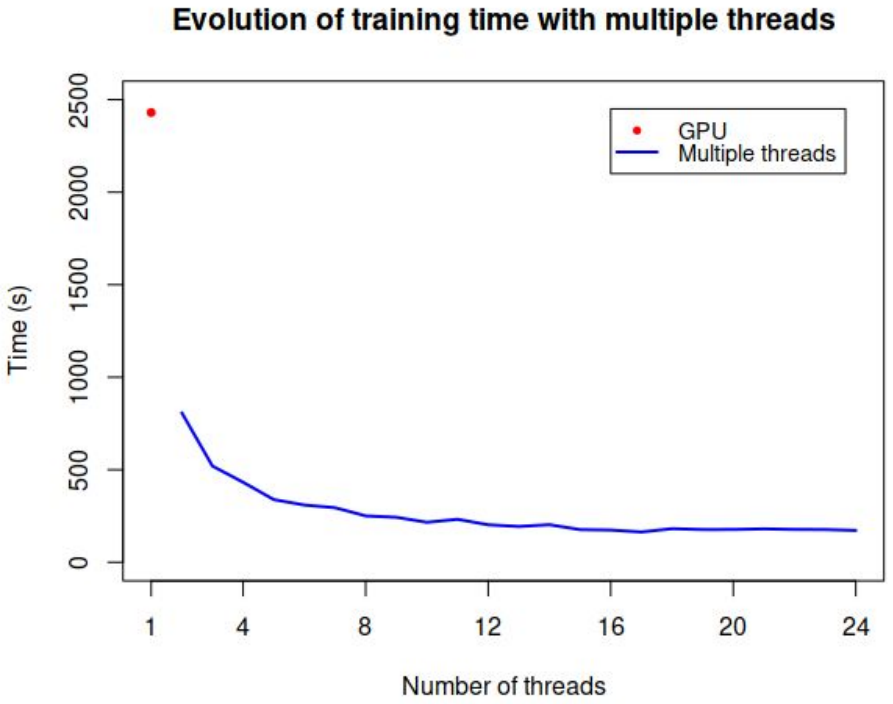
LogReg (s)	SVM (s)
1043.635	2220.472

Y si evaluamos con SVM pero en paralelo con 16 hebras:

Sequential (s)	Parallel (s)
2220.472	922.550

5.5.2 — Eficiencia: paralelización de hebras





6 — Conclusiones y trabajo futuro

6.1 — Conclusiones

- El código[1] cumple su función, además de ser reutilizable y paralelizable.
- Se utilizan tecnologías actuales (como *Python* o *TensorFlow*...).
- Hemos obtenido resultados muy prometedores sobre el conjunto de datos.
- La selección de características es una fase tan importante como la optimización de modelos.

1. Código fuente disponible en: https://github.com/jleon95/UGR_TFG

6.2 — Trabajo futuro

- Explorar el límite de características en la fase de selección.
- Optimizar los SVM para tratar de mejorar los resultados.
- Aplicar técnicas de paralelismo más avanzadas.
- Estudiar una posible transferencia de características.

Preguntas