Memoria proyecto
Aprendizaje Automático y Big Data

Detección de arritmias mediante ECG

Mario Michiels Toquero

Table of Contents

1. Introducción: Explicación del proyecto

1) Objetivo

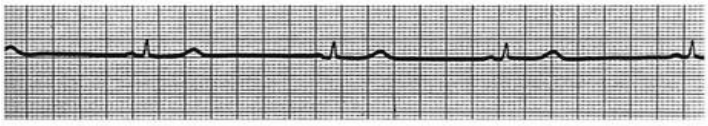
El **objetivo** del proyecto es:

Detectar si un paciente tiene algún tipo de arritmia y de qué clase(clasificación multiclase) mediante la realización de un ECG (electrocardiograma) hecho al paciente.

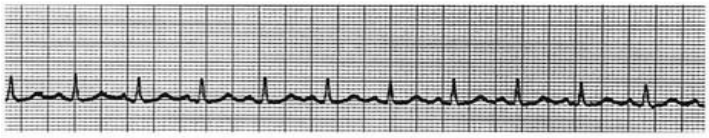
Del mismo modo también se ha realizado una clasificación binaria:

Pacientes con ECG normal vs paciente con arritmias.

2) ¿Qué es una arritmia?



Sinus bradycardia



Sinus tachycardia

Una arritmia es cualquier alteración del ritmo cardiaco usual del individuo.

Por ejemplo como se observa en la foto de arriba, bradycardia significa que el ritmo cardiaco es más lento del habitual, mientras que tachycardia es justamente lo contrario.

Hay muchos tipos de arritmias, y cada tipo tiene una lectura médica diferente y nos da pistas sobre posibles problemas de corazón.

Nota: No todas las arritmias simbolizan tener un problema de corazón, hay algunos tipos que son absolutamente normales como la bradycardia y la tachycardia, siempre que estén dentro de ciertos intervalos normales. Es decir un paciente puede tener alguna arritmia y estar totalmente sano.

Por ese motivo a lo largo de la memoria, se referirá a los que tienen un ECG normal(sin arritmias encontradas) como ECG normal, no como sano, ya que efectivamente puede haber alguien con arritmias que esté sano.

3) Utilidad de un ECG

Illustration 1: Medición de ECG

VS

Illustration 2: Medición del pulso

La ventaja de un ECG frente a otros métodos de medición para comprobar el estado del corazón es que un ECG nos da mucha más información que los métodos comunes como simplemente medir el pulso o un estetoscopio.

Esa información extra es:

- Medición continua durante mucho tiempo (días incluso si es un Holter(ECG portatil)).
- Se mide el movimiento de los músculos del corazón (los electrodos conectados al paciente registran esos movimientos en mV), lo cual permite saber cuando entra la sangre de él, cuando sale, cuanto dura cada movimiento, etc.

  Sin esa información sería imposible detectar ciertos tipos de arritmias y/o otras alteraciones del corazón.

2. Dataset

1) Instancias, clases y features

Se especifica que el dataset tiene **452 instancias** (filas), **16 clases** y **279 features.**

Cada instancia corresponde a un paciente y las features corresponden a los resultados del ECG(además de otros valores como la edad, altura, etc).

Finalmente la clase resultante para cada paciente se ha obtenido mediante médicos especialistas que han analizado los ECG del paciente y han dado su veredicto.

Hay 3 clases que no tienen instancias, por tanto realmente son 13 las clases que tiene el dataset. Se reparten de la siguiente manera:

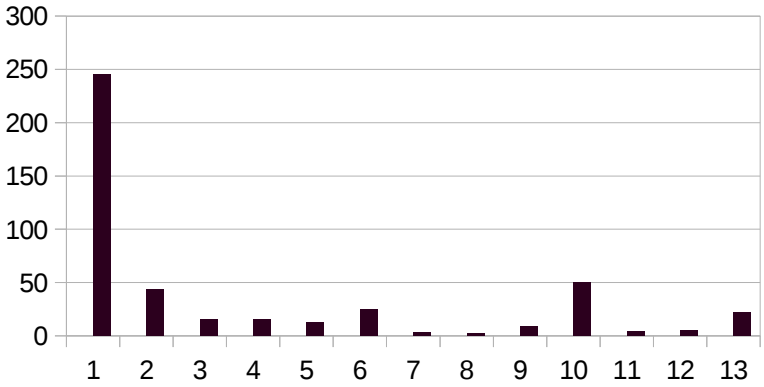


Illustration 3: Dataset: Clases (Eje X: clase) (Eje Y: número de instancias)

La clase 1 corresponde a un estado normal del paciente, es decir no se han encontrado arritmias

(**Nota**: Lo más normal (aunque no se especifica) es que los pacientes de la clase 1 hayan tenido alguna arritmia en el tiempo que se haya hecho el ECG pero serían tan insignificativas que no se han tenido en cuenta).

Todas las demás clases simbolizan un tipo determinado de arritmia encontrado en el ECG del paciente.

También podemos visualizar los datos de manera que enfrentemos los pacientes con ECG normal con los que tienen algún tipo de arritmias.

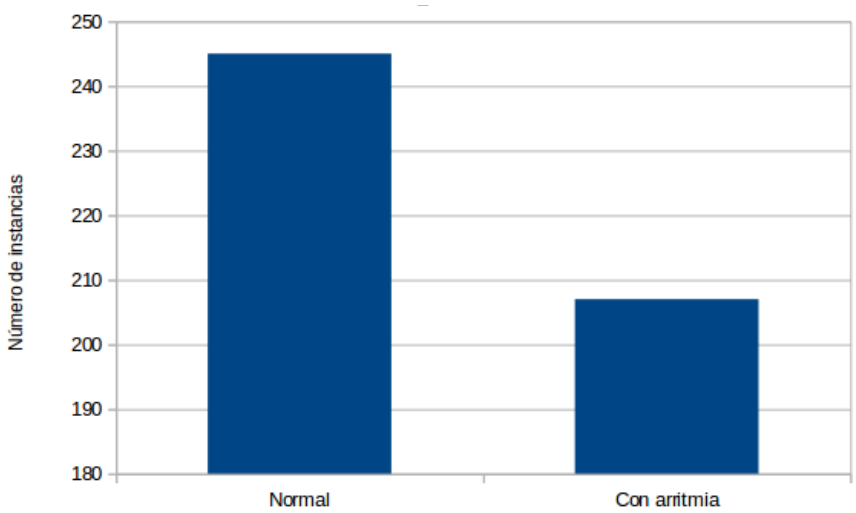


Illustration 4: Dataset: Clase normal vs resto de clases

3. Preprocesamiento

1) Problemas con los datos (y sus soluciones)

(1) Eliminar las clases sin instancias

Problema: como se ha comentado antes 3 clases no tenían ninguna instancia.

Solución: a la hora de establecer el número de clases para ajustar los parametros de los algoritmos, se hizo obviando esas clases.

(2) Borrar features invariantes

Problema: en el dataset había algunas features que no variaban entre sus valores en cada fila. Por tanto al ser iguales en todas las filas, no aportan ninguna información que diferencien a un paciente de otro.

Ejemplo de features invariantes:

0 10

0 10

0 10

Solución: Se hace el mínimo y el máximo de la columna (1 columna = 1 feature) y si son iguales es que es invariante y la borramos esa columna.

(3) Atributos desconocidos

Problema: en determinados pacientes había algunas features que eran desconocidas(simbolizadas con “?” en el dataset) bien sea porque se han perdido dichos datos, porque no se le hiciera correctamente el ECG, etc.

Solución: se sustituyen los “?” por el valor de la media de esa columna, así no desechamos toda esa instancia y aunque no sea el valor real que tendría, es algo razonable que está dentro de lo normal.

Se ha optado por esta solución porque los valores que desconocidos eran valores aislados en ciertos pacientes. Si hubieran sido generalizados: en casi todas las instancias fuera desconocida X feature, no se hubiera podido hacer esta solución porque la media estaría muy condicionada por los pocos valores que fueran conocidos, y por tanto no tendría por qué ser un valor dentro de lo normal.

2) Preparación de los datos

(1) Normalización de los datos

Siempre es recomendable normalizar los datos para que los valores estén dentro de un rango manejable y sean equiparables.

En este caso más que recomendable ha sido **imprescindible** ya que hay datos tan diferentes como la edad y la lectura en mV de un electrodo del ECG.

(2) PCA (Principal component analysis): Visualización de los datos

Al existir tantas features es imposible visualizar los datos en un gráfico en 2D (2 ejes), no se podría visualizar un gráfico con 279 ejes. Necesitamos tener como mucho 3 ejes (3D).

Por ello se ha optado por realizar el algoritmo **PCA** que consiste en combinar matemáticamente las características que más varían de las features y comprimir todas esas variaciones en X componentes.

En este caso se ha optado por obtener 2 componentes, así por ejemplo el 1er componente (principal component 1) es el componente con más variación de todo el dataset(obtenido de combinar las features con más variación).

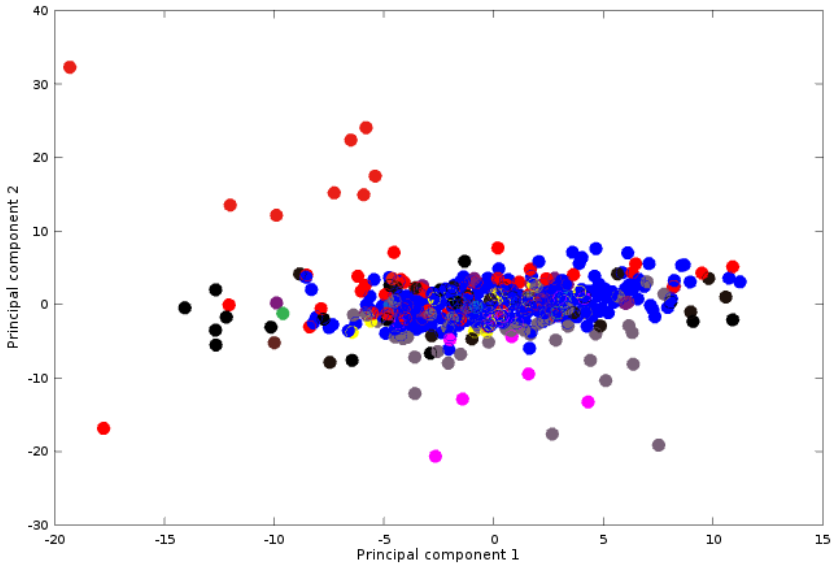


Illustration 5: PCA:

Todas las clases. Clase 1 (azul): Normal

Resto de clases (Cada una de un color diferente)

Nótese que al tener todas las features condensadas en tan solo 2 ejes se pierde información al comprimir tanto la información. Por eso en el gráfico puede a veces parecer que la clase 1 tendría las mismas características que otras clases.

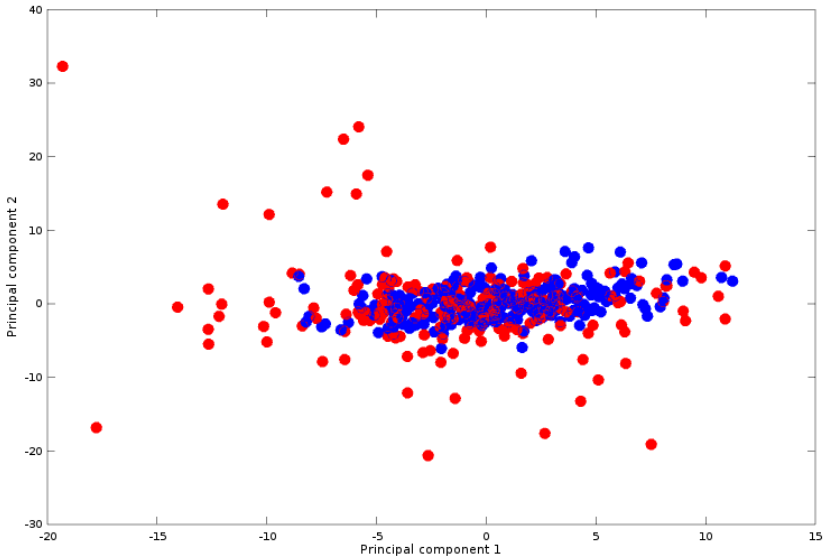


Illustration 6: PCA binario (solo 2 clases):

Clase 1(azul): Normal

Clase 2(roja): Con arritmias

(3) Feature selection

Se ha optado por eleminar ciertas features debido a que al haber tantas, en la vida real, al hacer un ECG a un paciente, no se podrían aplicar los algoritmos de este proyecto, ya que faltarían datos.

```
193,371,174,121,-16,13,64,-2,?,63,0,52,44,0,0,32,0,0,0,0,0
174,401,149,39,25,37,-17,31,?,53,0,48,0,0,0,24,0,0,0,0,0,0
,163,386,185,102,96,34,70,66,23,75,0,40,80,0,0,24,0,0,0,0,
,202,380,179,143,28,11,-5,20,?,71,0,72,20,0,0,48,0,0,0,0,0
181,360,177,103,-16,13,61,3,?,?,0,48,40,0,0,28,0,0,0,0,0,0
,167,321,174,91,107,66,52,88,?,84,0,36,48,0,0,20,0,0,0,0,0
129,377,133,77,77,49,75,65,?,70,0,44,0,0,0,24,0,0,0,0,0,0,
0,376,157,70,67,7,8,51,?,67,0,44,36,0,0,24,0,0,0,0,0,0,0,5
118,354,160,63,61,69,78,66,84,64,0,40,0,0,0,20,0,0,0,0,0,0
130,383,156,73,85,34,70,71,?,63,0,44,40,0,0,28,0,0,0,0,0,0
,135,401,156,83,72,71,68,72,?,70,20,36,48,0,0,36,0,0,0,0,0
143.373.150.65.12.37.49.26.?.72.0.40.28.0.0.20.0.0.0.0.0.0
```

Illustration 7: Feature selecion: Demasiadas features

Los datos del ECG se dividen por canales (1 canal = 1 cable).

En la vida real los ECG se miden normalmente con 12 o 13 canales, como en este dataset (12 canales). Sin embargo al realizar los holter(ECG portatil durante al menos 24 horas), suele ser bastantes menos canales, en mi caso la ultima vez que me hicieron uno sólo fueron 3 canales.

En nuestro TFG (monitorización de señales biomédicas) utilizamos algo intermedio: 8 canales, dado que su uso podra ser como holter(ECG portatil) o como simplemente un ecg comun de una clínica para medir durante solo unos segundos.

Por ese motivo la seleccion de features se ha realizado con 8 canales. Descartamos por tanto los canales DI, DII, DIII (por ser los cables que se ponen en las extremidades y no es habitual ponerlos) y el V6 por estar muy cerca del V5 y no perder asi casi informacion.

Illustration 8: Cables ECG del TFG, hay más de 8 canales pero el microcomputador utilizado solo tiene 8 entradas

De los canales con los que nos quedamos, se conservan todos sus atributos (ejemplo: media del tiempo que tarda la sangre en entrar al corazón, cuando sale, etc), ya que esperamos poder obtener todas esas métricas en el TFG.

(4) Dividisión del dataset en subconjuntos

Se ha optado por dividir el dataset en 3 subconjuntos:

- Training (60%): datos de entrada para entrenar el algoritmo
- Validation (20%): se prueba el algoritmo ya entrenado con Training sobre este subconjunto.

  Es una manera de ver cómo funcionaría el algoritmo con nuevos datos (distintos a los de entrenamiento).

Se hacen varios ciclos de Training+Validation, en cada ciclo se cambian los parámetros de ajuste de cada algoritmo para ir viendo cómo funciona el algoritmo (entrenado con Training) sobre nuevos ejemplos (subconjunto de Validation).

- Test (20%): prueba “real”, ya no se ajustan parametros. Solo se hace una vez, una vez ya hemos terminado de ajustar el modelo con Training+Validation. Se utiliza para mostrar los resultados del algoritmo, ya que son nuevos ejemplos, como los que se encontrarían en la vida real, es decir que no han influido para nada en el entrenamiento del algoritmo.

La división ha sido de acuerdo al porcentaje recomendado por Andrew Ng en su curso de Machine Learning.Además al tener relativamente pocas instancias (452) no sería recomendable utilizar menos del 60% de los datos para entrenar.

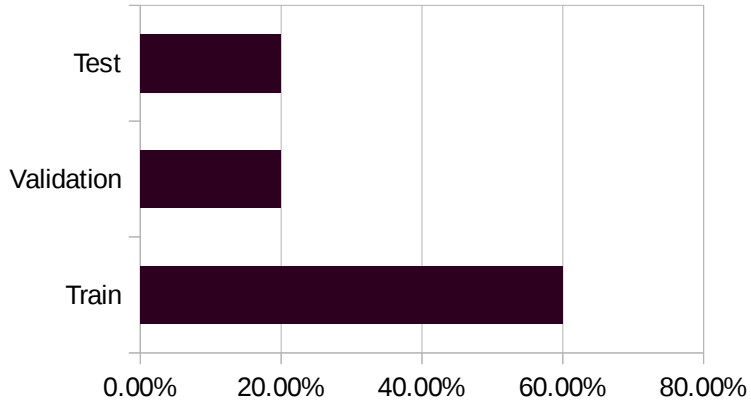


Illustration 9: División del dataset en subconjuntos

Existen varios métodos para dividir el dataset en subconjuntos:

- **Randomized**: aleatoriamente (según esté ordenado el dataset por ejemplo, que no suelen estar ordenados por clases) se ponen una instancias en un subconjunto u otro.
- **Stratified**: se reparten las instancias en un subconjunto u otro de manera equitativa.

  Si por ejemplo hay 10 instancias de la clase 1, y tuvieramos solo 2 subconjuntos (con el 50% de instancias cada uno), se pondrían 5 de esas instancias en un subconjunto, y las otras 5 en el otro. Si fuera randomized podrían caer las 10 instancias en el mismo subconjunto.

En este proyecto se ha optado por el método **stratified**, ya que el dataset tiene el problema de clases con muy pocas muestras y/o desbalanceadas(se explica en la siguiente sección).

En este proyecto sería un gran error hacerlo randomized, por ejemplo que en el subconjunto de Test no se pudiera ni siquiera comprobar si el algoritmo funciona con X clase por no haber ninguna instancia de esa clase en el subconjunto de Test, .

O lo que podría ser incluso peor, que X clase no tuviera ninguna instancia en Training y el algoritmo no fuera capaz de reconocer esa clase cuando llegue una instancia de esa clase.

3) Problemas del dataset para machine learning (y sus soluciones)

(1) Clases con muy pocas instancias

Problema: Hay clases con muy pocas instancias, incluso en alguna ocasión hay tan pocas que no se podría poner ni siquiera una en cada subconjunto del dataset (como el caso de la clase 8).

Clases con muy pocas instancias:

Class code	Class	Number of instances:
07	Ventricular Premature Contraction (PVC)	3
08	Supraventricular Premature Contraction	2
09	Left bundle branch block	9
12	Left ventricule hypertrophy	4
13	Atrial Fibrillation or Flutter	5

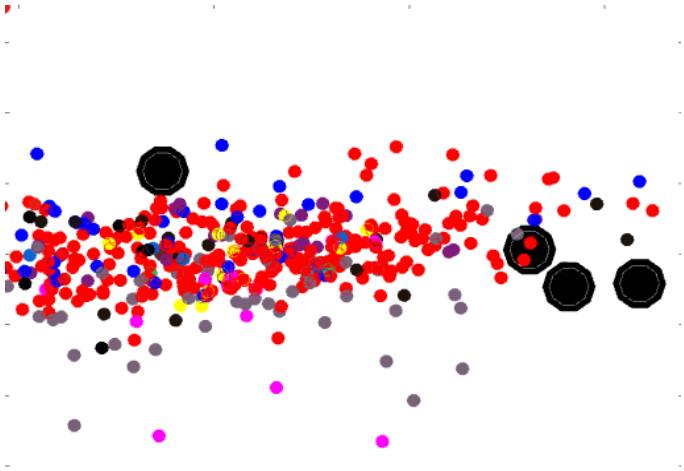


Illustration 10: Clase con muy pocas instancias: Clase 12 en negro más grande

Una posible solución podría haber sido directamente borrar todas esas clases y no tenerlas para nada en cuenta en el proyecto, sin embargo ya que el dataset las tenía quería utilizarlas para que el algoritmo fuese más completo.

Solución: Sobremuestreo (**oversampling**) de esas clases:

Dicha técnica utilizada consiste en introducir en el dataset instancias artificialmente creadas que sean exactamente iguales a las ya existentes.

Así conseguimos tener más instancias de esa clase, las cuales son coherentes (ya que sus features son iguales que las reales ya existentes).

Al tener más instancias de esas clases, ya tenemos suficientes instancias para que en cada subconjunto haya al menos unas cuentas, y además el algoritmo las tendrá más en cuenta, ya que el error de no detectar bien X clase, será mayor si hay más instancias de esa clase, y el algoritmo se ajustará para bajar ese error y así poder detectarlas mejor.

Nota: en todos los casos donde se ha realizado oversampling se ha tomado especial cuidado en que, a la hora de contabilizar los resultados del algoritmo, no se cuenten por duplicado aquellas instancias de las clases en las que se ha realizado oversampling. Para ello la solución ha sido hacer oversampling de esas instancias solo en Training, de tal manera que en Validation y en Test no hay nada igual que en Training.

(2) Clases desbalanceadas

Problema: hay muchas instancias de alguna clase (en este caso de la clase 1: normal), mientras que otras clases tienen muy pocas instancias en comparación con otras.

Nótese que es un problema distinto al de la sección anterior en el sentido de que son clases con muy pocas instancias pero en comparación con otras clases, es decir podría haber una clase que no tuviera el problema de la sección anterior pero sí tuviera este problema, por ejemplo podría haber una clase con 20.000 instancias pero que el resto de clases tuvieran +500.000 instancias.

Así mismo, podría haber clases que tuvieran ambos problemas, como es el caso de la clase 4 ya mostrada en la sección anterior.

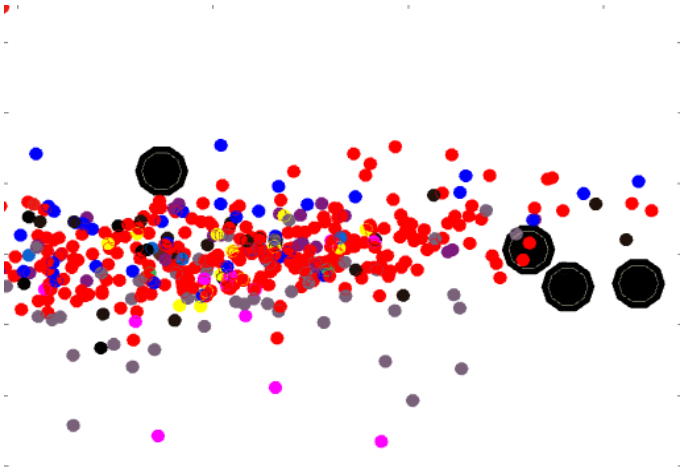


Illustration 11: Clases desbalanceadas:

Clase 4 en negro grande VS Clase 1 (Normal) en rojo

Al igual que en el problema anterior, se podría optar por borrar todas esas clases directamente, pero ya que el dataset las tenía quería utilizarlas para que el algoritmo fuese más completo.

Solución: Sobremuestreo (**oversampling**) de esas clases. Aunque es un problema diferente al anterior, se trata exactamente con la misma solución.

(3) Clases muy similares entre ellas

Problema: hay instancias que perteneciendo a una clase concreta tienen unas features muy similares con instancias pertenecientes a otra clase. Lo cual hace prácticamente imposible que el algoritmo pueda distinguir correctamente a qué clase pertenecen esas instancias.

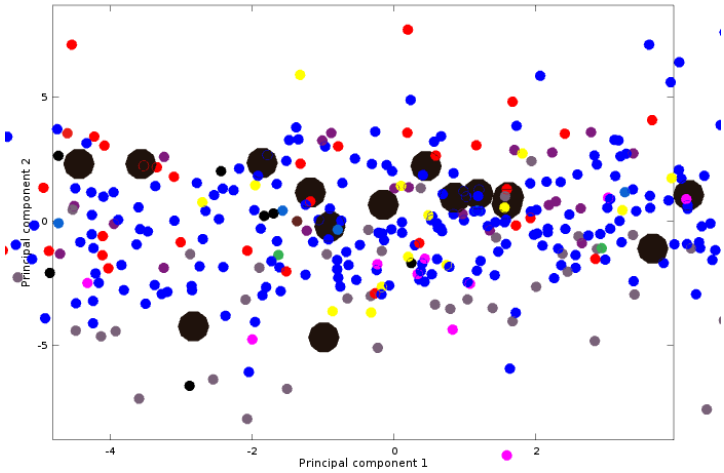


Illustration 12: Clases muy similares entre ellas: Clase 7 (negro grande). Clase 1: normal (azul)

Una posible solución sería directamente borrar todas esas clases, pero ya que el dataset las tenía quería utilizarlas para que el algoritmo fuese más completo.

Solución: sobreajustar (overfitting) dichas clases:

Esta es prácticamente la única solución posible si se deciden conservar esas instancias. Hay que tener en cuenta que al sobreajustar esas clases, nuestro algoritmo, una vez entrenado, solo sería capaz de reconocer clases muy similares a las que ha entrenado, esa es la idea, ya que hacer una diferenciación sin sobreajustar tanto es imposible al ser tan similares las features.

De esta manera al menos podremos detectar las nuevas instancias de estas clases que nos lleguen y sean muy similares a las entrenadas.

Dicha técnica se consigue poniendo en el subconjunto de validación las mismas instancias que en training para que así los valores del porcentaje de aciertos y otras métricas sean altas en esas clases e influya en el ajuste de los parametros del algoritmo.

Es importante hacer este método solo con las instancias de clases que tengan este problema, así solo se sobreajusta esa clase, porque si lo hicieramos con todas, obtendríamos un algoritmo con alta varianza(sobreajustado) que no sería capaz de detectar bien las nuevas instancias que le llegasen.

4. Procesamiento

1) Procedimientos comunes en todos los algoritmos

(1) Comprobación de que el algoritmo es correcto

Este es un primer paso que siempre es aconsejable realizar.

Consiste en entrenar el algoritmo con el subconjunto Training y probarlo sobre el mismo subconjunto Training, si nuestro objetivo es que el algoritmo tenga expresividad suficiente como para poder acertarlo todo perfectamente, deberíamos obtener un porcentaje de aciertos del 100%.

En caso de obtener un porcentaje algo inferior pero bastante más del 50%, se podría decir que el algoritmo es correcto pero nos falta expresividad en el algoritmo(o en las features).

En caso de rondar el 50% o ser inferior, claramente el algoritmo no está funcionando correctamente, y no conviene seguir trabajando en el proyecto hasta no solucionarlo, ya que si este primer paso no es satisfactorio, el resto de pasos tampoco lo van a ser.

En el caso de este proyecto, se obtuvo un 100% de porcentaje de aciertos en este paso ya que todos los algoritmos tienen expresividad suficiente para ello.

(2) Cross-Validation (k-folds)

Se llama **Cross-Validation** al entrenamiento de un algoritmo basándose en la realización de ciclos, de tal manera que en cada ciclo:

se cambian los parametros de ajuste y se deja un subconjunto que no participe en el entrenamiento(validation) sobre el que comprobar como varía el resultado con la variación del parametro de ajuste que hemos hecho.

Estas pruebas de cómo funciona el algoritmo con las instancias de Validation es útil para comprobar si el algoritmo es capaz de detectar correctamente instancias de X clase cuyas features no sean exactamente iguales que las de las instancias de esa clase de Training.

Si no es capaz de detectar las clases de esas nuevas instancias, pero si puede detectar las instancias de entrenamiento, se dice que el algorimo presenta un sobreajuste (overfit), es decir alta **varianza** (variance).

Por el contrario si se detectan estas nuevas instancias pero no se detectan las instancias de entrenamiento se dice que estamos en situación de **sesgo** (bias).

Al realizar Cross-Validation (y habiendo realizado previamente lo explicado en el apartado anterior), conseguimos no entrar en situación de sesgo ni de varianza, sino en una situación intermedia, equilibrada, mediante el ajuste de los parámetros en cada vuelta.

Así al terminar todos los ciclos podemos saber con qué valor de los parámetros de ajuste ha funcionado mejor el algoritmo.

Hay varias técnicas para realizar Cross-Validation:

- *Leave-one-out cross-validation (LOOCV)*: lo implementamos en las prácticas de manera que los subconjuntos de Training y Validation ya estaban definidos antes de empezar a hacer Cross-Validation, de tal manera que las instancias de los subconjuntos nunca variaban en cada vuelta.
- K-folds: por cada vuelta en la que se cambie el parámetro de ajuste del algoritmo, se realizan k vueltas en cada una de las cuales las instancias de Training y Validation son distintas. Dicho método permite entrenar y probar sobre más valores si el nuevo parámetro de ajuste del algoritmo funciona. Al final de las k vueltas se hace la media del resultado de cada vuelta, y así obtenemos el resultado que nos permite saber cómo de bueno ha sido ese parámetro de ajuste para el algortimo.

En este proyecto se ha utilizado K-folds por las ventajas previamente explicadas.

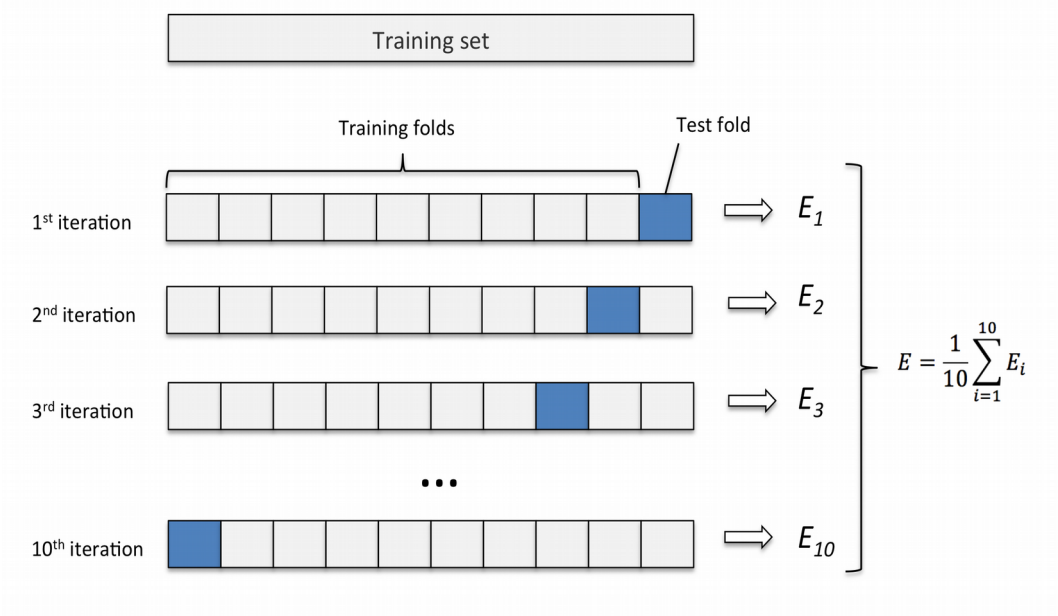


Illustration 13: Cross validation: K-folds method

Nota: En la imagen superior se le llama “Test fold” a lo que nosotros llamaríamos “Validation fold”.

Una buena comparativa visual de si hay sesgo o varianza es hacer el plot de cómo varía el AUC (o cualquier otra métrica que evalue lo bueno que es un algoritmo) en función de los parámetros de ajuste, y que los valores a dibujar sean tanto los de entrenamiento como los de validación, así se observa si el algoritmo solo es bueno en training, o en validación, si está equilibrado, etc.

Nota: Estas comparativas visuales se pueden ver en este proyecto en los primeros apartados del ajuste de los parámetros de cada algoritmo de los que hay continuación en las siguientes secciones.

Todos los parámetros óptimos son elegidos en base a cual es el máximo AUC para el subconjunto de Validation.

2) Regresión logística multiclase

(1) Ajuste de lambda (clasificación multiclase)

Lambda es un parámetro que se utiliza para determinar la regularización del algoritmo, es decir varía la expresividad de un algoritmo para aumentar o disminuir la varianza(y consecuentemente el sesgo).

Lambda varía de la siguiente:

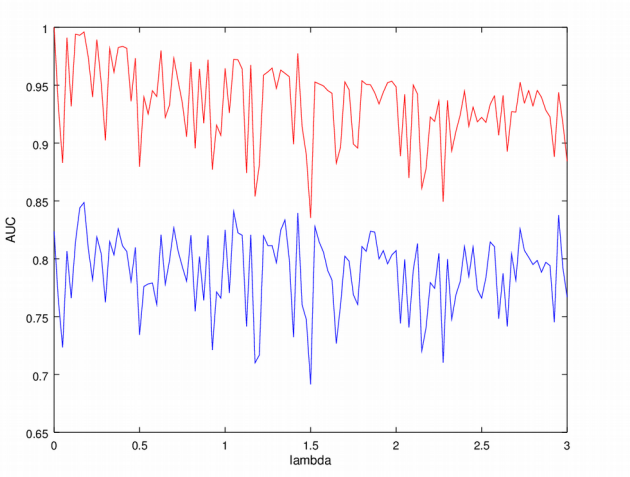
< lambda → > **varianza** (sobreajuste)

> lambda → > **sesgo**

Es decir con lambda = 0, el algoritmo (con las iteraciones suficientes) debería ser capaz de acertar el 100% de las instancias de Training.

En este proyecto se ha obtenido el siguiente lambda para la regresión logística multiclase:

Se realizó primero una ejecución del algoritmo hasta llegar a lambda = 3



Al ver que principalmente el AUC podía aumenta cuando lambda < 1, se decició hacer una segunda ejecución con saltos más pequeños de iteración, solo hasta lambda = 1:.

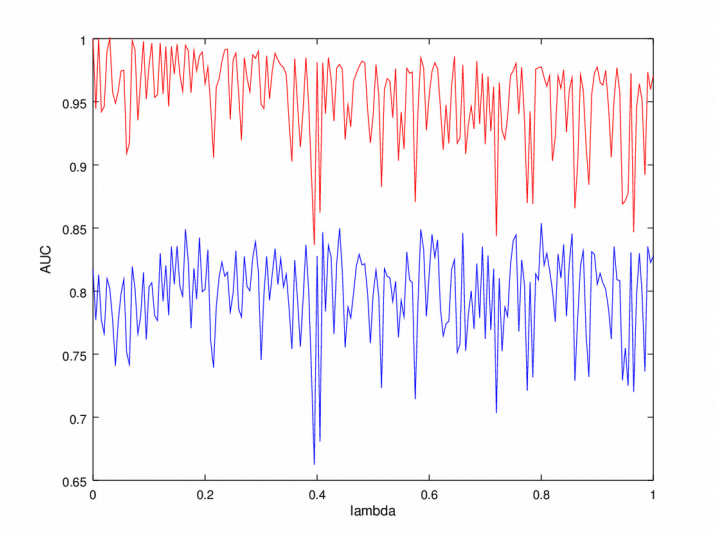


Ilustración 14: Lambda = 0.80000

AUC_validation = 0.85380

Rojo: Training

Azul: Validation

Lo cual significa que no estamos en situación de sesgo (pues se ve que con lambda 0 el AUC de Training es practicamente perfecto y en cuanto el AUC empieza a emperorar en Validation, tomamos ese valor de lambda de justo antes de empezar a empeorar el AUC, para que no se tenga más sesgo).

Mientras que tampoco estamos en situación de alta varianza (pues se ve que el AUC es bueno(bastante más del 50%) en validación, y que a partir del valor de lambda obtenido empieza a disminuir el AUC, lo cual indica que a partir del valor de lambda que ha resultado mejor, se podría tender hacia el sesgo y podría llegar incluso a aumentar tanto lambda que no se detectaran ni las instancias de Tranining ni de Validation).

(2) Ajuste de lambda (clasificación binaria)

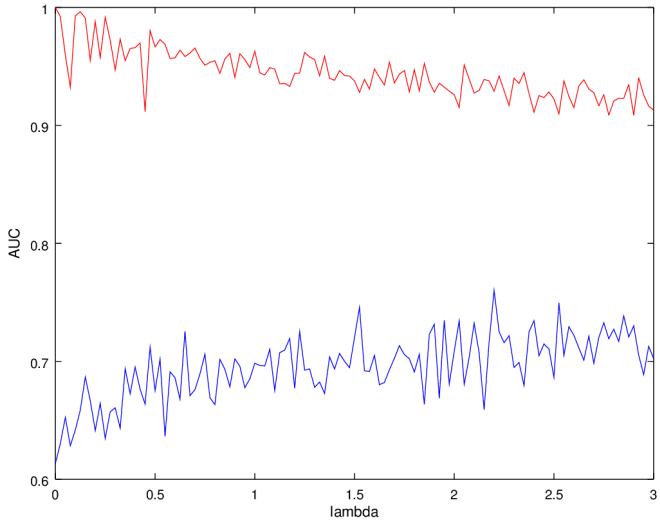


Ilustración 15: Lambda = 2.2000

AUC_validation = 0.76018

Rojo: Training

Azul: Validation

(3) Obtención de resultados multiclase (One vs All)

Si solo tuvieramos dos clases, la decisión de qué clase es cada uno sería facil de realizar en el algoritmo ya que si no es de una clase, es de la otra, no hay más opciones.

Sin embargo al tener que decidir entre 13 clases hay que realizar alguna técnica para poder hacer la decisión de a qué clase corresponse cada instancia.

La técnica One vs All consiste en iterar sobre todas las clases que haya y en cada iteración enfrentar a la clase X con todas las demás clases. Por ejemplo en este caso se pone a 1 la etiqueta de la clase X y el resto de clases a 0.

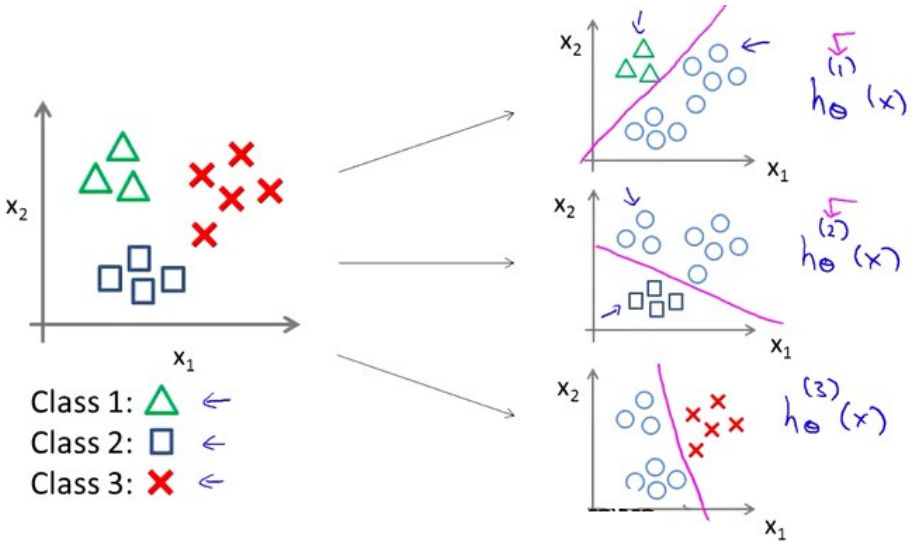


Illustration 16: Esquema One vs All

3) Redes neuronales

(1) Ajuste de lambda (Clasificación multiclase)

Lambda en las redes neuronales actúa de la misma manera que en regresión logística, por tanto las explicaciones teóricas y la interpretación del resultado es la misma que en el apartado 4.2.1.

En este proyecto se ha obtenido el siguiente lambda para las redes neuronales:

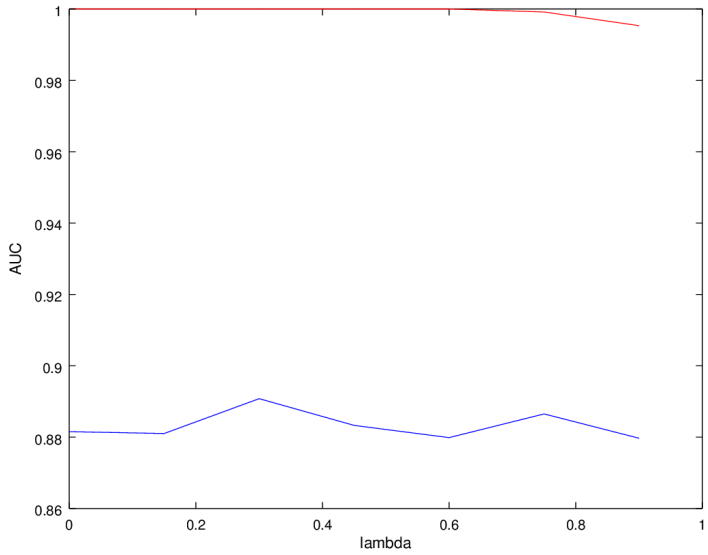


Illustration 17: Lambda = 0.31000

AUC_validation = 0.89835

Rojo: Training

Azul: Validation

(2) Ajuste de lambda (Clasificación binaria)

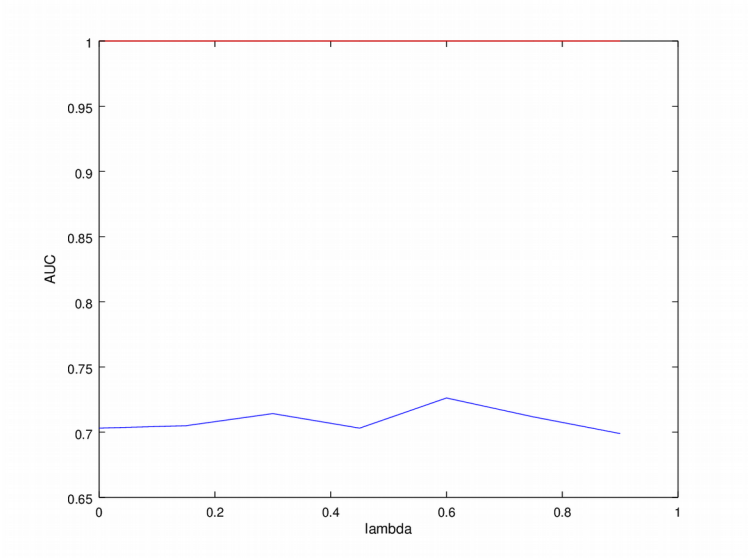


Illustration 18: Lambda = 0.60000

AUC_validation = 0.73911

Rojo: Training

Azul: Validation

(3) Ajuste de nodos por cada capa oculta (Clasificación multiclase)

Cuantos más nodos por cada capa oculta, más expresividad tendrá el algoritmo. Si el algoritmo está bien, así como el dataset bien preprocesado, debería llegar un número de nodos por tanto que si probamos el algoritmo en Training, nos debe dar 100% de AUC.

En este proyecto se ha obtenido el siguiente número de nodos por capa oculta para las redes neuronales:

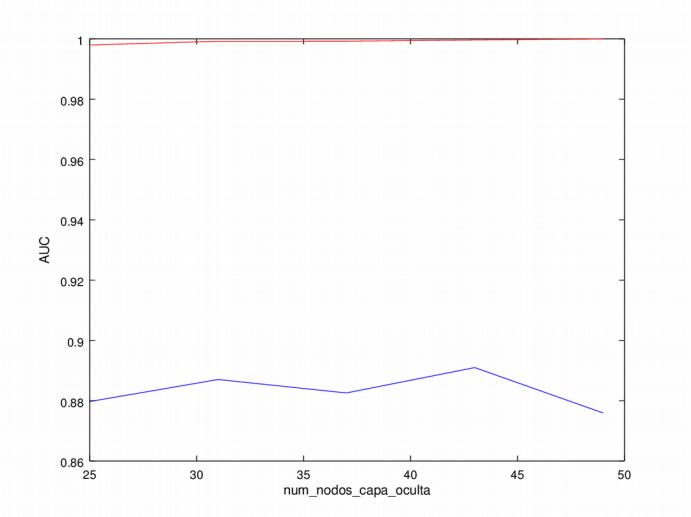


Illustration 19: Num nodos capa oculta = 43

AUC_validation = 0.90181

Rojo: Training

Azul: Validation

(4) Ajuste de nodos por cada capa oculta (Clasificación binaria)

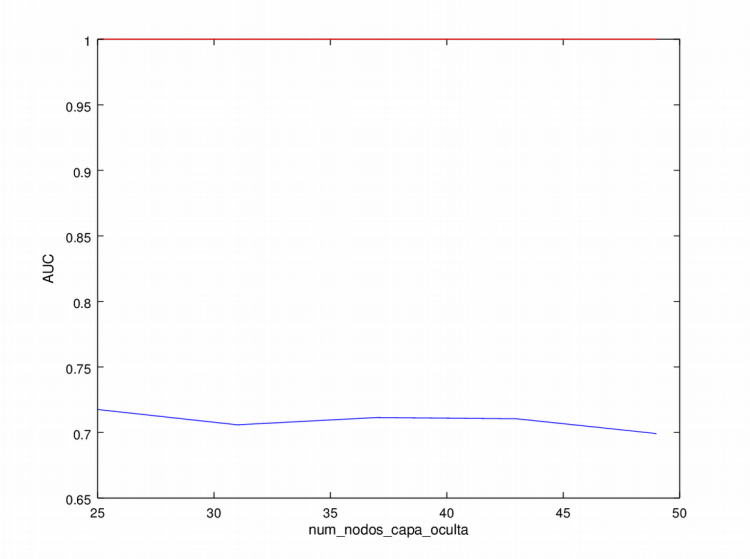


Illustration 20 Num nodos capa oculta = 25

AUC_validation = 0.73911

Rojo: Training

Azul: Validation

(5) Ajuste del número de capas ocultas

En este proyecto no ha sido necesario aumentar el número de capas ocultas ya que el resultado es bueno y lo único que haría sería ralentizar mucho el algoritmo. Se han consultado estudios que afirman que aumentar el número de capas ocultas más allá de 2 o 3 capas no tiene una mejora para nada significativa:

https://www.r-bloggers.com/selecting-the-number-of-neurons-in-the-hidden-layer-of-a-neural-network/

http://francky.me/aifaq/FAQ-comp.ai.neural-net.pdf

(6) Obtención de resultados multiclase

En el caso de las redes neuronales la salida del algoritmo consta de tantos nodos como clases tenga nuestro dataset.

El nodo X simboliza la probabilidad que ha estimado la red neuronal de que dicha instancia procesada sea la clase X.

Por tanto no hace falta realizar ni One vs All ni One vs One, ya que con calcular simplemente cuál es el nodo de la salida con mayor probabilidad, será ese el que haya estimado la red neuronal.

4) SVM (Support vector machines)

Para este proyecto se ha decidido utilizar libsvm para implmentar las SVM debido a su velocidad, precisión y escaso tamaño que poseen.

En la carpeta de código se incluyen los binarios para libsvm compilados para Ubuntu 64 bits(sistema operativo utilizado para realizar el proyecto).

(1) Ajuste de C (clasificación multiclase)

El parametro C define la posición de la línea de decisión en cuanto a la influencia que ejerce cada support vector(puntos que están más cerca de la línea de decisión).

> c → > **varianza** (sobreajuste)

< c → > **sesgo**

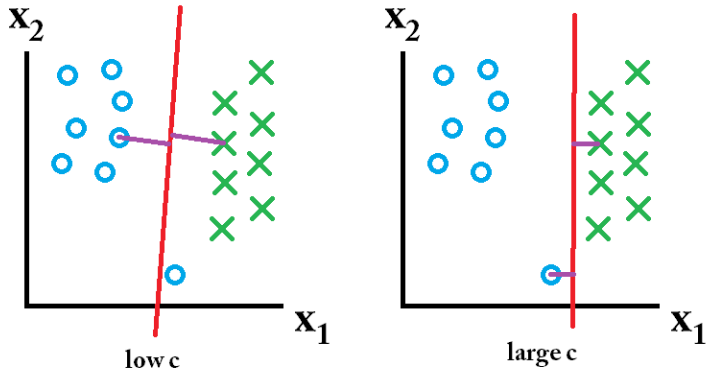


Illustration 21: Parámetro C en SVM

Nota: La imagen superior no corresponde a las instancias de este proyecto ya que en este proyecto al tener que visualizarse los datos con PCA, no se ve con tanta claridad la línea de decisión como se podría ver si solo hubiera realmente 2 ejes.

De todas maneras en la clasificación binaria sí que he generado la imagen de cómo se vería la línea de decisión con los valores óptimos de C y sigma obtenidos (en el apartado 5.2.3 se puede observar ese plot).

En este proyecto el valor óptimo de C en validación ha sido el siguiente:

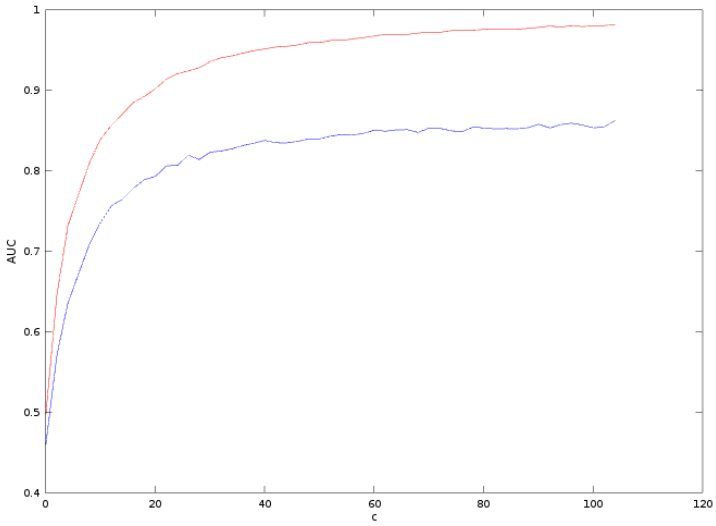


Illustration 22: C = 104.10

AUC_validation = 0.89846

Rojo: Training

Azul: Validation

Nótese que aunque en la gráfica se ve que no llega el AUC hasta ese valor máximo que pone en el pie de la imagen es porque llegará en alguna de las variaciones de sigma por cada C, cosa que no se puede apreciar en esta gráfica ni en la otra, ya que es una combinación de C y de Sigma la que ha producido ese máximo valor de AUC.

(2) Ajuste de C (clasificación binaria)

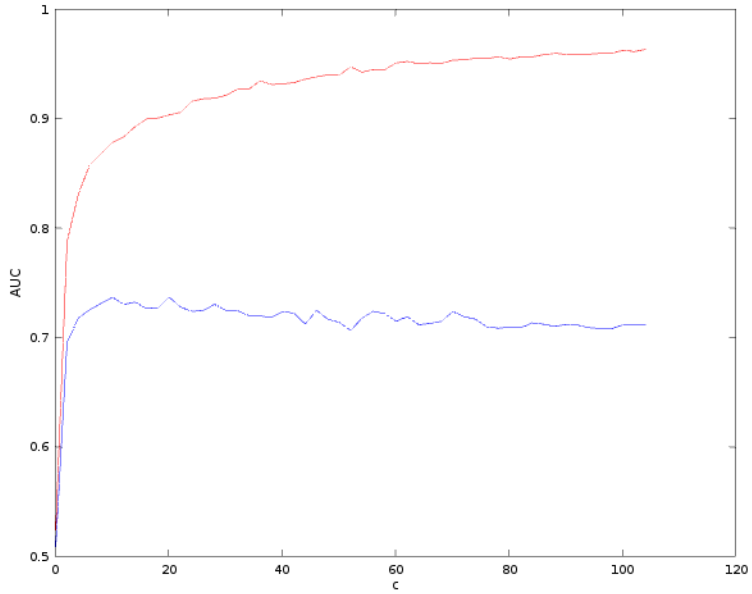


Illustration 23: C = 48.100

AUC_validation = 0.79029

Rojo: Training

Azul: Validation

Nótese que aunque en la gráfica se ve que no llega el AUC hasta ese valor máximo que pone en el pie de la imagen es porque llegará en alguna de las variaciones de sigma por cada C, cosa que no se puede apreciar en esta gráfica ni en la otra, ya que es una combinación de C y de Sigma la que ha producido ese máximo valor de AUC.

(3) Ajuste de sigma (también llamado gamma) (Clasificación multiclase)

El parámetro sigma define la distancia que alcanza la influencia que tiene una instancia para establecer la linea de decisión (decision boundary).

< sigma → > distancia de influencia

> sigma → < distancia de influencia

Mucha distancia de influencia significa que una instancia que está lejos de la línea de decisión(se supone una línea de decisión inicial como para separar 2 clases a la perfección), influye mucho en donde poner dicha línea de decisión.

Dicho de otra manera:

> sigma → > **varianza** (sobreajuste)

< sigma → > **sesgo**

Illustration 25: Sigma alto (varianza)

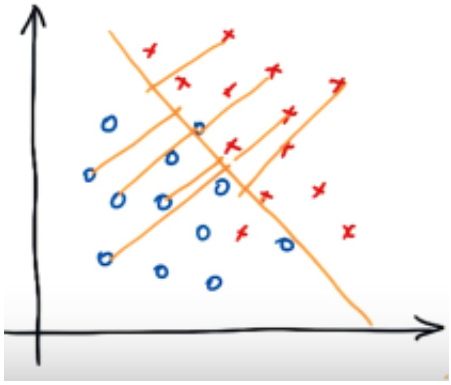


Illustration 24: Sigma más bajo que en la imagen anterior.

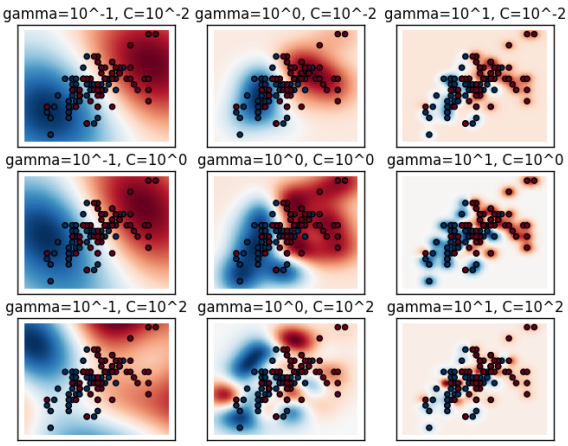


Illustration 26: Variación de C y sigma

Nota: las imágenes superiores no corresponden a las instancias de este proyecto ya que en este proyecto al tener que visualizarse los datos con PCA, no se ve con tanta claridad la línea de decisión como se podría ver si solo hubiera realmente 2 ejes. De todas maneras en la clasificación binaria sí que he generado la imagen de cómo se vería la línea de decisión con los valores óptimos de C y sigma obtenidos.

En este proyecto el valor óptimo de sigma ha sido el siguiente:

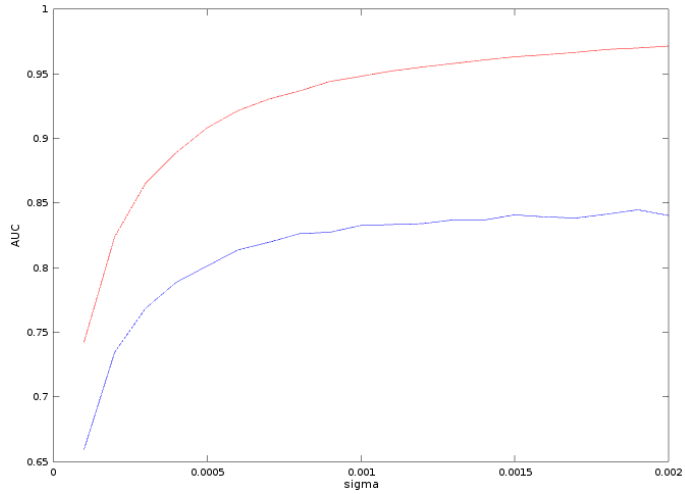


Illustration 27: Sigma = 0.0018000

AUC_validation = 0.89846

Azul: Validation

Rojo: Training

Nótese que aunque en la gráfica se ve que no llega el AUC hasta ese valor máximo que pone en el pie de la imagen es porque llegará en alguna de las variaciones de sigma por cada C, cosa que no se puede apreciar en esta gráfica ni en la otra, ya que es una combinación de C y de Sigma la que ha producido ese máximo valor de AUC.

(4) Ajuste de sigma (también llamado gamma) (Clasificación binaria)

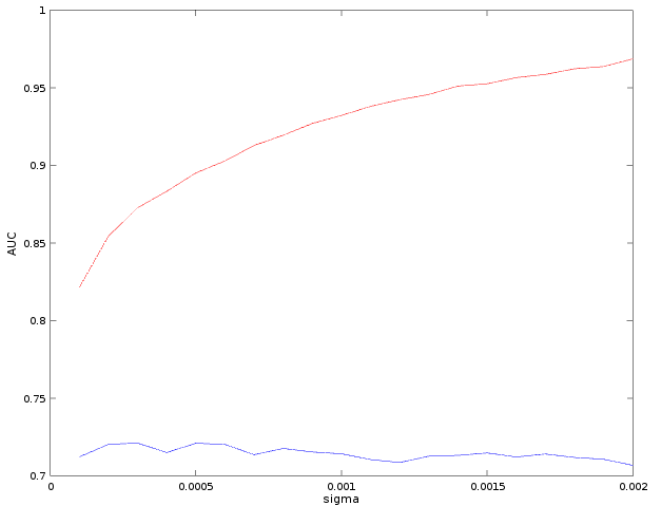


Illustration 28: Sigma = 3.0000e-04

AUC_validation = 0.79029

Rojo: Training

Azul: Validation

Nótese que aunque en la gráfica se ve que no llega el AUC hasta ese valor máximo que pone en el pie de la imagen es porque llegará en alguna de las variaciones de sigma por cada C, cosa que no se puede apreciar en esta gráfica ni en la otra, ya que es una combinación de C y de Sigma la que ha producido ese máximo valor de AUC.

(5) Obtención de resultados multiclase (One vs One)

Al usar LIBSVM, su implementación multiclase se realiza con la técnica One vs One.

Esta técnica consiste en enfrentar cada clase contra cada una de las otras.

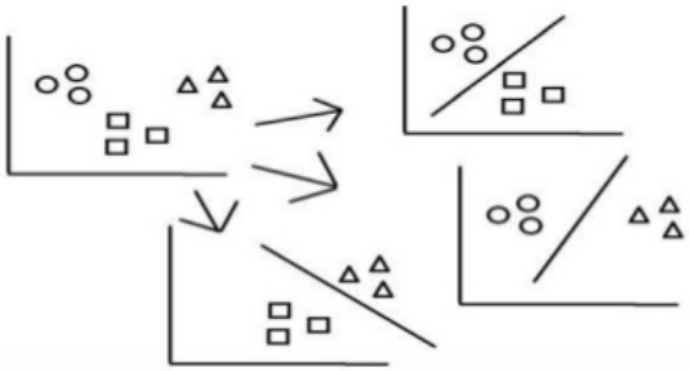


Illustration 29: Esquema One vs One

Hay algunos estudios(https://arxiv.org/pdf/0711.2914.pdf) que afirman que las diferencias en muchos casos entre One vs One y One vs All son prácticamente insignificantes, por eso no se ha tratado de realizar One vs All mediante el modelo resultante de libsvm.

5. Post-procesamiento (Resultados)

Todas las métricas explicadas a continuación son probadas sobre el subconjunto de Test.

1) Métricas utilizadas

(1) True positives, false positives, etc (Matriz de confusión)

Estas métricas contabilizan los aciertos y fallos que se cometen en una clasificación binaria.

		Actual class: 1	Actual class: 0
Predicted class	1	True positive	False Positive
	0	False negative	True negative

Illustration 30: Matriz de confusión:

True positives, false positives, etc

Aunque en principio solo es util para clasificaciones binarias (solo enfrenta la clase real con la estimada), en el proyecto se ha utilizado la técnica One vs All para poder obtener estas métricas en cada clase.

Se han realizado estas métricas en el proyecto ya que son muy útiles cuando hay clases desbalanceadas, para así poder ver si las clases con menos valores se están detectando correctamente.

Si la clasificación fuera perfecta se tendría todo True positives y True negatives.

(2) AUC

El AUC(area under the curve) es una métrica que permite apreciar de manera visual el grado en el que un algoritmo es bueno en base al análisis de la matriz de confusión del apartado anterior.

Trata de enfrentar los true positives VS false positives.

En este proyecto se ha escogido esta métrica como la **principal** para ver cómo de bueno es un algoritmo porque al tener clases desbalanceadas, esta métrica tiene una ventaja muy importante respecto al porcentaje de aciertos comúnmente utilizado:

La ventaja que tiene el AUC se entiende bien mediante este ejemplo:

En un dataset con:

90% no spam

10% spam

Si decimos que todos son no spam, tendríamos un porcentaje de acierto del 90%, lo cual puede parecer muy bueno.

Sin embargo habríamos acertado el 0% de los que no son spam. Por tanto nuestro algoritmo habría sido incapaz de realizar la tarea que se desea(detectar una clase u otra correctamente) y solo con el porcentaje de aciertos podríamos pensar que el algoritmo sí ha sido bueno.

Si por el contrario enfrentamos los true positives VS false positives(la clase positive es spam), nos encontramos con el AUC:

> AUC → mejor algoritmo.

Ya que el AUC se trata de analizar la matriz de confusión, el valor resultante en este ejemplo del spam sería muy bajo, lo cual nos demostraría que en efecto el algoritmo no cumple su cometido.

Al haber considerado el AUC como la métrica principal para ver cómo de bueno es un algoritmo, se ha realizado Cross-Validation teniendo en cuenta el AUC, las comparativas de qué algoritmo es mejor se hacen teniendo en cuenta el AUC, etc.

(3) Porcentaje de aciertos

Esta métrica consiste en calcular cuantas instancias hemos estimado que son la clase que en verdad era y dividir ese valor entre el número de instancias, finalmente se multiplica por 100 el valor obtenido para tenerlo en tanto por ciento.

Aunque, como se ha comentado en el apartado anterior, para este tipo de problemas con clases desbalanceadas no es buena opción tomar el porcentaje de aciertos, en este proyecto también se ha obtenido esta métrica para tener un proyecto más completo, por tener más información.

Normalmente se suele decir que:

> porcentaje de aciertos → mejor algoritmo

Sin embargo como hemos visto en el apartado anterior, esa afirmación no siempre es correcta.

(4) Precision, recall

Estas métricas, al igual que el AUC, se obtienen mediante cálculos realizados con la matriz de confusión. A diferencia del AUC estas métricas son específicas para analizar os (precision) o bien true negatives (recall).

Precision: de todos los estimados como positivos, ¿Cuantos son en verdad positivos?

$$\frac{\text{True positive}}{\#\text{ predicted positives}} = \frac{\text{True positive}}{\text{True pos} + \text{False pos}}$$

Illustration 31: Precision

Recall: de todos los que realmente son positivos, ¿Cuantos hemos estimado como positivos?

$$\frac{\text{True positive}}{\#\text{ actual positives}} = \frac{\text{True positive}}{\text{True pos} + \text{False neg}}$$

Illustration 32: Recall

Como se ha comentado en el apartado de la matriz de confusión, si la clasificación fuera perfecta se tendría todo True positives y True negatives.

Estas métricas realmente no se han tenido muy en cuenta en el proyecto, ya que para analizar los datos de la matriz de confusión, el AUC es una métrica más completa. Aun así, ya que se tenía la matriz de confusión, se ha optado por obtenerlas.

2) Comparativa de resultados entre todos los algoritmos

Los mejores valores entre uno u otro algoritmo están marcados en negrita.

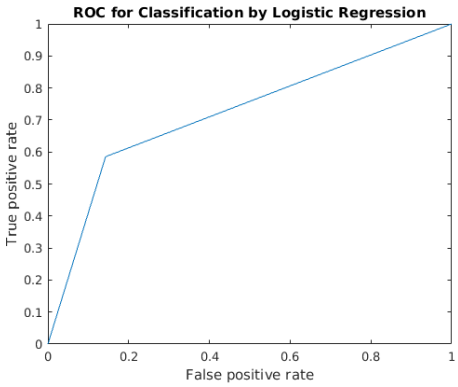
(1) AUC (Clasificación multiclase)

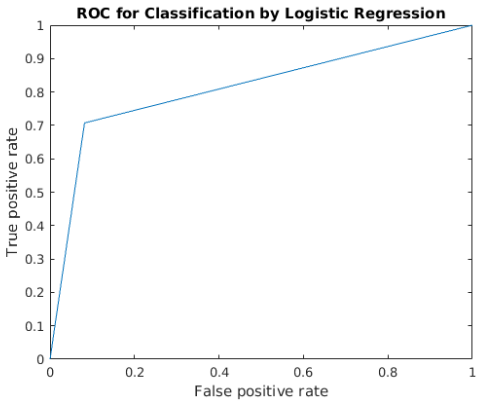
	Regresión logística multiclase	**Redes neuronales**	**SVM**
Clase 1	0.96551	**0.98707**	0.98620
Clase 2	0.65465	0.80380	**0.84384**
Clase 3	0.50427	0.83476	**1.00000**
Clase 4	0.80057	**0.82051**	0.64957
Clase 5	0.45299	0.62678	**0.63533**
Clase 6	0.38261	0.53043	**0.65043**
Clase 7	0.95726	**1.00000**	**1.00000**
Clase 8	0.96610	**1.00000**	**1.00000**
Clase 9	**0.67373**	**0.67373**	**0.67373**
Clase 10	0.64091	**0.76545**	0.70727
Clase 11	0.97414	**1.00000**	**1.00000**
Clase 12	0.96522	**1.00000**	**1.00000**
Clase 13	0.54545	**0.96939**	0.91558
Media	0.72949	0.84707	**0.85092**

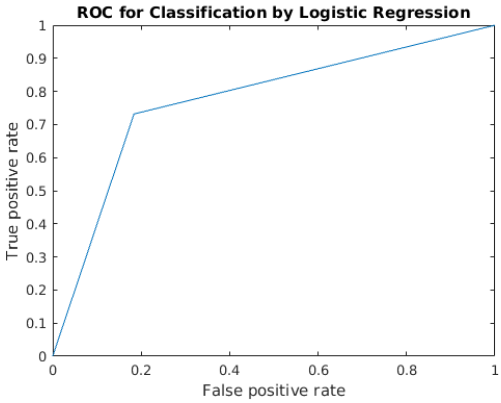
(2) AUC (Clasificación binaria)

Regresión logística multiclase	**Redes neuronales**	**SVM**
Clase 1	0.94923	0.96516	0.97611
Clase 2	0.51369	0.59134	0.64958
Media	0.73146	0.77825	**0.81284**

No se ha realizado el grafico del AUC en la clasificación multiclase porque habría que generar un gráfico por cada clase, o sea 13 gráficos para cada algoritmo.







Nota: para realizar estos gráficos se utilizó la función perfcurve de Matlab (en su versión de 30 días de prueba). Dicho código se incluye en el apéndice.

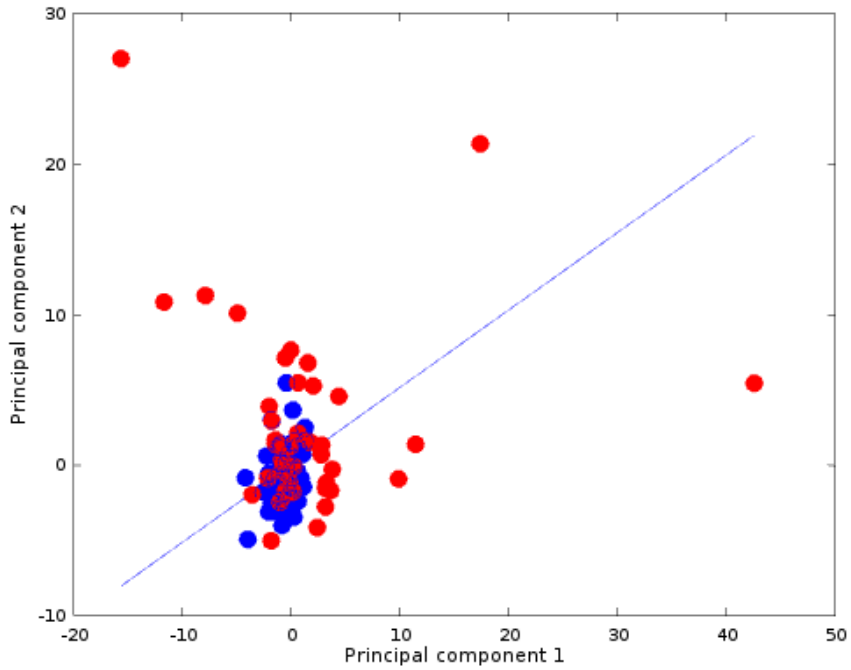
Nota: en el titulo de los plots superiores pone siempre logisitc regression, es una errata, cada uno corresponde al de la columna en la que esté en este apartado(Regresión logística, Redes Neuronales y SVM, por ese orden).

(3) Decision boundary (Clasificación binaria)

Como imagen complementaria se ha creado el plot de la línea de decisión(decision boundary) para la clasificación binaria en SVM.

El motivo por el que se ve un poco extraña(no divide los datos tal cual se esperaría al ojo humano) es por la visualización de los datos con PCA, si realmente solo hubiera 2 ejes, la línea de decisión sí que se realizaría como esperaríamos. Por este motivo, al no ser de gran utilidad este tipo de visualizaciones al hacerse con PCA, se ha decidido no realizar el plot de la decision boundary en el resto de algoritmos.

El motivo por el que se ve una línea totalmente recta es porque los datos son visualizados en 2D y el kernel del SVM utilizado (RBF) lo que hace es proyectar esos datos al menos en 3D para poder realizar su separación correcta, entonces al ver los datos en 2D solo se ve una línea recta.



(4) Porcentaje de aciertos (Clasificación multiclase)

El porcentaje de aciertos se suele entender como el total de todas las clases, ya que para métricas que permitan ver el rendimiento individualmente en cada clase, ya tenemos el AUC o métricas similares.

Regresión logística multiclase	**Redes neuronales**	**SVM**
62.500	82.500	**83.333**

(5) Porcentaje de aciertos (Clasificación binaria)

Regresión logística multiclase	**Redes neuronales**	**SVM**
74.444	78.889	**82.222**

(6) True positives, false positives, etc: matriz de confusión (Clasificación multiclase)

True positives / False positives

Regresión logística multiclase **Redes neuronales** **SVM**

	True positives	False positives	True positives	False positives	True positives	False positives
Clase 1	39	12	44	9	45	12
Clase 2	2	1	5	0	6	2
Clase 3	0	0	2	0	3	0
Clase 4	2	6	2	1	1	1
Clase 5	0	1	1	3	1	0
Clase 6	0	4	1	2	2	0
Clase 7	3	7	3	0	3	0
Clase 8	2	4	**2**	0	2	0
Clase 9	1	0	1	0	1	0
Clase 10	5	0	7	3	6	1
Clase 11	4	3	4	0	4	0
Clase 12	5	7	5	0	5	0
Clase 13	12	0	22	3	21	4
Media	5.7692	3.4615	7.6154	1.6154	**7.6923**	**1.5385**

True negatives / False negatives

	True negatives	False negatives	True negatives	False negatives	True negatives	False negatives
Clase 1	59	10	62	5	59	4
Clase 2	110	7	111	4	109	3
Clase 3	117	3	117	1	117	0
Clase 4	111	1	116	1	116	2
Clase 5	116	3	114	2	117	2
Clase 6	111	5	113	4	115	3
Clase 7	110	0	117	0	117	0
Clase 8	114	0	118	0	118	0
Clase 9	118	1	118	1	118	1
Clase 10	110	5	107	3	109	4
Clase 11	113	0	116	0	116	0
Clase 12	108	0	115	0	115	0
Clase 13	98	10	95	0	94	1
Media	107.31	3.4615	109.15	1.6154	**109.23**	**1.5385**

(7) True positives, false positives, etc: matriz de confusión (Clasificación binaria)

True positives / False positives

Regresión logística multiclase **Redes neuronales** **SVM**

	True positives	False positives	True positives	False positives	True positives	False positives
Clase 1	43	17	44	14	45	12
Clase 2	24	6	27	5	29	4
Media	33.500	11.500	35.500	9.5000	**37**	**8**

True negatives / False negatives

	True negatives	False negatives	True negatives	False negatives	True negatives	False negatives
Clase 1	24	6	27	5	29	4
Clase 2	43	17	44	14	45	12
Media	33.500	11.500	35.500	9.5000	**37**	**8**

(8) Precision, recall (Clasificación multiclase)

Regresión logística multiclase **Redes neuronales** **SVM**

	Precision	Recall	Precision	Recall	Precision	Recall
Clase 1	0.76471	0.79592	0.83019	0.89796	0.78947	0.91837
Clase 2	0.66667	0.22222	1.00000	0.55556	0.75000	0.66667
Clase 3	NaN	0.00000	1.00000	0.66667	1.00000	1.00000
Clase 4	0.25000	0.66667	0.66667	0.66667	0.50000	0.33333
Clase 5	0.00000	0.00000	0.25000	0.33333	1.00000	0.33333
Clase 6	0.00000	0.00000	0.33333	0.20000	1.00000	0.40000
Clase 7	0.30000	1.00000	1.00000	1.00000	1.00000	1.00000
Clase 8	0.33333	1.00000	1.00000	1.00000	1.00000	1.00000
Clase 9	1.00000	0.50000	1.00000	0.50000	1.00000	0.50000
Clase 10	1.00000	0.50000	0.70000	0.70000	0.85714	0.60000
Clase 11	0.57143	1.00000	1.00000	1.00000	1.00000	1.00000
Clase 12	0.41667	1.00000	1.00000	1.00000	1.00000	1.00000
Clase 13	1.00000	0.54545	0.88000	1.00000	0.84000	0.95455
Media	0.52523	0.55617	0.82001	0.73232	**0.90282**	**0.74663**

Nota: La presencia de algún valor NaN se debe a que en el denominador de la formula para obtener estas métricas habría un 0 (lo cual puede ser perfectamente correcto).

(9) Precision, recall (Clasificación binaria)

Regresión logística multiclase **Redes neuronales** **SVM**

	Precision	Recall	Precision	Recall	Precision	Recall
Clase 1	0.71667	0.87755	0.75862	0.89796	0.78947	0.91837
Clase 2	0.80000	0.58537	0.84375	0.65854	0.87879	0.70732
Media	0.75833	0.73146	0.80119	0.77825	**0.83413**	**0.81284**

3) Conclusión

Al haber utilizado el AUC como métrica principal, la lista de los algoritmos ordenados de mejor a peor queda de la siguiente manera, siendo SVM el mejor de los algoritmos en todos los casos (aunque seguido muy de cerca por las redes neuronales en la clasificación multiclase).

- Clasificación multiclase

  1. SVM (**0.85092**)
  2. Redes neuronales (0.84707)
  3. Regresión logística multiclase (0.72949)

- Clasificación binaria:

  1. SVM (**0.81284**)
  2. Redes neuronales (0.77825)
  3. Regresión logística multiclase (0.73146)

6. Uso real en el futuro

La idea es utilizar el modelo obtenido de los algoritmos para estimar nuevos ejemplos reales.

Estos nuevos ejemplos reales serán realizados en un principio en las pruebas del TFG que estoy realizando (monitorización de señales biomédicas mediante Android), en el cual en muy resumidas cuentas trata de crear un ECG portátil de alto rendimiento mediante un microcomputador y analizar su señal.

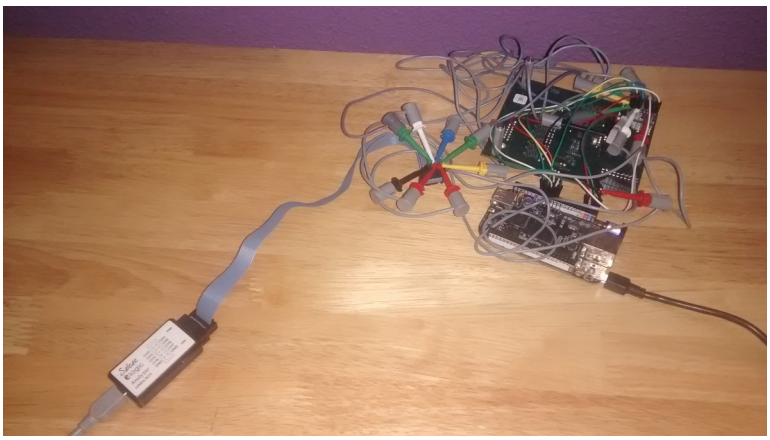


Illustration 33: Microcomputador (BeagleBone black) utilizado en el TFG

Por dicho motivo se plantea poder incluir este modelo para analizar las señales del ECG que recogemos y poder presentarlo mediante interfaz gráfica en Android.

Por otro lado, dado que trabajo en el departamento informático de una clínica sanitaria (CQS Salud), también existiría la posibilidad de poder consultar con compañeros médicos en la efectividad del modelo resultante de este proyecto, y así poder tener información de primera mano de la efectividad del modelo, pudiendo incluso generar nuevos ejemplos de entrenamiento.

7. Apéndice (Código)

1) Flujo de ejecución del código

Para organizar el código de todo el proyecto y para optimizar los tiempos de ejecución en cada paso, se ha optado por estructurar el código del proyecto de la siguiente manera:

Preprocesamiento:

- _1_preprocess_data.m (Para la clasificación multiclase)
- _1_preprocess_data_binary_classes.m (Para la clasificación binaria)

Carga el dataset a partir del fichero .data que se proporciona al descargar el dataset y realiza todos los procedimientos explicados en el apartado 3. Se guarda en un .mat(dataset_preprocessed.mat o dataset_preprocessed_binary.mat para la clasificación binaria) compatible también con matlab todo el dataset ya preprocesado para que no haya que preprocesarlo cada vez que se quiere ejecutar un algoritmo.

Procesamiento: para cada algoritmo hay un fichero que se encarga de realizarlo:

- reg_log_multiclas.m
- _red_neuronal_sinEntrenar_main.m
- _svm_train_ecg.m

Nota: Si se llama a esas funciones poniendo el primer y único argumento a True, se realiza la clasificación binaria. Por defecto si no se pone ningún parámetro(o se pone a false el primer y único parámetro), se realiza la clasificación multiclase.

Se carga el dataset preprocesado y se realiza el entrenamiento del algoritmo, es decir se realiza todo lo explicado en los apartados 4.1.2 y 4.2/4.3/4.4.

Se guarda en un .mat el modelo resultante del algoritmo(dataset_model_reg.mat, dataset_model_neural.mat, dataset_model_svm.mat), para no tener que volver a entrenar si lo que queremos probar es el post-procesamiento o bien para guardarlo ya definitivamente para su uso real. Se guarda además el resultado de probar el subconjunto de Test sobre el modelo resultante de los ciclos de Training+Validation, para así poder ejecutar todas las métricas que incluye el post-procesamiento sin necesidad de volver a ejecutar el modelo.

Nota: En la clasificación binaria se guarda también el AUC obtenido en un fichero aparte(dataset_aucs.mat) para pasárselo a la función perfcurve de Matlab.

Post-procesamiento:

- _3_postprocess.m
- compute_metrics.m

En el mismo fichero se cargan los resultados de haber probado el subconjunto de Test sobre el modelo obtenido por cada algoritmo. Se realizan una serie de cálculos para obtener todas las métricas que nos permiten ver cómo de bueno es cada algoritmo, así se presentan los resultados obtenidos para cada uno de los algoritmos.

Nota: No se incluye en el índice de este documento cada fichero de código porque quedaría un índice demasiado extenso y poco manejable.

2) Preprocesamiento

_1_preprocess_data.m

```
function _1_preprocess_data(all_features)


#0.1. Read and parse the data.
#================================================================================#
fid = fopen("arrhythmia.data");
line = 0;
line_i=1;
while (-1 ~= (line=fgetl(fid)))
 elems = strsplit(line, ",");
 nums = str2double(elems);
 data(line_i++,:)= nums;
end
fclose(fid);

#0.2. Randomize the rows of the data.
#================================================================================#
#data = _randomize_rows_data(data);


#0.3. Get X and Y from data.
#================================================================================#
X = data(:, 1:columns(data)-1); #All columns except the 1st one
Y = data(:, columns(data):columns(data)); #Only the last columns

#The information of the dataset said it has 16 classes, but actually 3 classes
don't
#have any instances, those classes are: 11, 12, 13. So I set the class number of
#14, 15, 16 to 11, 12, 13, so this way we don't have any gap between real
classes.
for i=13:16
  class_to_substitute = find(i==Y);
  Y(class_to_substitute) = i-3;
endfor;

number_instances = rows(X);
number_features = columns(X);
number_classes = 13;


mean_feature = zeros(number_features, 1);


#0.4. Feature selection: based in real life medical procedures:
#================================================================================#
#{
En la vida real los ECG se miden normalmente con 12 o 13 canales(cables), como
en este
dataset (12 canales). Sin embargo al realizar los holter(ECG portatil durante al
menos 24 horas),
suele ser bastantes menos canales, en mi caso la ultima vez solo fueron 3.
```

```
En nuestro TFG utilizamos algo intermedio: 8 canales, dado que su uso podra ser
como
holter o como simplemente un ecg comun de una clinica para medir durante unos
segundos.

Por ese motivo la seleccion de features la realizare con 8 canales. Descartamos
por tanto
los canales DI, DII, DIII (por ser los cables que se ponen en las extremidades y
no es
habitual ponerlos) y el V6 por estar muy cerca del V5 y no perder asi casi
informacion.

     Of channel DI:
      16 .. 27
      160 .. 169
     Of channel DII:
      28 .. 39
      170 .. 179
     Of channels DIII:
      40 .. 51
      180 .. 189
     Of channel AVR:
      52 .. 63
      190 .. 199
     Of channel AVL:
      64 .. 75
      200 .. 209
     Of channel AVF:
      76 .. 87
      210 .. 219
     Of channel V1:
      88 .. 99
      220 .. 229
     Of channel V2:
      100 .. 111
      230 .. 239
     Of channel V3:
      112 .. 123
      240 .. 249
     Of channel V4:
      124 .. 135
      250 .. 259
     Of channel V5:
      136 .. 147
      260 .. 269
     Of channel V6:
      148 .. 159
      270 .. 279
#}

features_to_delete = [];
DI = [16:1:27];
DI = [DI, [160:1:169]];

DII = [28:1:39];
DII = [DII, [170:1:179]];

DIII = [40:1:51];
DIII = [DIII, [180:1:189]];
```

```
V6 = [148:1:159];
V6 = [V6, [270:1:279]];

features_to_delete = [DI, DII, DIII, V6];


if (exist("all_features", "var"))
  if (!all_features)
    X(:, features_to_delete) = []; #Delete the columns previously chosen
  endif;
else
  X(:, features_to_delete) = []; #Delete the columns previously chosen
endif;

#Recompute the rows and columns
number_instances = rows(X);
number_features = columns(X);

#1.1 Delete the features that are invariant (i.e. minimum = maximum in all the
rows)
#=============================================================================#
columns_to_delete = [];
for i=1:number_features
  min_of_feature = min(X(:, i));
  max_of_feature = max(X(:, i));
  if min_of_feature==max_of_feature #Delete that feature (i.e that column)
    columns_to_delete = [columns_to_delete, i];
  endif;
  mean_feature(i) = mean(X(:, i)(~isnan(X(:, i))));
endfor;


#2. Set values in the features that are unknown (?):
#=============================================================================#
#We did the mean of all the instances for that feature and now we set the
unknowns
#values to that mean, because we don't want to delete all the instance
#because just a few features of that instance are unknown.
for i=1:number_instances
  indices = isnan(X(i, :));
  indices = find(indices==1);
  for a=1:columns(indices) #If indices is empty, it won't enters here
    X(i, indices(a)) = mean_feature(indices(a));
  endfor;
endfor;

#1.2 Delete the features that are invariant (i.e. minimum = maximum in all the
rows)
#=============================================================================#
#We delete them now, and not before because we needed all the means of the
features
#and if we deleted these columns now, the indices would have disarrange
X(:, columns_to_delete) = []; #Delete the columns previously chosen

#Recompute the rows and columns
number_instances = rows(X);
number_features = columns(X);

#3. Normalization of the data:
#=============================================================================#
```

```
[X mu1 sigma1] = featureNormalize_firstTime(X);


#5. PCA (Principal component analysis):
#=============================================================================#
k = 2; #Number of PC(principal components) we want. We want 2 so we can plot the
data
sigma = (1/number_features)*(X'*X);
[U, S, V] = svd(sigma);
Ureduce = U(:, 1:k);
z = X*Ureduce;

_plot_pca(z, Y);



#6. Set training, validation and test subsets:
#=============================================================================#
#{
I use 3 subsets: training, validation, and test.

-----------------------------------------------------
Training set + Validation set: are used to train the model.
-----------------------------------------------------
I train the model with the training set, and then I try their accuracy and AUC
in
the validation set(this is called cross validation). So I do that method several
times
and the results of trying the model in the validation tests are useful to adjust
the
parameter of each machine learning algorithm, so the validation test influences
the model.

The method I use to perform the training influenced by a validation test is
called:
--------------------------
k-fold cross validation
--------------------------
This method consists in partitioning the training set in 2 sets(trainig set and
validation set)
The number of instances in each set is set by establishign a value: k.

k is the number of chunks the training set is divided. For example, if we have a
training
set of 100 instances, if we set k=10, we'll have 10 chunks, and each one would
have 10 values.

So we perform a loop of k(=10) iterations, and inside each iteration, following
the previous example
we randomly divide the training set into training set(90 instances) and
validation test(10 instances), and we train the model with the training test and
test it in
the validation test.

We do that loop n-times (we establish n to the value tdesired), and finally we
get
what has been the best model and what were their parameters.
```

```
-------------------------------------------------------
Test set: it's used to test the accuracy and AUC of the model in a "real world"
environment
-------------------------------------------------------
It's like a "real world" environment" because this instances were never used to
influence
the achieved model.


I use a slit known as stratified, i.e it's not random but it stands for doing
balanced
partition, so if we do 2 divisions(in fact we do 3(training+validation+test),
this is just
a example), of 70% and 30%, and in total we have 10 instances of class 9, in the
division we
would have 7 instances of class 9 in the 1st subset, and 3 instances of class 9
in the 2nd
subset.

I follow the recommendation a 60%(training set)-20%(validation set)-20%(test
set).
#}

training_subtest_split_percentage = 0.6;
validation_subtest_split_percentage = 0.2;
test_subtest_split_percentage = 0.2;

training_val_split_percentage = training_subtest_split_percentage +
validation_subtest_split_percentage;

k_folds_cross_val = 4; #So, training+validation = 80%, and 80%/4 = 20%
(validation test)

number_instances_training_subset =
number_instances*training_subtest_split_percentage;
number_instances_validation_subset =
number_instances*validation_subtest_split_percentage;
number_instances_test_subset = number_instances*test_subtest_split_percentage;

number_instances_training_and_val_subset = number_instances_training_subset +
number_instances_validation_subset;


X_train_val = zeros(0, number_features);
X_test = zeros(0, number_features);
Y_train_val = zeros(0, 1);
Y_test = zeros(0, 1);

X_train_val_oversampled = zeros(0, number_features);
Y_train_val_oversampled = zeros(0, 1);


overfitted_classes = [];

for i=1:number_classes
  i_class = find(i==Y);
  number_instances_i_class = rows(i_class);
```

```
  if number_instances_i_class > 2
    number_instances_i_class_train_val =
round(number_instances_i_class*training_val_split_percentage);
    number_instances_i_class_test =
round(number_instances_i_class*test_subtest_split_percentage);
  else
    number_instances_i_class_train_val = round(number_instances_i_class*0.5);
    number_instances_i_class_test = round(number_instances_i_class*0.5);
  endif
  i_class_train_val = i_class(1:number_instances_i_class_train_val);
  i_class_test =
i_class(number_instances_i_class_train_val+1:number_instances_i_class);


  #Overfitting of classes impossible to detect without overfitting (because they
are so similar to healthy class)
  if (i == 7 || i==8 || i == 11 || i ==12 || i ==13)
      i_class_train_val = i_class(1:end);

      X_train_val = [X_train_val; X(i_class_train_val, :)];
      X_train_val_oversampled = [X_train_val_oversampled;
X(i_class_train_val, :)];
      X_test = [X_test; X(i_class_train_val, :)];

      Y_train_val = [Y_train_val; Y(i_class_train_val, :)];
      Y_train_val_oversampled = [Y_train_val_oversampled;
Y(i_class_train_val, :)];
      Y_test = [Y_test; Y(i_class_train_val, :)];
  endif;




      #Oversampling of classes with very few instances
    if (number_instances_i_class < 10)
    #I duplicate the values(15 times: 1(original insert)+14 duplications)
    #to achieve the oversampling on these classes
        duplicate_times = 14;

        if (i != 7 && i!=8 && i != 11 && i != 12 && i != 13)
          nuevas_filas_x_train_val = X(i_class_train_val, :);
          nuevas_filas_y_train_val = Y(i_class_train_val, :);

         for (a=1:duplicate_times)
            X_train_val_oversampled = [X_train_val_oversampled;
nuevas_filas_x_train_val];
            Y_train_val_oversampled = [Y_train_val_oversampled;
nuevas_filas_y_train_val];
         endfor

       endif;
    endif




     #Rest of the classes, I don't do nothing special with them
```

```
    if (i != 7 && i!=8 && i != 11 && i != 12 && i != 13) # &&
(number_instances_i_class >= 10)
      X_train_val = [X_train_val; X(i_class_train_val, :)];
      X_train_val_oversampled = [X_train_val_oversampled;
X(i_class_train_val, :)];
      X_test = [X_test; X(i_class_test, :)];

      Y_train_val = [Y_train_val; Y(i_class_train_val, :)];
      Y_train_val_oversampled = [Y_train_val_oversampled;
Y(i_class_train_val, :)];
      Y_test = [Y_test; Y(i_class_test, :)];
    endif;
number_instances = rows(X);
number_features = columns(X);



endfor;

save -mat7-binary 'dataset_preprocessed.mat', 'X_train_val', 'Y_train_val',
'X_test', 'Y_test', 'training_val_split_percentage', 'k_folds_cross_val',
'overfitted_classes', 'X_train_val_oversampled', 'Y_train_val_oversampled';

endfunction
```

_1_preprocess_data_binary_classes.m

```
function _1_preprocess_data_binary_classes(all_features)



#0.1. Read and parse the data.
#============================================================================#
fid = fopen("arrhythmia.data");
line = 0;
line_i=1;
while (-1 ~= (line=fgetl(fid)))
 elems = strsplit(line, ",");
 nums = str2double(elems);
 data(line_i++,:)= nums;
end
fclose(fid);

#0.2. Randomize the rows of the data.
#============================================================================#
#data = _randomize_rows_data(data);


#0.3. Get X and Y from data.
#============================================================================#
X = data(:, 1:columns(data)-1); #All columns except the 1st one
Y = data(:, columns(data):columns(data)); #Only the last columns

#The information of the dataset said it has 16 classes, but actually 3 classes
don't
#have any instances, those classes are: 11, 12, 13. So I set the class number of
#14, 15, 16 to 11, 12, 13, so this way we don't have any gap between real
classes.
```

```
for i=13:16
  class_to_substitute = find(i==Y);
  Y(class_to_substitute) = i-3;
endfor;

not_healthy = find(Y!=1);
Y(not_healthy) = 2;

number_instances = rows(X);
number_features = columns(X);
number_classes = 2;


mean_feature = zeros(number_features, 1);


#0.4. Feature selection: based in real life medical procedures:
#=============================================================================#
#{
En la vida real los ECG se miden normalmente con 12 o 13 canales(cables), como
en este
dataset (12 canales). Sin embargo al realizar los holter(ECG portatil durante al
menos 24 horas),
suele ser bastantes menos canales, en mi caso la ultima vez solo fueron 3.

En nuestro TFG utilizamos algo intermedio: 8 canales, dado que su uso podra ser
como
holter o como simplemente un ecg comun de una clinica para medir durante unos
segundos.

Por ese motivo la seleccion de features la realizare con 8 canales. Descartamos
por tanto
los canales DI, DII, DIII (por ser los cables que se ponen en las extremidades y
no es
habitual ponerlos) y el V6 por estar muy cerca del V5 y no perder asi casi
informacion.

     Of channel DI:
      16 .. 27
      160 .. 169
     Of channel DII:
      28 .. 39
      170 .. 179
     Of channels DIII:
      40 .. 51
      180 .. 189
     Of channel AVR:
      52 .. 63
      190 .. 199
     Of channel AVL:
      64 .. 75
      200 .. 209
     Of channel AVF:
      76 .. 87
      210 .. 219
     Of channel V1:
      88 .. 99
      220 .. 229
     Of channel V2:
      100 .. 111
```

```
      230 .. 239
     Of channel V3:
      112 .. 123
      240 .. 249
     Of channel V4:
      124 .. 135
      250 .. 259
     Of channel V5:
      136 .. 147
      260 .. 269
     Of channel V6:
      148 .. 159
      270 .. 279
#}

features_to_delete = [];
DI = [16:1:27];
DI = [DI, [160:1:169]];

DII = [28:1:39];
DII = [DII, [170:1:179]];

DIII = [40:1:51];
DIII = [DIII, [180:1:189]];

V6 = [148:1:159];
V6 = [V6, [270:1:279]];

features_to_delete = [DI, DII, DIII, V6];


if (exist("all_features", "var"))
  if (!all_features)
    X(:, features_to_delete) = []; #Delete the columns previously chosen
  endif;
else
  X(:, features_to_delete) = []; #Delete the columns previously chosen
endif;

#Recompute the rows and columns
number_instances = rows(X);
number_features = columns(X);

#1.1 Delete the features that are invariant (i.e. minimum = maximum in all the
rows)
#=============================================================================#
columns_to_delete = [];
for i=1:number_features
  min_of_feature = min(X(:, i));
  max_of_feature = max(X(:, i));
  if min_of_feature==max_of_feature #Delete that feature (i.e that column)
    columns_to_delete = [columns_to_delete, i];
  endif;
  mean_feature(i) = mean(X(:, i)(~isnan(X(:, i))));
endfor;


#2. Set values in the features that are unknown (?):
#=============================================================================#
```

```
#We did the mean of all the instances for that feature and now we set the
unknowns
#values to that mean, because we don't want to delete all the instance
#because just a few features of that instance are unknown.
for i=1:number_instances
  indices = isnan(X(i, :));
  indices = find(indices==1);
  for a=1:columns(indices) #If indices is empty, it won't enters here
    X(i, indices(a)) = mean_feature(indices(a));
  endfor;
endfor;

#1.2 Delete the features that are invariant (i.e. minimum = maximum in all the
rows)
#=============================================================================#
#We delete them now, and not before because we needed all the means of the
features
#and if we deleted these columns now, the indices would have disarrange
X(:, columns_to_delete) = []; #Delete the columns previously chosen
#Recompute the rows and columns
number_instances = rows(X);
number_features = columns(X);

#3. Normalization of the data:
#=============================================================================#
[X mu1 sigma1] = featureNormalize_firstTime(X);


#5. PCA (Principal component analysis):
#=============================================================================#
k = 2; #Number of PC(principal components) we want. We want 2 so we can plot the
data
sigma = (1/number_features)*(X'*X);
[U, S, V] = svd(sigma);
Ureduce = U(:, 1:k);
z = X*Ureduce;

_plot_pca(z, Y);


#6. Set training, validation and test subsets:
#=============================================================================#
#{
I use 3 subsets: training, validation, and test.

-----------------------------------------------------
Training set + Validation set: are used to train the model.
-----------------------------------------------------
I train the model with the training set, and then I try their accuracy and AUC
in
the validation set(this is called cross validation). So I do that method several
times
and the results of trying the model in the validation tests are useful to adjust
the
parameter of each machine learning algorithm, so the validation test influences
the model.

The method I use to perform the training influenced by a validation test is
called:
-------------------------
```

```
k-fold cross validation
-------------------------
This method consists in partitioning the training set in 2 sets(trainig set and
validation set)
The number of instances in each set is set by establishign a value: k.

k is the number of chunks the training set is divided. For example, if we have a
training
set of 100 instances, if we set k=10, we'll have 10 chunks, and each one would
have 10 values.

So we perform a loop of k(=10) iterations, and inside each iteration, following
the previous example
we randomly divide the training set into training set(90 instances) and
validation test(10 instances), and we train the model with the training test and
test it in
the validation test.

We do that loop n-times (we establish n to the value tdesired), and finally we
get
what has been the best model and what were their parameters.



-------------------------------------------------------------
Test set: it's used to test the accuracy and AUC of the model in a "real world"
environment
-------------------------------------------------------------
It's like a "real world" environment" because this instances were never used to
influence
the achieved model.


I use a split known as stratified, i.e it's not random but it stands for doing
balanced
partition, so if we do 2 divisions(in fact we do 3(training+validation+test),
this is just
a example), of 70% and 30%, and in total we have 10 instances of class 9, in the
division we
would have 7 instances of class 9 in the 1st subset, and 3 instances of class 9
in the 2nd
subset.

I follow the recommendation a 60%(training set)-20%(validation set)-20%(test
set).
#}

training_subtest_split_percentage = 0.6;
validation_subtest_split_percentage = 0.2;
test_subtest_split_percentage = 0.2;

training_val_split_percentage = training_subtest_split_percentage +
validation_subtest_split_percentage;

k_folds_cross_val = 4; #So, training+validation = 80%, and 80%/4 = 20%
(validation test)

number_instances_training_subset =
number_instances*training_subtest_split_percentage;
```

```
number_instances_validation_subset =
number_instances*validation_subtest_split_percentage;
number_instances_test_subset = number_instances*test_subtest_split_percentage;

number_instances_training_and_val_subset = number_instances_training_subset +
number_instances_validation_subset;


X_train_val = zeros(0, number_features);
X_test = zeros(0, number_features);
Y_train_val = zeros(0, 1);
Y_test = zeros(0, 1);
for i=1:number_classes
  i_class = find(i==Y);
  number_instances_i_class = rows(i_class);

  if number_instances_i_class > 2
    number_instances_i_class_train_val =
round(number_instances_i_class*training_val_split_percentage);
    number_instances_i_class_test =
round(number_instances_i_class*test_subtest_split_percentage);
  else
    number_instances_i_class_train_val = round(number_instances_i_class*0.5);
    number_instances_i_class_test = round(number_instances_i_class*0.5);
  endif
  i_class_train_val = i_class(1:number_instances_i_class_train_val);
  i_class_test =
i_class(number_instances_i_class_train_val+1:number_instances_i_class);


  X_train_val = [X_train_val; X(i_class_train_val, :)];
  X_test = [X_test; X(i_class_test, :)];

  Y_train_val = [Y_train_val; Y(i_class_train_val, :)];
  Y_test = [Y_test; Y(i_class_test, :)];



number_instances = rows(X);
number_features = columns(X);


endfor;

save -mat7-binary 'dataset_preprocessed_binary.mat', 'X_train_val',
'Y_train_val', 'X_test', 'Y_test', 'training_val_split_percentage',
'k_folds_cross_val';

endfunction
```

featureNormalize_firstTime.m

```
function [X_norm, mu, sigma] = featureNormalize_firstTime(X)

%FEATURENORMALIZE Normalizes the features in X
%   FEATURENORMALIZE(X) returns a normalized version of X where
%   the mean value of each feature is 0 and the standard deviation
%   is 1. This is often a good preprocessing step to do when
%   working with learning algorithms.


mu = mean(X);
X_norm = bsxfun(@minus, X, mu);

sigma = std(X_norm);
X_norm = bsxfun(@rdivide, X_norm, sigma);


% ============================================================


endfunction
```

featureNormalize_otherTimes.m

```
function X_norm = _normalize_mu_sigma(X, mu, sigma)

X_norm = bsxfun(@minus, X, mu);
X_norm = bsxfun(@rdivide, X_norm, sigma);

endfunction
```

_plot_pca.m

```
function _plot_pca(z, Y)


number_instances = rows(z);

for i=1:number_instances
  if(Y(i) == 1)
    scatter(z(i, 1), z(i, 2), 10, "blue", "filled");
  endif;

  if(Y(i) == 2)
    scatter(z(i, 1), z(i, 2), 10,  "red", "filled");
  endif;

  if(Y(i) == 3)
    scatter(z(i, 1), z(i, 2), 10,  "black", "filled");
  endif;

  if(Y(i) == 4)
    scatter(z(i, 1), z(i, 2), 10,  "yellow", "filled");
  endif;
```

```
  if(Y(i) == 5)
    scatter(z(i, 1), z(i, 2), 10,  "magenta", "filled");
  endif;

  if(Y(i) == 6)
    scatter(z(i, 1), z(i, 2), 10,  [126 26 125] ./ 255, "filled");
  endif;

  if(Y(i) == 7)
    scatter(z(i, 1), z(i, 2), 10,  [53 178 82] ./ 255, "filled");
  endif;

  if(Y(i) == 8)
    scatter(z(i, 1), z(i, 2), 10,  [100 038 32] ./ 255, "filled");
  endif;

  if(Y(i) == 9)
    scatter(z(i, 1), z(i, 2), 10,  [233 31 23] ./ 255, "filled");
  endif;

  if(Y(i) == 10)
    scatter(z(i, 1), z(i, 2), 10,  [121 98 121] ./ 255, "filled");
  endif;

  if(Y(i) == 11)
    scatter(z(i, 1), z(i, 2), 10,  "yellow", "filled");
  endif;

  if(Y(i) == 12)
    scatter(z(i, 1), z(i, 2), 10,  [012 100 211] ./ 255, "filled");
  endif;

  if(Y(i) == 13)
    scatter(z(i, 1), z(i, 2), 10,  [31 18 12] ./ 255, "filled");
  endif;

  hold on;
endfor;
xlabel("Principal component 1", "fontsize", 11);
ylabel("Principal component 2", "fontsize", 11);

endfunction
```

_randomize_rows_data.m

```
function [data_randomized] = _randomize_rows_data(data)


rows_shuffled = randperm(rows(data));
data_randomized = data(rows_shuffled,:);

endfunction
```

3) Procesamiento

(1) Regresión logística

reg_log_multiclas.m

```
function porcentajeCorrectos = reg_log_multiclas(binary_classes)

more off;

#Selección de si se ha metido el parametro binary_classes(para que solo haya
sanos-no_sanos
#o en su defecto que sean las 13 clases originales
if (exist("binary_classes", "var"))
  if (binary_classes)
    num_etiquetas = 2;
    etiquetas = [1 2];
    load("dataset_preprocessed_binary.mat"); #Guarda los datos de entrada en X e
Y
  else
    num_etiquetas = 13;
    etiquetas = [1 2 3 4 5 6 7 8 9 10 11 12 13];
    load("dataset_preprocessed.mat"); #Guarda los datos de entrada en X e Y
  endif;
else
  num_etiquetas = 13;
  etiquetas = [1 2 3 4 5 6 7 8 9 10 11 12 13];
   load("dataset_preprocessed.mat"); #Guarda los datos de entrada en X e Y
endif

#lambda = [0:0.025:3];
lambda = [0:0.005:1];
n_iteraciones = length(lambda);
aucs_mean = zeros(length(lambda), 1);
aucs_one_fold = zeros(k_folds_cross_val, 1);

aucs_mean_training = zeros(length(lambda), 1);
aucs_one_fold_training = zeros(k_folds_cross_val, 1);

for i=1:length(lambda)
  for k_fold=1:k_folds_cross_val
      #Training:

#=============================================================================#
      [X_train_fold Y_train_fold X_val_fold Y_val_fold] =
split_fold(X_train_val, Y_train_val,training_val_split_percentage,
k_folds_cross_val, k_fold);
      #===Se le añade el termino x0=1 a todos los casos de entrada========
      x0 = ones(rows(X_train_fold),1);
      X_train_fold = [x0, X_train_fold];
      #====================================================================

      thetas = oneVsAll(X_train_fold, Y_train_fold, num_etiquetas, lambda(i));

      #Cross validation:

#=============================================================================#
```

```
num_casos = rows(X_val_fold);
#===Se le añade el termino x0=1 a todos los casos de entrada========
x0 = ones(rows(X_val_fold),1);
X_val_fold = [x0, X_val_fold];

h0 = X_val_fold*thetas';
[maximo indicesMax] = max(h0');
correctos=indicesMax'==Y_val_fold;
correctos = sum(correctos);
porcentajeCorrectos = (correctos/num_casos)*100

predicted_labels = indicesMax';
auc = zeros(num_etiquetas, 1);
for a=1:num_etiquetas
  a_class = find(a==Y_val_fold);
  number_instances_a_class = rows(a_class);

  ind_not_this_class = find(a!=Y_val_fold);
  Y_tmp = Y_val_fold;
  Y_tmp(ind_not_this_class) = -1;
  Y_tmp(a_class) = 1;


  predicted_a_class = find(a==predicted_labels);
  predicted_not_this_class = find(a!=predicted_labels);
  predicted_tmp = predicted_labels;
  predicted_tmp(predicted_a_class) = 1;
  predicted_tmp(predicted_not_this_class) = -1;


  auc(a) = auc_compute(Y_tmp, predicted_tmp, 1)
 endfor;

auc_mean = mean(auc(~isnan(auc)));
aucs_one_fold(k_fold) = auc_mean
```

```
  #==================Sobre training=====================================#
num_casos = rows(X_train_fold);
#===Se le añade el termino x0=1 a todos los casos de entrada========
#x0 = ones(rows(X_train_fold),1);
#X_train_fold = [x0, X_train_fold];

h0 = X_train_fold*thetas';
[maximo indicesMax] = max(h0');
correctos=indicesMax'==Y_train_fold;
correctos = sum(correctos);
porcentajeCorrectos = (correctos/num_casos)*100

predicted_labels = indicesMax';
auc_training = zeros(num_etiquetas, 1);
for a=1:num_etiquetas
  a_class = find(a==Y_train_fold);
  number_instances_a_class = rows(a_class);
```

```
        ind_not_this_class = find(a!=Y_train_fold);
        Y_tmp = Y_train_fold;
        Y_tmp(ind_not_this_class) = -1;
        Y_tmp(a_class) = 1;


        predicted_a_class = find(a==predicted_labels);
        predicted_not_this_class = find(a!=predicted_labels);
        predicted_tmp = predicted_labels;
        predicted_tmp(predicted_a_class) = 1;
        predicted_tmp(predicted_not_this_class) = -1;


        auc_training(a) = auc_compute(Y_tmp, predicted_tmp, 1)
       endfor;

      auc_mean_training = mean(auc_training(~isnan(auc_training)));
      aucs_one_fold_training(k_fold) = auc_mean_training

    endfor;

    aucs_mean(i) = mean(aucs_one_fold(~isnan(aucs_one_fold)))
    aucs_mean_training(i) =
mean(aucs_one_fold_training(~isnan(aucs_one_fold_training)))

    for zz=1:1000 #Para ver por la salida estándar por donde va el algoritmo
      i
    endfor;

endfor;
#=====================================================================

figure(4);
plot(lambda, aucs_mean);
hold on;
plot(lambda, aucs_mean_training, "r");
xlabel("lambda", "fontsize", 11);
ylabel("AUC", "fontsize", 11);

#========Cálculo de qué etiqueta hemos estimado que es cada
ejemplo==============
num_casos = rows(X_test);

[best_model_I, ind_best_model_i] = max(aucs_mean);
best_lambda = lambda(ind_best_model_i);
#===Se le añade el termino x0=1 a todos los casos de entrada========
if (exist("binary_classes", "var"))
  if (binary_classes)
      x0 = ones(rows(X_train_val),1);
      X_train_val = [x0, X_train_val];

     thetas = oneVsAll(X_train_val, Y_train_val, num_etiquetas, best_lambda);
  else
    x0 = ones(rows(X_train_val_oversampled),1);
    X_train_val_oversampled = [x0, X_train_val_oversampled];

    thetas = oneVsAll(X_train_val_oversampled, Y_train_val_oversampled,
num_etiquetas, best_lambda);
  endif;
```

```
else
    x0 = ones(rows(X_train_val_oversampled),1);
    X_train_val_oversampled = [x0, X_train_val_oversampled];

    thetas = oneVsAll(X_train_val_oversampled, Y_train_val_oversampled,
num_etiquetas, best_lambda);
endif;


#===Se le añade el termino x0=1 a todos los casos de entrada========
x0 = ones(rows(X_test),1);
X_test = [x0, X_test];

h0 = X_test*thetas';
[maximo indicesMax] = max(h0');
correctos=indicesMax'==Y_test;
correctos = sum(correctos);
porcentajeCorrectos = (correctos/num_casos)*100
predicted_labels = indicesMax';

predicted_labels_reg = predicted_labels;
Y_test_reg = Y_test;
model_reg = thetas;
#{
Se sacan predicted labels con h0 = X_test*thetas';
[maximo indicesMax] = max(h0');
predicted_labels = indicesMax';
#}
save -mat7-binary 'dataset_model_reg.mat', 'predicted_labels_reg', 'Y_test_reg',
'model_reg';

best_lambda
[best_aucs_mean, ind_best_model_i] = max(aucs_mean)
[best_aucs_mean_training, ind_best_model_it] = max(aucs_mean_training)

#En indicesMax se guarda la posicion en la que está el valor máximo de cada
#ejemplo. Es decir es a nivel fila. Si el índice es 5 por ejemplo, querrá
#decir que la posicion máxima era el 5 para ese ejemplo, por lo que nuestro
programa
#habrá estimado que el valor para esa imagen es un 5

#h0: Para cada caso se saca cuantas probabilidades tiene de ser x etiqueta.
#Es decir h0 es un vector en el cual el número más alto corresponde a la
#probabilidad de ser esa etiqueta, así que nos quedamos con esa.
#Ejemplo h0 = [13, 16, 95] nos quedaríamos con 95, ya que se podría decir que
hay
#un 95% de probabilidad de que sea esa etiqueta (las etiquetas en este caso
serían
# [1,2,3] por ejemplo. La conclusión sería que hemos estimado que ese caso es un
3

#Nota: Se podría haber hecho la función sigmoide sobre h0, pero no es necesario
en este caso.
#Al no hacerla simplemente tenemos un rango de valores más allá de 0 a 1, pero
no nos influye
#para sacar el máximo valor.
#================================================================
```

```
endfunction
```

oneVsAll.m

```
function [thetas] = oneVsAll(X, y, num_etiquetas, lambda)

%ONEVSALL entrena varios clasificadores por regresión logística y devuelve
% el resultado en una matriz all_theta , donde l a fila iésima
% corresponde al clasificador de la etiqueta i!ésima
num_fixtures = columns(X);

initial_theta = zeros(num_fixtures, 1);

options=optimset('GradObj','on','MaxIter', 600);
for i=1:num_etiquetas
  #Con el y==c de la siguiente llamada, conseguimos hacer el OneVsAll
  #es decir ponemos a 1 la etiqueta que queramos entrenar, y a 0 todas las demás
  thetas(i, :) = _fmincg(@(t)(lrCostFunction(t, X,(y==i), lambda)),
initial_theta, options);
endfor;


endfunction
```

split_fold.m

```
function [X_train_fold Y_train_fold X_val_fold Y_val_fold] =
split_fold(X_train_val, Y_train_val,training_val_split_percentage,
k_folds_cross_val, k_fold)


number_features = columns(X_train_val);
number_instances = rows(X_train_val);
val_subtest_split_percentage = training_val_split_percentage /
k_folds_cross_val; #0.8 / 4 = 0.2
training_subtest_split_percentage = training_val_split_percentage -
val_subtest_split_percentage; #0.8 - 0.2 = 0.6

number_instances_training_subset =
round(number_instances*training_subtest_split_percentage);
number_instances_validation_subset =
round(number_instances*val_subtest_split_percentage);


rows_shuffled = randperm(rows(X_train_val));
X_train_val = X_train_val(rows_shuffled,:);
Y_train_val = Y_train_val(rows_shuffled,:);


X_train_fold = X_train_val(1:number_instances_training_subset, :);
X_val_fold = X_train_val(number_instances_training_subset+1:end, :);

Y_train_fold = Y_train_val(1:number_instances_training_subset, :);
Y_val_fold = Y_train_val(number_instances_training_subset+1:end, :);
```

```
number_classes = 13;
for i=1:number_classes
  i_class_train_val = find(i==Y_train_val);
  i_class_train_fold = find(i==Y_train_fold);
  i_class_val_fold = find(i==Y_val_fold);

    number_instances_i_class = rows(i_class_train_val);

    #Overfitting of classes impossible to detect without overfitting
  if (i==7 || i==8 || i == 11 || i ==12 || i ==13)
      X_train_fold(i_class_train_fold,:) = [];
      X_val_fold(i_class_val_fold,:) = [];
      Y_train_fold(i_class_train_fold,:) = [];
      Y_val_fold(i_class_val_fold,:) = [];

      X_train_fold = [X_train_fold; X_train_val(i_class_train_val, :)];
      X_val_fold = [X_val_fold; X_train_val(i_class_train_val, :)];

      Y_train_fold = [Y_train_fold; Y_train_val(i_class_train_val, :)];
      Y_val_fold = [Y_val_fold; Y_train_val(i_class_train_val, :)];
   endif;



   #Oversampling of classes with very few instances
  if (number_instances_i_class < 10)
  #I duplicate the values(15 times: 1(original insert)+14 duplications)
  #to achieve the oversampling on these classes
      duplicate_times = 14;

      if (i != 7 && i!=8 && i != 11 && i != 12 && i != 13)
        nuevas_filas_x_train_fold = X_train_fold(i_class_train_fold, :);
        nuevas_filas_y_train_fold = Y_train_fold(i_class_train_fold, :);
        X_train_fold(i_class_train_fold,:) = [];
        Y_train_fold(i_class_train_fold,:) = [];

       for (a=1:duplicate_times)
          X_train_fold = [X_train_fold; nuevas_filas_x_train_fold];
          Y_train_fold = [Y_train_fold; nuevas_filas_y_train_fold];
       endfor

     endif;
  endif

endfor;

endfunction
```

sigmoid_function.m

```
function sigmoid = sigmoid_function(z)


if isscalar(z)
  sigmoid = (1/(1+exp(-z)));
else isvector(z)
  sigmoid = (1./(1.+exp(-z)));
endif

endfunction;
```

mapFeature.m

```
function out = mapFeature(X1, X2)

% MAPFEATURE Feature mapping function to polynomial features
%
%   MAPFEATURE(X1, X2) maps the two input features
%   to quadratic features used in the regularization exercise.
%
%   Returns a new feature array with more features, comprising of
%   X1, X2, X1.^2, X2.^2, X1*X2, X1*X2.^2, etc..
%
%   Inputs X1, X2 must be the same size
%

degree = 6;
out = ones(size(X1(:,1)));
for i = 1:degree
    for j = 0:i
        out(:, end+1) = (X1.^(i-j)).*(X2.^j);
    end
end

end
```

lrCostFunction.m

```
function [J, grad] = lrCostFunction(theta, X, y, lambda)


m = length(y);
number_of_thetas = length(theta);

z = X*theta;
sigmoid = sigmoid_function(z);


J = ((1/m)*sum((-y.*log(sigmoid)) - ((1.-y).*log(1.-sigmoid)))) + ((lambda/
(2*m))*(theta'*theta));


grad = ((1/m)*((sigmoid.-y)'*X)).+((lambda/m).*theta'); #A cada gradiente se le
suma la carga de lambda
```

```
grad(1,1) = ((1/m)*((sigmoid(1,1).-y(1,1))'*X(1,1))); #Al grad(1,1) no se le
aplica lambda
#=========================================================================
grad = grad'; #Se hace la traspuesta porque lo pide así la función fmincg

endfunction
```

_fmincg.m

```
function [X, fX, i] = _fmincg(f, X, options, P1, P2, P3, P4, P5)

% Minimize a continuous differentialble multivariate function. Starting point
% is given by "X" (D by 1), and the function named in the string "f", must
% return a function value and a vector of partial derivatives. The Polack-
% Ribiere flavour of conjugate gradients is used to compute search directions,
% and a line search using quadratic and cubic polynomial approximations and the
% Wolfe-Powell stopping criteria is used together with the slope ratio method
% for guessing initial step sizes. Additionally a bunch of checks are made to
% make sure that exploration is taking place and that extrapolation will not
% be unboundedly large. The "length" gives the length of the run: if it is
% positive, it gives the maximum number of line searches, if negative its
% absolute gives the maximum allowed number of function evaluations. You can
% (optionally) give "length" a second component, which will indicate the
% reduction in function value to be expected in the first line-search (defaults
% to 1.0). The function returns when either its length is up, or if no further
% progress can be made (ie, we are at a minimum, or so close that due to
% numerical problems, we cannot get any closer). If the function terminates
% within a few iterations, it could be an indication that the function value
% and derivatives are not consistent (ie, there may be a bug in the
% implementation of your "f" function). The function returns the found
% solution "X", a vector of function values "fX" indicating the progress made
% and "i" the number of iterations (line searches or function evaluations,
% depending on the sign of "length") used.
%
% Usage: [X, fX, i] = fmincg(f, X, options, P1, P2, P3, P4, P5)
%
% See also: checkgrad
%
% Copyright (C) 2001 and 2002 by Carl Edward Rasmussen. Date 2002-02-13
%
%
% (C) Copyright 1999, 2000 & 2001, Carl Edward Rasmussen
%
% Permission is granted for anyone to copy, use, or modify these
% programs and accompanying documents for purposes of research or
% education, provided this copyright notice is retained, and note is
% made of any changes that have been made.
%
% These programs and documents are distributed without any warranty,
% express or implied.  As the programs were written for research
% purposes only, they have not been tested to the degree that would be
% advisable in any important application.  All use of these programs is
% entirely at the user's own risk.
%
% [ml-class] Changes Made:
% 1) Function name and argument specifications
% 2) Output display
%
```

```
% Read options
if exist('options', 'var') && ~isempty(options) && isfield(options, 'MaxIter')
    length = options.MaxIter;
else
    length = 100;
end


RHO = 0.01;                            % a bunch of constants for line searches
SIG = 0.5;       % RHO and SIG are the constants in the Wolfe-Powell conditions
INT = 0.1;    % don't reevaluate within 0.1 of the limit of the current bracket
EXT = 3.0;                    % extrapolate maximum 3 times the current bracket
MAX = 20;                         % max 20 function evaluations per line search
RATIO = 100;                                      % maximum allowed slope ratio

argstr = ['feval(f, X'];                      % compose string used to call
function
for i = 1:(nargin - 3)
  argstr = [argstr, ',P', int2str(i)];
end
argstr = [argstr, ')'];

if max(size(length)) == 2, red=length(2); length=length(1); else red=1; end
S=['Iteration '];

i = 0;                                            % zero the run length counter
ls_failed = 0;                             % no previous line search has failed
fX = [];
[f1 df1] = eval(argstr);                      % get function value and gradient
i = i + (length<0);                                            % count epochs?!
s = -df1;                                    % search direction is steepest
d1 = -s'*s;                                                % this is the slope
z1 = red/(1-d1);                             % initial step is red/(|s|+1)

while i < abs(length)                                      % while not finished
  i = i + (length>0);                                      % count iterations?!

  X0 = X; f0 = f1; df0 = df1;                   % make a copy of current values
  X = X + z1*s;                                            % begin line search
  [f2 df2] = eval(argstr);
  i = i + (length<0);                                        % count epochs?!
  d2 = df2'*s;
  f3 = f1; d3 = d1; z3 = -z1;             % initialize point 3 equal to point 1
  if length>0, M = MAX; else M = min(MAX, -length-i); end
  success = 0; limit = -1;                     % initialize quanteties
  while 1
    while ((f2 > f1+z1*RHO*d1) | (d2 > -SIG*d1)) & (M > 0)
      limit = z1;                                         % tighten the bracket
      if f2 > f1
        z2 = z3 - (0.5*d3*z3*z3)/(d3*z3+f2-f3);                 % quadratic fit
      else
        A = 6*(f2-f3)/z3+3*(d2+d3);                                 % cubic fit
        B = 3*(f3-f2)-z3*(d3+2*d2);
        z2 = (sqrt(B*B-A*d2*z3*z3)-B)/A;       % numerical error possible - ok!
      end
      if isnan(z2) | isinf(z2)
        z2 = z3/2;                  % if we had a numerical problem then bisect
      end
      z2 = max(min(z2, INT*z3),(1-INT)*z3);  % don't accept too close to limits
```

```
    z1 = z1 + z2;                                          % update the step
    X = X + z2*s;
    [f2 df2] = eval(argstr);
    M = M - 1; i = i + (length<0);                          % count epochs?!
    d2 = df2'*s;
    z3 = z3-z2;                    % z3 is now relative to the location of z2
  end
  if f2 > f1+z1*RHO*d1 | d2 > -SIG*d1
    break;                                                 % this is a failure
  elseif d2 > SIG*d1
    success = 1; break;                                                % success
  elseif M == 0
    break;                                                             % failure
  end
  A = 6*(f2-f3)/z3+3*(d2+d3);                      % make cubic extrapolation
  B = 3*(f3-f2)-z3*(d3+2*d2);
  z2 = -d2*z3*z3/(B+sqrt(B*B-A*d2*z3*z3));        % num. error possible - ok!
  if ~isreal(z2) | isnan(z2) | isinf(z2) | z2 < 0   % num prob or wrong sign?
    if limit < -0.5                                % if we have no upper limit
      z2 = z1 * (EXT-1);                 % the extrapolate the maximum amount
    else
      z2 = (limit-z1)/2;                                    % otherwise bisect
    end
  elseif (limit > -0.5) & (z2+z1 > limit)          % extraplation beyond max?
    z2 = (limit-z1)/2;                                                 % bisect
  elseif (limit < -0.5) & (z2+z1 > z1*EXT)       % extrapolation beyond limit
    z2 = z1*(EXT-1.0);                           % set to extrapolation limit
  elseif z2 < -z3*INT
    z2 = -z3*INT;
  elseif (limit > -0.5) & (z2 < (limit-z1)*(1.0-INT))   % too close to limit?
    z2 = (limit-z1)*(1.0-INT);
  end
  f3 = f2; d3 = d2; z3 = -z2;                  % set point 3 equal to point 2
  z1 = z1 + z2; X = X + z2*s;                      % update current estimates
  [f2 df2] = eval(argstr);
  M = M - 1; i = i + (length<0);                           % count epochs?!
  d2 = df2'*s;
end                                                      % end of line search

if success                                               % if line search succeeded
  f1 = f2; fX = [fX' f1]';
  fprintf('%s %4i | Cost: %4.6e\r', S, i, f1);
  s = (df2'*df2-df1'*df2)/(df1'*df1)*s - df2;      % Polack-Ribiere direction
  tmp = df1; df1 = df2; df2 = tmp;                         % swap derivatives
  d2 = df1'*s;
  if d2 > 0                                      % new slope must be negative
    s = -df1;                            % otherwise use steepest direction
    d2 = -s'*s;
  end
  z1 = z1 * min(RATIO, d1/(d2-realmin));          % slope ratio but max RATIO
  d1 = d2;
  ls_failed = 0;                                 % this line search did not fail
else
  X = X0; f1 = f0; df1 = df0;  % restore point from before failed line search
  if ls_failed | i > abs(length)          % line search failed twice in a row
    break;                             % or we ran out of time, so we give up
  end
  tmp = df1; df1 = df2; df2 = tmp;                         % swap derivatives
  s = -df1;                                                    % try steepest
  d1 = -s'*s;
```

```
    z1 = 1/(1-d1);
    ls_failed = 1;                                  % this line search failed
  end
  if exist('OCTAVE_VERSION')
    fflush(stdout);
  end
end
fprintf('\n');
```

(2) Redes neuronales

_red_neuronal_sinEntrenar_main.m

```
function [porcentaje_aciertos] = _red_neuronal_sinEntrenar_main(binary_classes)

more off;
#Selección de si se ha metido el parametro binary_classes(para que solo haya
sanos-no_sanos
#o en su defecto que sean las 13 clases originales
if (exist("binary_classes", "var"))
  if (binary_classes)
    num_etiquetas = 2;
    labels = [1, 2];
    load("dataset_preprocessed_binary.mat"); #Guarda los datos de entrada en X e
Y
  else
    num_etiquetas = 13;
    labels = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13];
    load("dataset_preprocessed.mat"); #Guarda los datos de entrada en X e Y
  endif;
else
  num_etiquetas = 13;
  labels = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13];
  load("dataset_preprocessed.mat"); #Guarda los datos de entrada en X e Y
endif
#===========Carga de datos=======================================
m = rows(X_train_val); #Número ejemplos de entrenamiento

#{
#======Display de unos cuantos ejemplos de entrada================
% Selecciona aleatoriamente 100 ejemplos
rand_indices = randperm(m);
sel = X(rand_indices(1:100),:);

displayData(sel);
#}

#===========Inicialización aleatoria de Thetas==================
#Es necesario que sea aleatorio porque si no al calcular el gradiente
#el error seria siempre 0 para todos los nodos de la red
number_fixtures = columns(X_train_val);
num_filas_l1 = number_fixtures;
num_filas_l2 = 25;
```

```
num_filas_l3 = num_etiquetas;
%seleciona aleatoriamente 100 ejemplos
Theta1 = _pesosAleatorios(num_filas_l2, num_filas_l1+1)';
Theta2 = _pesosAleatorios(num_filas_l3, num_filas_l2+1)';

#Juntamos Theta1 y Theta2 en la misma matriz (hacemos una matriz columna con
todas)
tam=rows(Theta1)*columns(Theta1);
Theta=reshape(Theta1,1,tam);
tam=rows(Theta2)*columns(Theta2);
Theta=[Theta reshape(Theta2,1,tam)];

num_entradas=number_fixtures;
num_ocultas=num_filas_l2;

#lambda = [0:0.15:6];
#num_nodos_capa_oculta = [num_ocultas:1:num_ocultas*2];
lambda = [0:0.15:1];
num_nodos_capa_oculta = [num_ocultas:6:num_ocultas*2];

aucs_mean = zeros(length(lambda), length(num_nodos_capa_oculta));
aucs_one_fold = zeros(k_folds_cross_val, 1);


aucs_mean_training = zeros(length(lambda), length(num_nodos_capa_oculta));
aucs_one_fold_training = zeros(k_folds_cross_val, 1);


for i=1:length(lambda)
  for j=1:columns(num_nodos_capa_oculta)
    j_num_ocultas = num_nodos_capa_oculta(j);
    for k_fold=1:k_folds_cross_val
      [X_train_fold Y_train_fold X_val_fold Y_val_fold] =
split_fold(X_train_val, Y_train_val,training_val_split_percentage,
k_folds_cross_val, k_fold);

      %seleciona aleatoriamente 100 ejemplos
      Theta1 = _pesosAleatorios(j_num_ocultas, num_filas_l1+1)';
      Theta2 = _pesosAleatorios(num_filas_l3, j_num_ocultas+1)';

      #Juntamos Theta1 y Theta2 en la misma matriz (hacemos una matriz columna
con todas)
      tam=rows(Theta1)*columns(Theta1);
      Theta=reshape(Theta1,1,tam);
      tam=rows(Theta2)*columns(Theta2);
      Theta=[Theta reshape(Theta2,1,tam)];


      options = optimset('GradObj','on','MaxIter',600);
      #checkNNGradients(lambda); #Se comparara nuestro gradiente obtenido(en
costeRN) con el numérico estimado
      #Se miniminza J en la función fmincg, ya que le pasamos thet
      params_rn = fmincg(@(thetas)(_costeRN(thetas, num_entradas, j_num_ocultas,
num_etiquetas, X_train_fold, Y_train_fold, lambda(i))),Theta' , options ) ;

      #Desenrollamos la matriz Theta que nos devulve fmincg en Theta1 y Theta2
      Theta1 = reshape(params_rn(1:j_num_ocultas * ( num_entradas + 1)),
j_num_ocultas, ( num_entradas + 1));
      Theta2 = reshape(params_rn((1 + (j_num_ocultas * ( num_entradas +
1))):end), num_etiquetas, (j_num_ocultas + 1));
```

```
      #PROPAGACIÓN HACIA DELANTE para computar el valor de h0(x(i)) para cada
ejemplo i====
      #==================a1=============================================
      a1 = X_val_fold;
      a0 = ones(rows(a1),1);
      a1 = [a0, a1]; #Se le añade el término a0=1 a la a
      #==================a2=============================================
      z2 = a1*Theta1';
      a2 = _sigmoid_function(z2);
      a0 = ones(rows(a2),1);
      a2 = [a0, a2]; #Se le añade el término a0=1 a la a
      #==================a3=============================================
      z3 = a2*Theta2';
      a3 = _sigmoid_function(z3);
      #================================================================
      h0 = a3; #La salida de la última capa es h0 (nuestra estimación)

      #=====Cálculo del porcentaje de casos que hemos acertado=====
      [maximo indicesMax] = max(h0');
      correctos = indicesMax'==Y_val_fold;
      correctos = sum(correctos);
      porcentaje_aciertos = (correctos/m)*100
      #================================================================
      #En indicesMax se guarda la posicion en la que está el valor máximo de
cada
      #ejemplo. Es decir es a nivel fila. Si el índice es 5 por ejemplo, querrá
      #decir que la posicion máxima era el 5 para ese ejemplo, por lo que
nuestra red neuronal
      #habrá estimado que el valor para esa imagen es un 5

      predicted_labels = indicesMax';
      auc = zeros(num_etiquetas, 1);
      for a=1:num_etiquetas
        a_class = find(a==Y_val_fold);
        number_instances_a_class = rows(a_class);

        ind_not_this_class = find(a!=Y_val_fold);
        Y_tmp = Y_val_fold;
        Y_tmp(ind_not_this_class) = -1;
        Y_tmp(a_class) = 1;


        predicted_a_class = find(a==predicted_labels);
        predicted_not_this_class = find(a!=predicted_labels);
        predicted_tmp = predicted_labels;
        predicted_tmp(predicted_a_class) = 1;
        predicted_tmp(predicted_not_this_class) = -1;


        auc(a) = auc_compute(Y_tmp, predicted_tmp, 1)
       endfor;

      auc_mean = mean(auc(~isnan(auc)));
      aucs_one_fold(k_fold) = auc_mean

      #===========Sobre training================
      #PROPAGACIÓN HACIA DELANTE para computar el valor de h0(x(i)) para cada
ejemplo i====
      #==================a1=============================================
```

```
      a1 = X_train_fold;
      a0 = ones(rows(a1),1);
      a1 = [a0, a1]; #Se le añade el término a0=1 a la a
      #==================a2=============================================
      z2 = a1*Theta1';
      a2 = _sigmoid_function(z2);
      a0 = ones(rows(a2),1);
      a2 = [a0, a2]; #Se le añade el término a0=1 a la a
      #====================a3===========================================
      z3 = a2*Theta2';
      a3 = _sigmoid_function(z3);
      #================================================================
      h0 = a3; #La salida de la última capa es h0 (nuestra estimación)

      #=====Cálculo del porcentaje de casos que hemos acertado=====
      [maximo indicesMax] = max(h0');
      correctos = indicesMax'==Y_train_fold;
      correctos = sum(correctos);
      porcentaje_aciertos = (correctos/m)*100
      #================================================================
      #En indicesMax se guarda la posicion en la que está el valor máximo de
cada
      #ejemplo. Es decir es a nivel fila. Si el índice es 5 por ejemplo, querrá
      #decir que la posicion máxima era el 5 para ese ejemplo, por lo que
nuestra red neuronal
      #habrá estimado que el valor para esa imagen es un 5

      predicted_labels = indicesMax';
      auc_training = zeros(num_etiquetas, 1);
      for a=1:num_etiquetas
        a_class = find(a==Y_train_fold);
        number_instances_a_class = rows(a_class);

        ind_not_this_class = find(a!=Y_train_fold);
        Y_tmp = Y_train_fold;
        Y_tmp(ind_not_this_class) = -1;
        Y_tmp(a_class) = 1;


        predicted_a_class = find(a==predicted_labels);
        predicted_not_this_class = find(a!=predicted_labels);
        predicted_tmp = predicted_labels;
        predicted_tmp(predicted_a_class) = 1;
        predicted_tmp(predicted_not_this_class) = -1;


        auc_training(a) = auc_compute(Y_tmp, predicted_tmp, 1)
       endfor;

      auc_mean_training = mean(auc_training(~isnan(auc_training)));
      aucs_one_fold_training(k_fold) = auc_mean_training

    endfor;
    aucs_mean(i, j) = mean(aucs_one_fold(~isnan(aucs_one_fold)));
    aucs_mean_training(i, j) =
mean(aucs_one_fold_training(~isnan(aucs_one_fold_training)));


      for zz=1:2000
        i
```

```
        j
      endfor;

  endfor;



endfor;

figure(5);
plot(lambda, mean(aucs_mean, 2), "b");
hold on;
plot(lambda, mean(aucs_mean_training, 2), "r");
xlabel("lambda", "fontsize", 11);
ylabel("AUC", "fontsize", 11);



figure(6);
plot(num_nodos_capa_oculta, mean(aucs_mean, 1), "b");
hold on;
plot(num_nodos_capa_oculta, mean(aucs_mean_training, 1), "r");
xlabel("num_nodos_capa_oculta", "fontsize", 11);
ylabel("AUC", "fontsize", 11);
#========Cálculo de qué etiqueta hemos estimado que es cada
ejemplo==============
num_casos = rows(X_test);

[best_model_i ind_best_model_i] = max(aucs_mean);
ind_best_model_i = ind_best_model_i(1);
[best_model ind_best_model_j] = max(best_model_i);

best_lambda = lambda(ind_best_model_i)
best_num_nodos_capa_oculta = num_nodos_capa_oculta(ind_best_model_j)


%seleciona aleatoriamente 100 ejemplos
Theta1 = _pesosAleatorios(best_num_nodos_capa_oculta, num_filas_l1+1)';
Theta2 = _pesosAleatorios(num_filas_l3, best_num_nodos_capa_oculta+1)';

#Juntamos Theta1 y Theta2 en la misma matriz (hacemos una matriz columna con
todas)
tam=rows(Theta1)*columns(Theta1);
Theta=reshape(Theta1,1,tam);
tam=rows(Theta2)*columns(Theta2);
Theta=[Theta reshape(Theta2,1,tam)];

options = optimset('GradObj','on','MaxIter',600);
#checkNNGradients(lambda); #Se comparara nuestro gradiente obtenido(en costeRN)
con el numérico estimado
#Se miniminza J en la función fmincg, ya que le pasamos thet
if (exist("binary_classes", "var"))
  if (binary_classes)
    params_rn = fmincg(@(thetas)(_costeRN(thetas, num_entradas,
best_num_nodos_capa_oculta, num_etiquetas, X_train_val, Y_train_val,
best_lambda)),Theta' , options ) ;
  else
    params_rn = fmincg(@(thetas)(_costeRN(thetas, num_entradas,
best_num_nodos_capa_oculta, num_etiquetas, X_train_val_oversampled,
Y_train_val_oversampled, best_lambda)),Theta' , options ) ;
```

```
  endif;
else
  params_rn = fmincg(@(thetas)(_costeRN(thetas, num_entradas,
best_num_nodos_capa_oculta, num_etiquetas, X_train_val_oversampled,
Y_train_val_oversampled, best_lambda)),Theta' , options ) ;
endif


#Desenrollamos la matriz Theta que nos devulve fmincg en Theta1 y Theta2
Theta1 = reshape(params_rn(1:best_num_nodos_capa_oculta * ( num_entradas + 1)),
best_num_nodos_capa_oculta, ( num_entradas + 1));
Theta2 = reshape(params_rn((1 + (best_num_nodos_capa_oculta * ( num_entradas +
1))):end), num_etiquetas, (best_num_nodos_capa_oculta + 1));

#PROPAGACIÓN HACIA DELANTE para computar el valor de h0(x(i)) para cada ejemplo
i====
#==================a1=============================================
a1 = X_test;
a0 = ones(rows(a1),1);
a1 = [a0, a1]; #Se le añade el término a0=1 a la a
#==================a2=============================================
z2 = a1*Theta1';
a2 = _sigmoid_function(z2);
a0 = ones(rows(a2),1);
a2 = [a0, a2]; #Se le añade el término a0=1 a la a
#====================a3=============================================
z3 = a2*Theta2';
a3 = _sigmoid_function(z3);
#=================================================================
h0 = a3; #La salida de la última capa es h0 (nuestra estimación)

#=====Cálculo del porcentaje de casos que hemos acertado=====
[maximo indicesMax] = max(h0');
correctos = indicesMax'==Y_test;
correctos = sum(correctos);
porcentaje_aciertos = (correctos/m)*100
#=================================================================
#En indicesMax se guarda la posicion en la que está el valor máximo de cada
#ejemplo. Es decir es a nivel fila. Si el índice es 5 por ejemplo, querrá
#decir que la posicion máxima era el 5 para ese ejemplo, por lo que nuestra red
neuronal
#habrá estimado que el valor para esa imagen es un 5

predicted_labels = indicesMax';

predicted_labels_neural = predicted_labels;
Y_test_neural = Y_test;
model_neural_theta1 = Theta1;
model_neural_theta2 = Theta2;
save -mat7-binary 'dataset_model_neural.mat', 'predicted_labels_neural',
'Y_test_neural', 'model_neural_theta1', 'model_neural_theta2';


best_lambda
best_num_nodos_capa_oculta

[best_model_aucs_mean ind_best_model_i] = max(aucs_mean)
[best_model_aucs_mean_training ind_best_model_it] = max(aucs_mean_training)

endfunction
```

_sigmoid_function.m

```
function sigmoid = _sigmoid_function(z)


if isscalar(z)
  sigmoid = (1/(1+exp(-z)));
else isvector(z)
  sigmoid = (1./(1.+exp(-z)));
endif

endfunction;
```

_sigmoid_function_derivative.m

```
function sigmoid = _sigmoid_function_derivative(z)


if isscalar(z)
      sigmoid = _sigmoid_function(z)*(1-_sigmoid_function(z));
elseif isvector(z)
      sigmoid = _sigmoid_function(z).*(1.-_sigmoid_function(z));
elseif ismatrix(z)
      sigmoid = _sigmoid_function(z).*(1.-_sigmoid_function(z));
endif

endfunction;
```

_pesosAleatorios.m

```
function W = _pesosAleatorios(l_out, l_in)

  INIT_EPSILON = sqrt(6)/sqrt(l_out+l_in);

  W = rand(l_out, l_in)*(2*INIT_EPSILON) - INIT_EPSILON;
endfunction
```

debugInitializeWeights.m

```
function W = debugInitializeWeights(fan_out, fan_in)

%DEBUGINITIALIZEWEIGHTS Initialize the weights of a layer with fan_in
%incoming connections and fan_out outgoing connections using a fixed
%strategy, this will help you later in debugging
%   W = DEBUGINITIALIZEWEIGHTS(fan_in, fan_out) initializes the weights
%   of a layer with fan_in incoming connections and fan_out outgoing
%   connections using a fix set of values
%
%   Note that W should be set to a matrix of size(1 + fan_in, fan_out) as
%   the first row of W handles the "bias" terms
%
```

```
% Set W to zeros
W = zeros(fan_out, 1 + fan_in);

% Initialize W using "sin", this ensures that W is always of the same
% values and will be useful for debugging
W = reshape(sin(1:numel(W)), size(W)) / 10;

% =========================================================================

end
```

_costeRN.m

```
function [J grad] = _costeRN(params_rn, num_entradas, num_ocultas,
num_etiquetas, X, y, lambda)

  #{
  Obtención de las matrices Theta (se desenrollan), estaban enrolladas en 1
matriz columna
  que hacía la labor de un array, ya que no se pueden juntar las 2 matrices
según vienen,
  porque las dimensiones no coinciden, por eso se juntaron todas en 1 columna
  #}
  Theta1 = reshape(params_rn(1:num_ocultas * ( num_entradas + 1)), num_ocultas,
( num_entradas + 1));
  Theta2 = reshape(params_rn((1 + (num_ocultas * ( num_entradas + 1))):end),
num_etiquetas, (num_ocultas + 1));

  m = rows(X);
  Y=zeros(m,num_etiquetas);

  etiquetas = eye(num_etiquetas); #etiquetas es de (10x10) Cada fila tiene un 1
en la columna=fila
  for i=1:m
    Y(i,:) = etiquetas(y(i),:);  #Ejemplo de valores de y [1..10], 5=[0 0 0 0 1
0 0 0 0 0]
  endfor
  #PROPAGACIÓN HACIA DELANTE para computar el valor de h0(x(i)) para cada
ejemplo i====
  #==================a1=============================================
  a1 = X;
  a0 = ones(rows(a1),1);
  a1 = [a0, a1]; #Se le añade el término a0=1 a la a
  #==================a2=============================================
  z2 = a1*Theta1';
  a2 = _sigmoid_function(z2);
  a0 = ones(rows(a2),1);
  a2 = [a0, a2]; #Se le añade el término a0=1 a la a
  #==================a3=============================================
  z3 = a2*Theta2';
  a3 = _sigmoid_function(z3);
  #=================================================================
  h0 = a3; #La salida de la última capa es h0 (nuestra estimación)

  #================Función de coste
J(0)=======================================
  J = (1/m)*sum(sum((-Y).*log(h0) - (1-Y).*log(1-h0), 2));
```

```
  #El último 2 significa sum(matriz, dimension=2), es decir se suma por
columnas(el resultado es columna por tanto)
  regularizacion_thetas = (lambda/(2*m))*(sum(sum(Theta1(:, 2:end).^2, 2)) +
sum(sum(Theta2(:,2:end).^2, 2)));
  J = J + regularizacion_thetas;

  #PROPAGACIÓN HACIA ATRÁS (backpropagation) para computar el valor de h0(x(i))
para cada ejemplo i====
  #================Cálculo del
gradiente=======================================
  #Cálculo del error(sigma) en cada nivel
  sigma3 = a3 - Y;
  sigma2 = (sigma3*Theta2 .* _sigmoid_function_derivative([ones(rows(z2),1),
z2]))(:,2:end);
  #[ones(rows(z2),1), z2] añade el término bias(el que lleva un 1) a cada caso
de entrada

  #El (:,2:end) es para eliminar el término bias, es decir el que lleva un 1 al
principio de cada nivel
  #Se tiene que añadir porque Theta2 lo tenía, así se pueden multiplicar las
matrices, aunque
  #luego se tenga que quitar porque el error de la variable 0 no nos
sirve(porque siempre tiene valor
  #1 ese nodo y no se acumularía bien?

  #Acumulacón(delta) del error en cada nivel
  delta_1= sigma2'*a1;
  delta_2= sigma3'*a2;

  #Regularización de Theta:
  grad_1_regularizado = zeros(size(Theta1)); #Se reserva el espacio de memoria
  grad_2_regularizado = zeros(size(Theta2)); #Se reserva el espacio de memoria
  grad_1_regularizado = delta_1./m + (lambda/m)*[zeros(rows(Theta1), 1)
Theta1(:, 2:end)];
  grad_2_regularizado = delta_2./m + (lambda/m)*[zeros(rows(Theta2), 1)
Theta2(:, 2:end)];

  grad = [grad_1_regularizado(:) ; grad_2_regularizado(:)];



endfunction;
```

(3) SVM (Support vector machines)

_svm_train_ecg.m

```
function [porcentaje c sigma] = _svm_train_ecg(binary_classes)

more off;

#Selección de si se ha metido el parametro binary_classes(para que solo haya
sanos-no_sanos
#o en su defecto que sean las 13 clases originales
if (exist("binary_classes", "var"))
  if (binary_classes)
    num_classes = 1; #We just need 1 loop(to compare class 1 vs class 2)
    labels = [1, 2];
    load("dataset_preprocessed_binary.mat"); #Guarda los datos de entrada en X e
Y
  else
    num_classes = 13;
    labels = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13];
    load("dataset_preprocessed.mat"); #Guarda los datos de entrada en X e Y
  endif;
else
  num_classes = 13;
  labels = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13];
  load("dataset_preprocessed.mat"); #Guarda los datos de entrada en X e Y
endif

n=rows(Y_train_val);


k_fold_cross_validation = 4;




#c= [0.1:30:140];
#sigma = [0.001:0.001:0.003];
c= [0.1:2:105];
sigma = [0.0001:0.0001:0.002];

n=rows(Y_train_val);
c_done = zeros(length(c), 1);
sigma_done = zeros(length(sigma), 1);


aucs_mean = zeros(length(c), length(sigma));
aucs_one_fold = zeros(k_folds_cross_val, 1);

aucs_mean_training = zeros(length(c), length(sigma));
aucs_one_fold_training = zeros(k_folds_cross_val, 1);

for i=1:length(c)
  for j=1:length(sigma)
    for k_fold=1:k_folds_cross_val
      [X_train_fold Y_train_fold X_val_fold Y_val_fold] =
split_fold(X_train_val, Y_train_val,training_val_split_percentage,
k_folds_cross_val, k_fold);
```

```
      options_lib_svm = sprintf('-c %d -g %d -q', c(i), sigma(j));
      model = svmtrain(Y_train_fold, X_train_fold, options_lib_svm);
      [predicted_labels, accuracy, prob_estimates] = svmpredict(Y_val_fold,
X_val_fold, model);


      for z=1:2000 #Contador de vueltas (Asi lo veo cuanto queda por la salida
estandar)
        i
        j
      endfor

      auc = zeros(num_classes, 1);
      for a=1:num_classes
        a_class = find(a==Y_val_fold);
        number_instances_a_class = rows(a_class);

        ind_not_this_class = find(a!=Y_val_fold);
        Y_tmp = Y_val_fold;
        Y_tmp(ind_not_this_class) = -1;
        Y_tmp(a_class) = 1;


        predicted_a_class = find(a==predicted_labels);
        predicted_not_this_class = find(a!=predicted_labels);
        predicted_tmp = predicted_labels;
        predicted_tmp(predicted_a_class) = 1;
        predicted_tmp(predicted_not_this_class) = -1;

        auc(a) = auc_compute(Y_tmp, predicted_tmp, 1)

        endfor;

      auc_mean = mean(auc(~isnan(auc)));
      aucs_one_fold(k_fold) = auc_mean


      #========Sobre training===========
      [predicted_labels, accuracy, prob_estimates] = svmpredict(Y_train_fold,
X_train_fold, model);


      auc_training = zeros(num_classes, 1);
      for a=1:num_classes
        a_class = find(a==Y_train_fold);
        number_instances_a_class = rows(a_class);

        ind_not_this_class = find(a!=Y_train_fold);
        Y_tmp = Y_train_fold;
        Y_tmp(ind_not_this_class) = -1;
        Y_tmp(a_class) = 1;


        predicted_a_class = find(a==predicted_labels);
        predicted_not_this_class = find(a!=predicted_labels);
        predicted_tmp = predicted_labels;
        predicted_tmp(predicted_a_class) = 1;
        predicted_tmp(predicted_not_this_class) = -1;

        auc_training(a) = auc_compute(Y_tmp, predicted_tmp, 1)
```

```
        endfor;

      auc_mean_training = mean(auc_training(~isnan(auc_training)));
      aucs_one_fold_training(k_fold) = auc_mean_training

    endfor;
    aucs_mean(i, j) = mean(aucs_one_fold(~isnan(aucs_one_fold)))
    aucs_mean_training(i, j) =
mean(aucs_one_fold_training(~isnan(aucs_one_fold_training)))

    #sigma_done(i) =sigma(j);
  endfor
  #c_done(i) = c(i);
endfor

figure(2);
plot(c, mean(aucs_mean, 2), "b");
hold on;
plot(c,mean(aucs_mean_training, 2), "r");
xlabel("c", "fontsize", 11);
ylabel("AUC", "fontsize", 11);

figure(3);
plot(sigma, mean(aucs_mean, 1), "b");
hold on;
plot(sigma, mean(aucs_mean_training, 1), "r");
xlabel("sigma", "fontsize", 11);
ylabel("AUC", "fontsize", 11);


[best_model_i ind_best_model_i] = max(aucs_mean);
ind_best_model_i = ind_best_model_i(1);
[best_model ind_best_model_j] = max(best_model_i);

best_c = c(ind_best_model_i);
best_sigma = sigma(ind_best_model_j);

options_lib_svm = sprintf('-c %d -g %d -q', best_c, best_sigma);

if (exist("binary_classes", "var"))
  if (binary_classes)
      model = svmtrain(Y_train_val, X_train_val, options_lib_svm);
  else
    model = svmtrain(Y_train_val_oversampled, X_train_val_oversampled,
options_lib_svm);
  endif;

else
    model = svmtrain(Y_train_val_oversampled, X_train_val_oversampled,
options_lib_svm);
endif;

[predicted_labels, accuracy_total, prob_estimates] = svmpredict(Y_test, X_test,
model)

 #========PLOT===========
    figure(1);
    number_features = columns(X_test);
    #PCA (Principal component analysis):
```

```
    k = 2; #Number of PC(principal components) we want. We want 2 so we can plot
the data
    sigma = (1/number_features)*(X_test'*X_test);
    [U, S, V] = svd(sigma);
    Ureduce = U(:, 1:k);
    z = X_test*Ureduce;

    _plot_pca(z, Y_test);

    hold on;

    #----Plot boundary line-----------
    % calculate w and b
    w = model.SVs' * model.sv_coef;
    b = -model.rho;

    if model.Label(1) == 1
      w = -w;
      b = -b;
    end
    disp(w);
    disp(b);

    % plot the boundary line
    x = [min(z(:,1)):.01:max(z(:,1))];
    y = (-b - w(1)*x ) / w(2);
    hold on;
    plot(x,y)
    #Fuente del plot de la boundary line: http://www.alivelearn.net/wp-
content/uploads/2009/10/cuixu_test_svm1.m
   #-----------------------------------
   #==========================


predicted_labels_svm = predicted_labels;
Y_test_svm = Y_test;
model_svm = model;
#To use the model: [predicted_labels, accuracy_total, prob_estimates] =
svmpredict(Y_test, X_test, model)
save -mat7-binary 'dataset_model_svm.mat', 'predicted_labels_svm', 'Y_test_svm',
'model_svm';


best_c
best_sigma

[best_model_aucs_i ind_best_model_i1] = max(aucs_mean);
[best_model_aucs_mean ind_best_model_j] = max(best_model_aucs_i)

[best_model_aucs_t ind_best_model_i2] = max(aucs_mean_training);
[best_model_aucs_training ind_best_model_j] = max(best_model_aucs_t)
endfunction

#"one-against-one" multi-class method:
#================================================================================#
#{
LIBSVM implements "one-against-one" multi-class method, so there are k(k-1)/2
binary models,
where k is the number of classes.
```

```
We can consider two ways to conduct parameter selection.

For any two classes of data, a parameter selection procedure is conducted.
Finally, each decision function has its own optimal parameters.
The same parameters are used for all k(k-1)/2 binary classification problems.
We select parameters that achieve the highest overall performance.

Each has its own advantages. A single parameter set may not be uniformly good
for
all k(k-1)/2 decision functions. However, as the overall accuracy is the final
consideration,
one parameter set for one decision function may lead to over-fitting. In the
paper

#}
```

visualizeBoundary.m

```
function visualizeBoundary(X, y, model, varargin)

%VISUALIZEBOUNDARY plots a non-linear decision boundary learned by the SVM
%   VISUALIZEBOUNDARYLINEAR(X, y, model) plots a non-linear decision
%   boundary learned by the SVM and overlays the data on it

% Plot the training data on top of the boundary
plotData(X, y)

% Make classification predictions over a grid of values
x1plot = linspace(min(X(:,1)), max(X(:,1)), 100)';
x2plot = linspace(min(X(:,2)), max(X(:,2)), 100)';
[X1, X2] = meshgrid(x1plot, x2plot);
vals = zeros(size(X1));
for i = 1:size(X1, 2)
   this_X = [X1(:, i), X2(:, i)];
   vals(:, i) = svmPredict(model, this_X);
end

% Plot the SVM boundary
hold on
%contour(X1, X2, vals, [0 0], 'Color', 'k');
contour(X1, X2, vals, 'linecolor', 'blue');
hold off;

end
```

4) Post-procesamiento (Resultados)

_2_postprocess.m

```
function _2_postprocess()


load("dataset_model_reg.mat");
load("dataset_model_neural.mat");
load("dataset_model_svm.mat");


#POST-PROCESSING================================

#==============REGRESION=====================================
display("")
display("==================")
display("==========REGRESION========")
display("==================")
[accuracy_total, auc,true_positives, false_positives, true_negatives,
false_negatives,true_positives_mean,false_positives_mean,true_negatives_mean,fal
se_negatives_mean, precision, recall,precision_mean, recall_mean, accuracy] =
compute_metrics(predicted_labels_reg, Y_test_reg);
#==============REDES NEURONALES=====================================
display("")
display("==================")
display("==========REDES NEURONALES========")
display("==================")
[accuracy_total, auc,true_positives, false_positives, true_negatives,
false_negatives,true_positives_mean,false_positives_mean,true_negatives_mean,fal
se_negatives_mean, precision, recall,precision_mean, recall_mean, accuracy] =
compute_metrics(predicted_labels_neural, Y_test_neural);


#==============SVM=====================================
display("")
display("==================")
display("==========SVM========")
display("==================")

[accuracy_total, auc,true_positives, false_positives, true_negatives,
false_negatives,true_positives_mean,false_positives_mean,true_negatives_mean,fal
se_negatives_mean, precision, recall,precision_mean, recall_mean, accuracy] =
compute_metrics(predicted_labels_svm, Y_test_svm);



endfunction
```

auc_compute.m

```
%fnorm Calculates the aucscore.

% usage auc = aucscore(y, ypred, plot)
% y is the actual values and ypred the predicted ones.
% if plot is true a graph with auc will be ploted. default is 0
% positive labels are 1 and negative -1.
function auc = aucscore(y, y_pred, plot)
if (~exist('plot', 'var') || isempty(plot))
plot = 0;
end

if (size(y, 2) ~= 1)
    y = y';
    y_pred = y_pred';
end

[~,ind] = sort(y_pred,'descend');
roc_y = y(ind);
stack_x = cumsum(roc_y == -1)/sum(roc_y == -1);
stack_y = cumsum(roc_y == 1)/sum(roc_y == 1);
auc = sum((stack_x(2:length(roc_y),1)-stack_x(1:length(roc_y)-
1,1)).*stack_y(2:length(roc_y),1));


#{
if (plot)
    plot(stack_x, stack_y);
    xlabel('False Positive Rate');
    ylabel('True Positive Rate');
    title(['ROC curve of (AUC = ' num2str(auc) ' )']);
end
#}


end
```

compute_metrics.m

```
function [accuracy_total, auc,true_positives, false_positives, true_negatives,
false_negatives,true_positives_mean,false_positives_mean,true_negatives_mean,fal
se_negatives_mean, precision, recall,precision_mean, recall_mean, accuracy] =
compute_metrics(predicted_labels, Y_test)


number_classes = rows(unique(Y_test))


m = rows(Y_test);
correctos = predicted_labels ==Y_test;
correctos = sum(correctos);
accuracy_total = (correctos/m)*100;


auc = zeros(number_classes, 1);
true_positives = zeros(number_classes, 1);
```

```
false_positives = zeros(number_classes, 1);
true_negatives = zeros(number_classes, 1);
false_negatives = zeros(number_classes, 1);
accuracy = zeros(number_classes, 1);
precision = zeros(number_classes, 1);
recall = zeros(number_classes, 1);



for a=1:number_classes
  a_class = find(a==Y_test);
  number_instances_a_class = rows(a_class);

  ind_not_this_class = find(a!=Y_test);
  Y_tmp = Y_test;
  Y_tmp(ind_not_this_class) = -1;
  Y_tmp(a_class) = 1;


  predicted_a_class = find(a==predicted_labels);
  predicted_not_this_class = find(a!=predicted_labels);
  predicted_tmp = predicted_labels;
  predicted_tmp(predicted_a_class) = 1;
  predicted_tmp(predicted_not_this_class) = -1;


  auc(a) = auc_compute(Y_tmp, predicted_tmp, 1);

  true_positives(a) = sum((predicted_tmp==1)&(Y_tmp == 1));
  false_positives(a) = sum((predicted_tmp==1)&(Y_tmp != 1));

  true_negatives(a) = sum((predicted_tmp!=1)&(Y_tmp != 1));
  false_negatives(a) = sum((predicted_tmp!=1)&(Y_tmp == 1));

  precision(a) = true_positives(a) / (true_positives(a) + false_positives(a));
  recall(a) = true_positives(a) / (true_positives(a) + false_negatives(a));

  num_casos = rows(Y_tmp);
  correctos = predicted_tmp == Y_tmp;
  correctos = sum(correctos);
  accuracy(a) = (correctos/num_casos)*100;
endfor;

#METRICAS:=============================================================================
auc
true_positives
false_positives
true_negatives
false_negatives
precision
recall

auc_mean = mean(auc(~isnan(auc)))

true_positives_mean = mean(true_positives(~isnan(true_positives)))
false_positives_mean = mean(false_positives(~isnan(false_positives)))
true_negatives_mean = mean(true_negatives(~isnan(true_negatives)))
false_negatives_mean = mean(false_negatives(~isnan(false_negatives)))

precision_mean = mean(precision(~isnan(precision)))
```

```
recall_mean = mean(recall(~isnan(recall)))

accuracy_total


endfunction
```

plot_auc.m

```
function plot_auc()

load('dataset_model_neural.mat');

[X,Y,T,AUC] = perfcurve(Y_test_neural, predicted_labels_neural,2); %
perfcurve(labels,predicted_label,positive_class);

AUC
plot(X,Y)
xlabel('False positive rate')
ylabel('True positive rate')
title('ROC for Classification by Logistic Regression')
```

8. Referencias

Campus virtual de la asignatura aprendizaje automático y big data.

https://es.coursera.org/learn/machine-learning

https://www.youtube.com/playlist?list=PL4th1XikMZ-yTF9H2njcXsNleLVzjEJhv

http://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel

http://www.academia.edu/11235385/Methods_used_in_Machine_Learning_ECG_Signal_Analysis_for_Arrhythmia_Classification

http://machinelearningmastery.com/assessing-comparing-classifier-performance-roc-curves-2/

https://florianhartl.com/thoughts-on-machine-learning-dealing-with-skewed-classes.html

http://www.sciencedirect.com/science/article/pii/S2212017313004933

http://www.iaras.org/iaras/filedownloads/ijbb/2016/021-0001.pdf