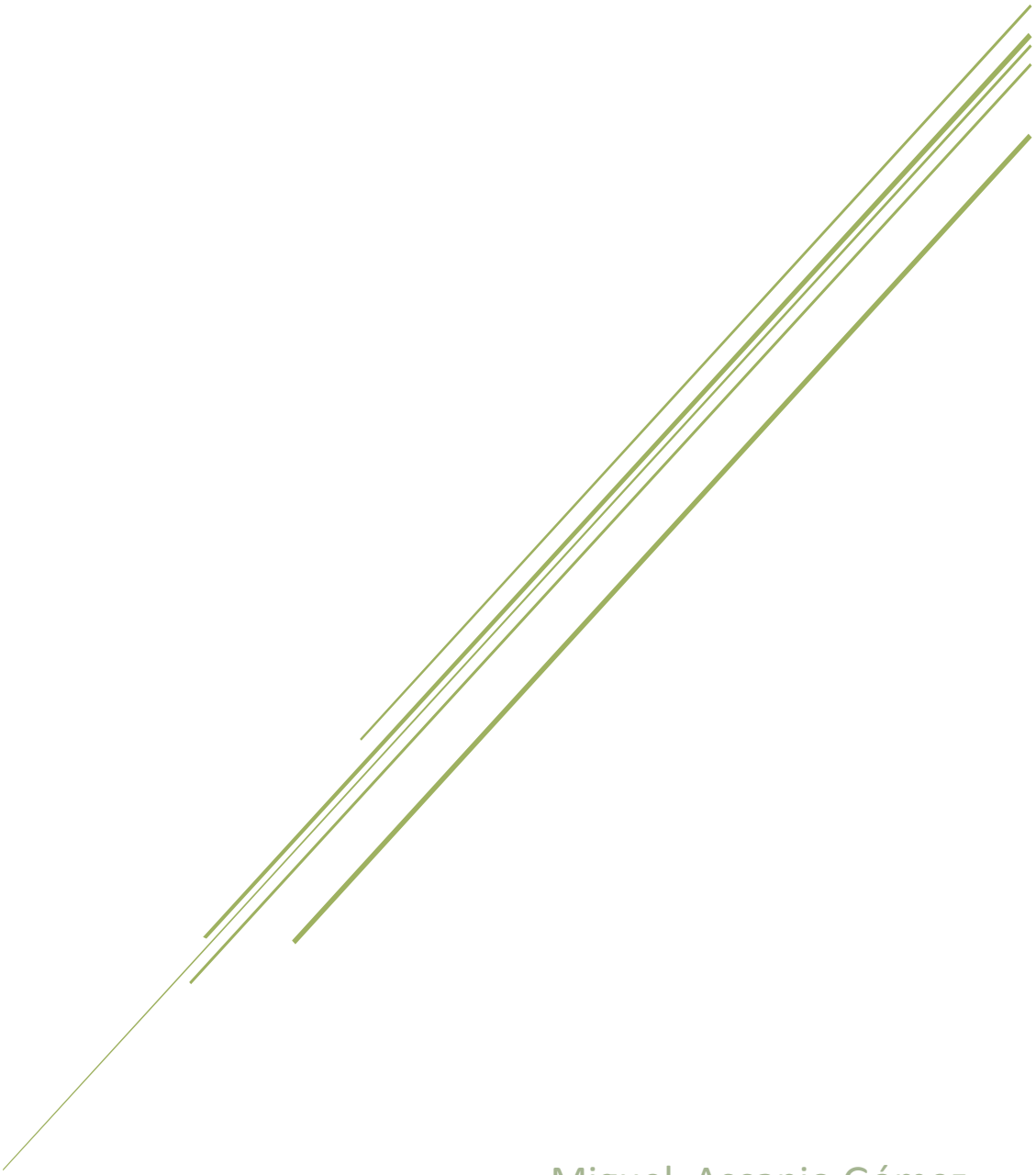
# APRENDIZAJE AUTOMÁTICO

## Clasificación de páginas web



Miguel  Ascanio Gómez

Universidad Complutense de Madrid

# INTRODUCCIÓN

El problema a resolver es la clasificación de páginas web en temas, a partir de las palabras contenidas en la web.

## MÉTODOS A UTILIZAR

Sobre ese problema aplicaremos algunas de las técnicas estudiadas en clase: regresión logística multiclase, redes neuronales, SVM o aprendizaje Bayesiano.

## DESCRIPCIÓN DEL DATASET

Estos métodos requieren de un conjunto de ejemplos de entrenamiento sobre los que entrenarse, los cuales han sido obtenidos de un dataset ya hecho: http://mldata.org/repository/data/viewslug/dmoz-web-directory-topics/

En este dataset se presentan 1329 páginas webs, parseadas como índice inverso de palabras de un diccionario (hablaré de esto a continuación), clasificadas en 5 temas (arte, juegos, infantil, compras, sociedad)

El dataset se ha construido a partir de http://www.dmoz.org/, un directorio con millones de páginas web clasificadas por temas.

Para construir un dataset, como el anterior, a partir de un directorio como dmoz, lo más lógico sería:

- Obtener los enlaces de webs de los temas deseados. Dmoz provee archivos en diferentes formatos para este fin.
- Procesar las páginas webs. Para esta aproximación, en la que sólo tenemos en cuenta las apariciones de las palabras, podemos eliminar todo lo que no sea texto de la web: tomando el código HTML de la misma, eliminamos todo lo que no sea texto plano (eliminando también las etiquetas HTML por supuesto). Para ello hay herramientas como http://jsoup.org/ que facilitan el trabajo.
- A continuación, se puede aplicar un algoritmo de *stemming* (http://tartarus.org/martin/PorterStemmer/) para quedarnos sólo con la raíz de las palabras obtenidas en el paso anterior.
- Una vez obtenidas las raíces de las palabras, creamos el diccionario con todas las palabras, y marcamos en cada web el número de apariciones de cada token del diccionario.

Aunque sea un esquema bastante lógico, no parece que se haya usado al pie de la letra en el dataset seleccionado, pues hay palabras en el diccionario que no aparecen en ninguna de las webs.

La estructura presentada en el dataset es una matriz con los 1329 ejemplos de entrenamiento, cada uno con 10630 campos que indican el número de apariciones de cada palabra en dicho ejemplo; y otro vector de 1329 posiciones guarda la etiqueta correspondiente a cada ejemplo, esto es, el tema de la web (5-arte, 6-juegos, 7-infantil, 8-compras, 9-sociedad).

# PRIMERA APROXIMACIÓN: REDES NEURONALES

Para comenzar, entreno una red neuronal probando varios valores para el número de ejemplos de entrenamiento:

```
clear;
% Párametros del test %
fixed_test_m = 329;
step = 25;
% Carga estática de datos %

load dmoz-web-directory-topics.mat;
X = full(data)';
y = label';
y = y.-4;
clear data label mldata_descr_ordering;

num_entradas = size(X)(2);
num_ocultas = 150;
num_etiquetas = 5;
numItersMax = 75;
lambda = 0;
m = size(X)(1);

Xtest = X(end-fixed_test_m+1:end,:);
ytest = y(end-fixed_test_m+1:end);
X_test_unos =  [ones(fixed_test_m, 1), Xtest];

ThetaIni1 = pesosAleatorios(num_entradas,num_ocultas);
ThetaIni2 = pesosAleatorios(num_ocultas,num_etiquetas);
thetaIni = [ThetaIni1(:); ThetaIni2(:)];

opciones = optimset ('Gradobj', 'on', 'MaxIter', numItersMax);

i=1;
for mtrain = step:step:m-fixed_test_m
    printf("Iteración i: %d, mtrain: %d\n", i,mtrain);
    tic()
    dataSave(i).numEntren = mtrain;

    Xtrain = X(1:mtrain,:);
    ytrain = y(1:mtrain);

    X_train_unos = [ones(mtrain, 1), Xtrain];

    func = @(p)(costeRN(p, num_entradas, num_ocultas, num_etiquetas,
X_train_unos, ytrain, lambda));

    [thetaProc, cost] = fmincg (func, thetaIni, opciones);
    [dataSave(i).costeEntren void] = costeRN(thetaProc, num_entradas,
num_ocultas, num_etiquetas, X_train_unos, ytrain, lambda);
    [dataSave(i).costeVal void] = costeRN(thetaProc, num_entradas,
num_ocultas, num_etiquetas, X_test_unos, ytest, lambda);

    ThetaProc1 = reshape(thetaProc(1:num_ocultas * (num_entradas +
1)), ...
```

```
        num_ocultas, (num_entradas + 1));
    ThetaProc2 = reshape(thetaProc((1 + (num_ocultas * (num_entradas +
1))):end), ...
        num_etiquetas, (num_ocultas + 1));

    [a1, a2, z2, yPred] = red(ThetaProc1, ThetaProc2, X_train_unos);
    dataSave(i).porcentajeEntren = porcentajeAciertos(ytrain, yPred);
    dataSave(i).porcentajeEntren;
    [a1, a2, z2, yPred] = red(ThetaProc1, ThetaProc2, X_test_unos);
    dataSave(i).porcentajeTest = porcentajeAciertos(ytest, yPred);
    dataSave(i).porcentajeTest;
    i++;
    toc()
endfor
```

Ejecutando el código anterior, se obtienen los datos para las siguientes gráficas:

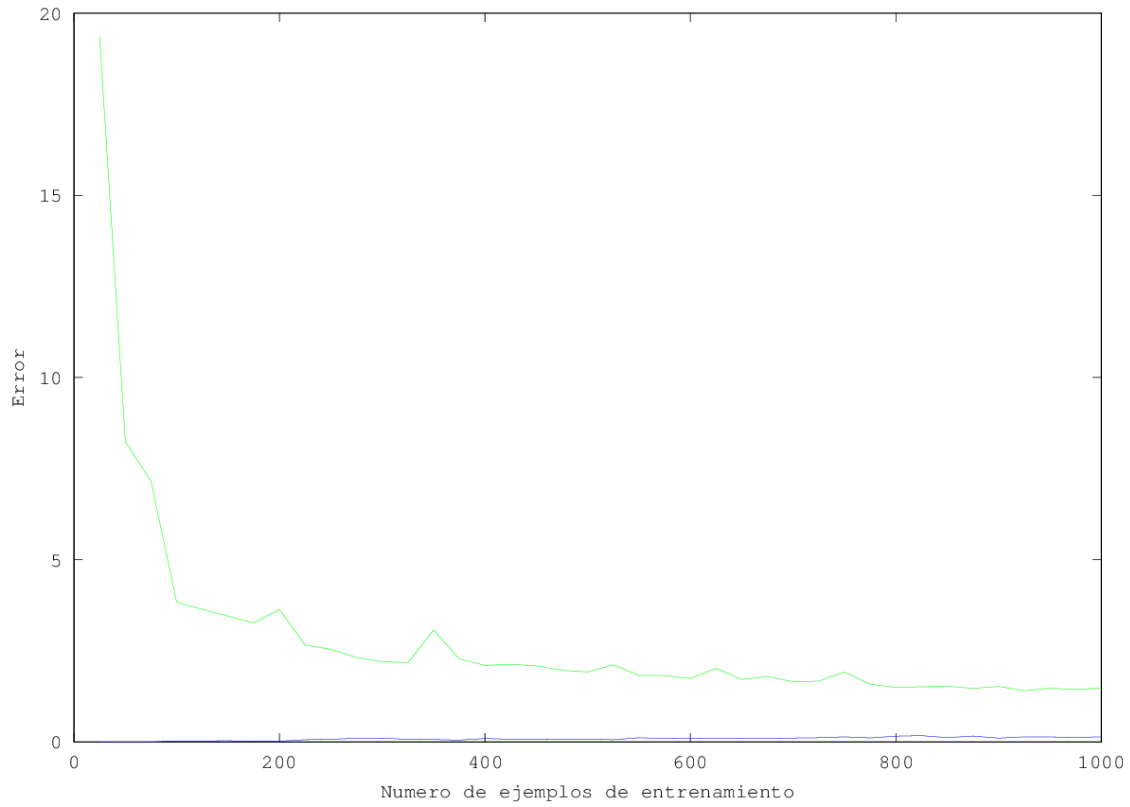



Fig. 1

Se observa que a partir de 800 ejemplos de entrenamiento el error sobre los ejemplos de validación (verde) casi no disminuye, mientras que el error sobre los ejemplos de entrenamiento (azul) es prácticamente nulo.

En la siguiente gráfica se observan comportamientos análogos sobre el porcentaje de aciertos:

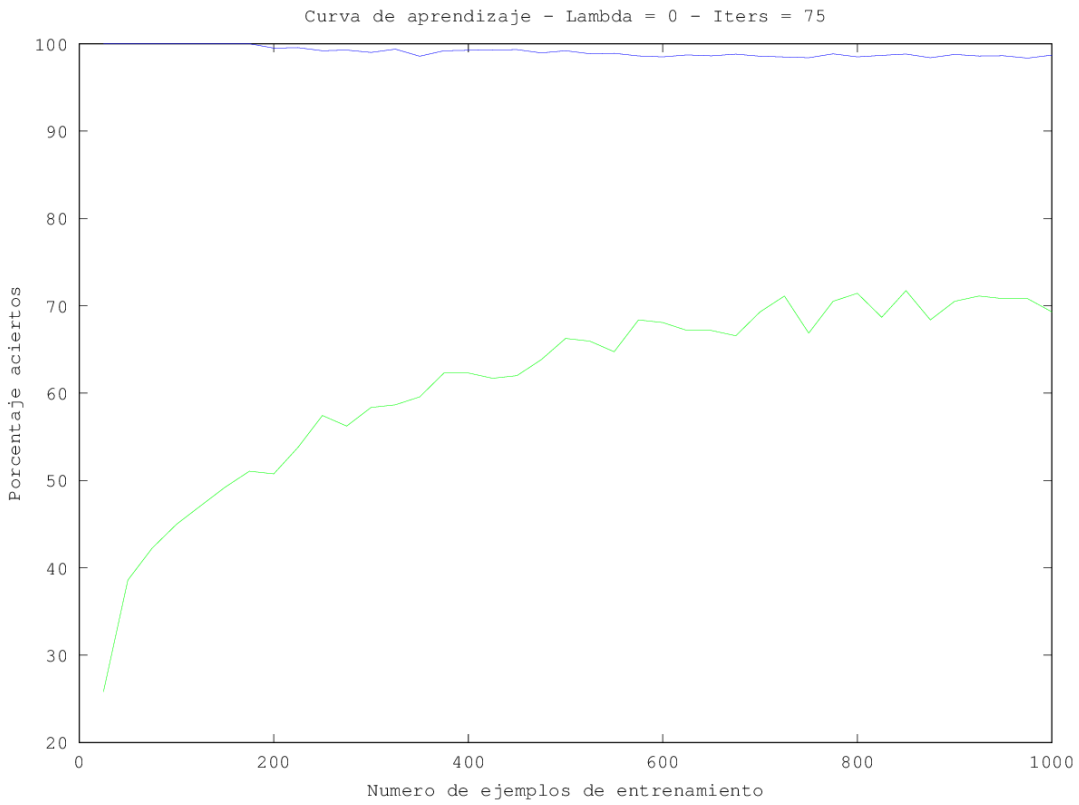



**Fig. 2**

Con un código parecido al anterior, solo que fijando el número de ejemplos de entrenamiento a 850, con 75 iteraciones, se pueden obtener gráficas para ver el comportamiento de la red neuronal cambiando el valor del parámetro lambda:

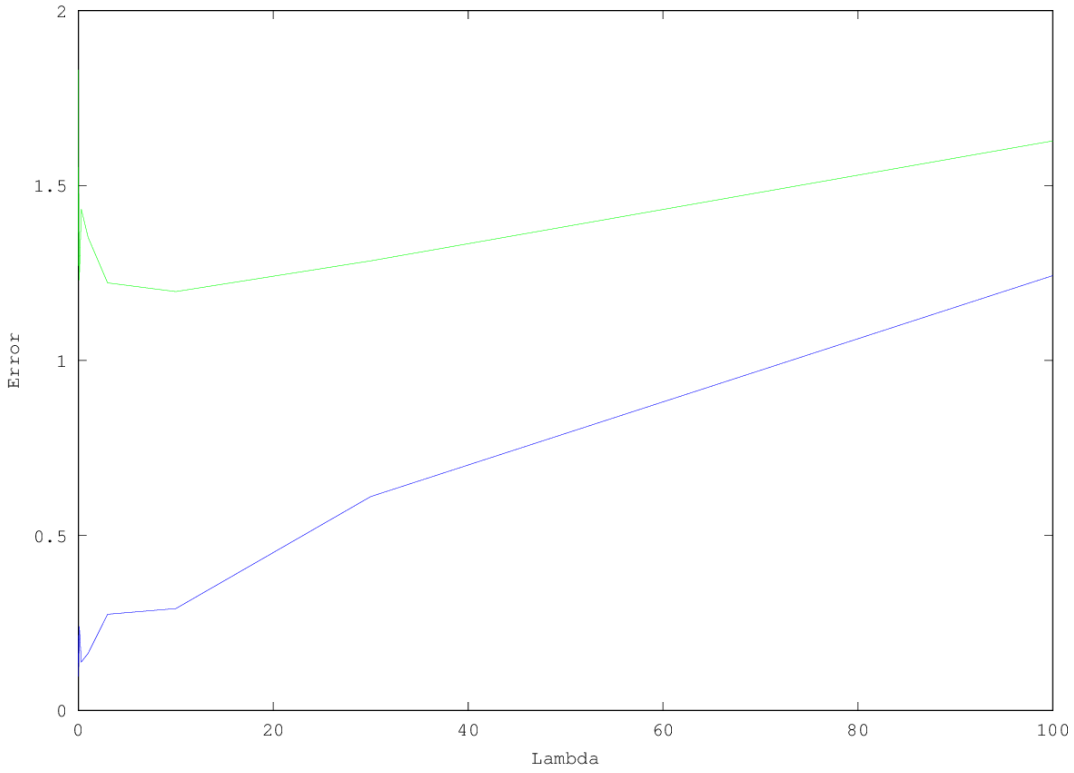



Fig. 3

Se observa que se obtiene el error mínimo para los ejemplos de entrenamiento con un valor de lambda = 10. Al seguir aumentando lambda, aumenta el error sobre los ejemplos de entrenamiento, al igual que sobre los de validación.

Probando ahora sobre el número de iteraciones:

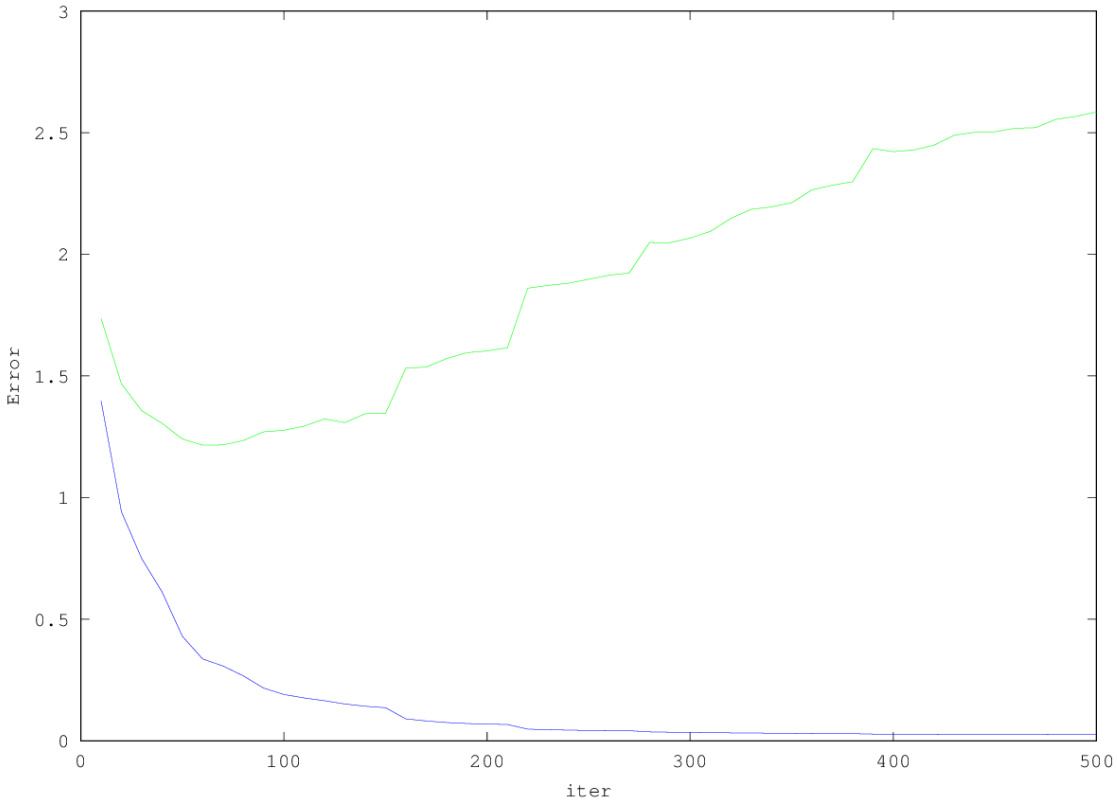



Fig. 4

Se observa que con pocas iteraciones (unas 75) se obtiene el mejor resultado sobre los ejemplos de validación. Aumentar el número solo hace que se sobreajuste más sobre los ejemplos de entrenamiento, empeorando aquellos que nunca ha visto.

## CONCLUSIONES INICIALES

A la vista de las gráficas obtenidas, parece bastante coherente afirmar que existe un sobreajuste sobre los ejemplos de entrenamiento: en las primeras gráficas se observa que se clasifican perfectamente los ejemplos de entrenamiento, lo que viene a ser sobreajuste.

Realmente esto no es nada extraño al tener tantos atributos y muy pocos ejemplos de entrenamiento.

### ABORDANDO EL SOBREAJUSTE

Posibles soluciones al problema son:

- Obtener más ejemplos de entrenamiento, esto no es posible ya que simplemente no los hay en este dataset. Además, en la gráfica de número de ejemplos de entrenamiento no parece que vaya a ayudar en exceso.
- Aumentar el valor de Lambda, se observa en **Fig.3** que esto no ayuda.
- Eliminando atributos

## ELIMINAR ATRIBUTOS

Con el análisis del sobreajuste, parece que lo único que podría funcionar es eliminar atributos de los ejemplos, la cuestión es cómo hacerlo.

Una primera idea es quitar las palabras que menos aparezcan en todo el conjunto de documentos, básicamente contamos las apariciones de cada palabra en el conjunto, y eliminamos aquellas palabras que aparezcan menos de *n* veces; la idea es que si *n* es un número bajo, quitaríamos aquellas palabras que aparecen tan poco que no aportan nada. Aplicando esto, para distintos valores de *n,* se observa una mejora de alrededor de un 2% de aciertos en el caso mejor. Dicho valor de *n* es bastante bajo, en torno a 10, quitando unas 2000 palabras (lo que es un número considerable), no así el porcentaje de mejora. Además se observa que para valores de *n* grandes el porcentaje de aciertos cae drásticamente.

También se puede optar por la estrategia opuesta, que es eliminar las palabras que más aparezcan, pues es posible que estas palabras aparezcan en muchas webs de distintos temas, no aportando nada.

Otra opción es una mezcla de las dos anteriores, sin embargo, podríamos estar quitando palabras que sean muy significativas para diferenciar entre los diferentes temas.

## IMPORTANCIA DE LAS PALABRAS

Parece bastante sensato pensar que algunas palabras son más importantes para clasificar los documentos en las que aparecen que otras: el, este, bajo… son palabras que aparecen con mucha frecuencia y en todos los tipos de documentos, por lo que no aportan mucho a la hora de clasificar.

Estas palabras que aparecen en muchos documentos se denominan en Inglés stop-words, y sería interesante que se tuviera en cuenta que son poco importantes o, lo que es lo mismo, tener en cuenta la importancia en general de todas las palabras.

## ALGORITMO TF-IDF

El algoritmo TF-IDF, del inglés *term frequency – inverse document frequency,* expresa cuán relevante es una palabra para un documento en una colección.

Palabras que aparezcan en muchos documentos tendrán un valor cercano a 0, si aparecen en todos el valor será exactamente 0. Por el contrario, un valor alto se dará cuando una palabra aparezca en pocos documentos de la colección, pero muchas veces en dichos documentos.

Esto se consigue con las siguientes fórmulas:

- La fórmula del TF-IDF en sí, del término t en el documento d, sobre el conjunto de documentos D, que se define como:

$$Tfidf(t,d,D) = tf(t,d) * idf(t,D)$$

- Siendo tf la frecuencia del término t en el documento d, expresada en escala logarítmica (para tener en cuenta, por ejemplo, que una palabra que aparezca 600 veces no es 600 veces más importante que una que aparezca una vez):

$$tf(t,d) = 1 + \log\big(f(t,d)\big)\,,\ 0\ si\ t\ no\ aparece$$

- E idf la frecuencia inversa, del término t en el conjunto D:

$$idf(t,D) = \log\left(\frac{N}{nº\ apariciones\ de\ t\ en\ D}\right)$$

La implementación correspondiente en Octave sería:

```
function tfidf = tfidf(term, document, Collection)
        tfidf = tfLog(term, document).* idf(term, Collection);
endfunction


function tf = tfLog(term, document)
    a = document(:,term)(:);
    idx = find(a > 0);

    tf = zeros(size(a));
    tf(idx) = 1 + log(a(idx));

    tf = reshape(tf, size(document(:,term)));

endfunction


function idf = idf(term, Collection)
    N = rows(Collection);

    for i=1:size(Collection)(2)
        df(i) = (length(find(Collection(:,i)!= 0)));
    endfor

    idf = log(N ./ df);

endfunction
```

## RESULTADO DE APLICAR TF-IDF

Si aplicamos el algoritmo sobre los ejemplos, tanto de entrenamiento como de validación, cambiando las frecuencias absolutas de cada palabra en cada documento por su valor TF-IDF, obtenemos lo siguiente (nuevamente utilizando un código similar al primero):

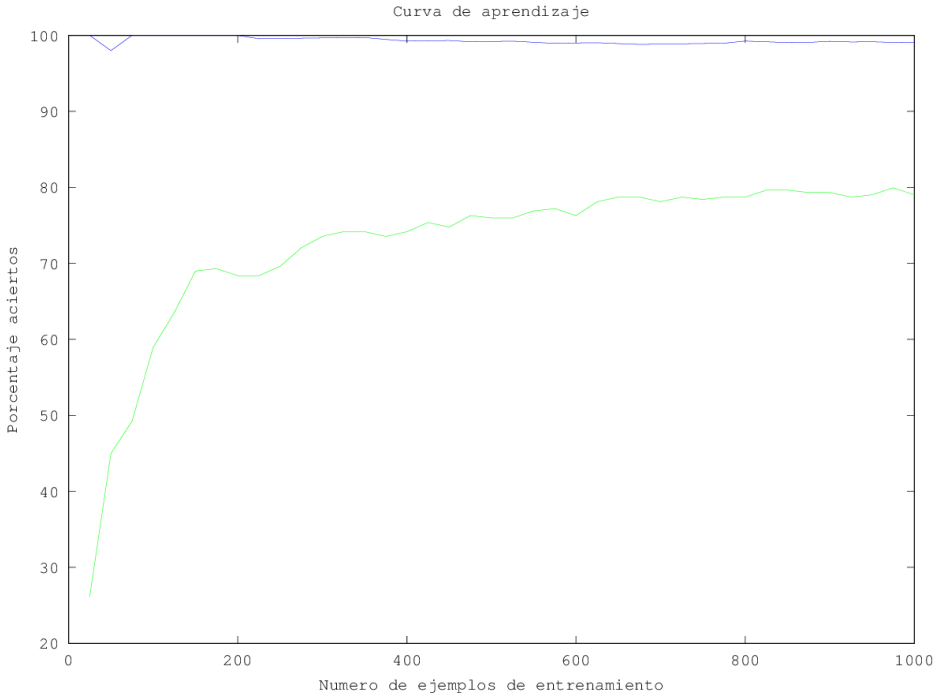



Fig. 5

Se observa, comparando esta Fig.5 con Fig.2 (la misma gráfica pero sin TF-IDF) que la curva es igual, sólo que con TF-IDF se consigue una mejora de un 10%. Haciendo más pruebas el porcentaje de aciertos llega a un 83%, desde luego una mejora considerable del porcentaje de aciertos.

## UN PAR DE PRUEBAS MÁS

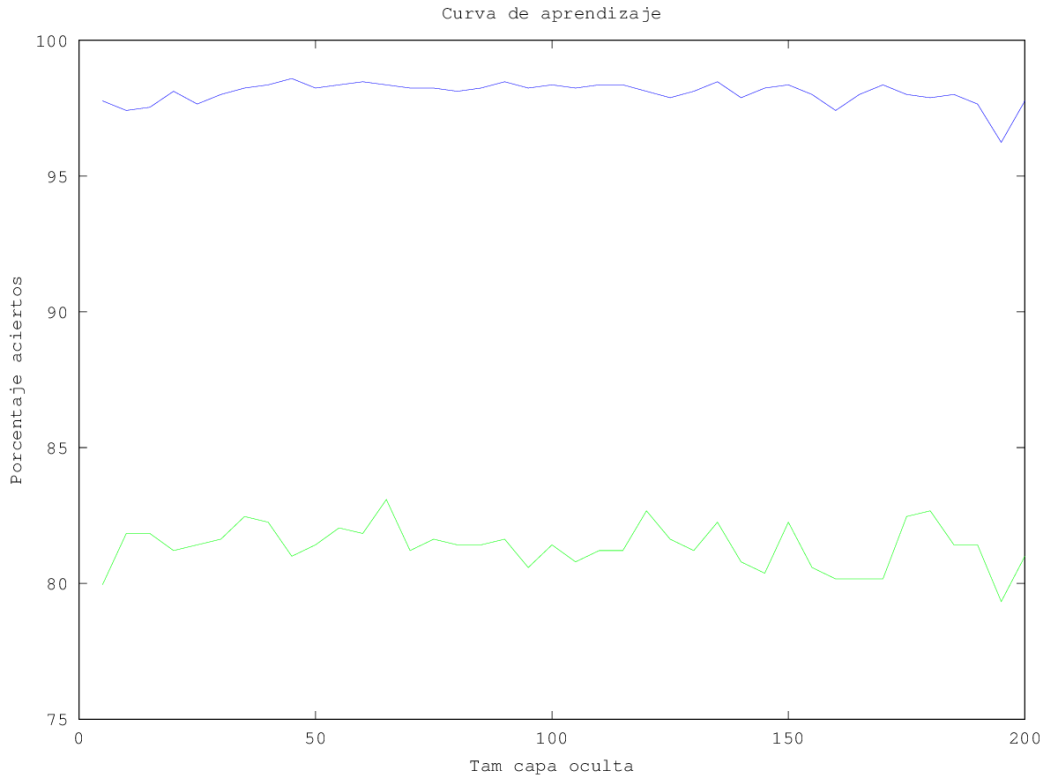



Fig. 6

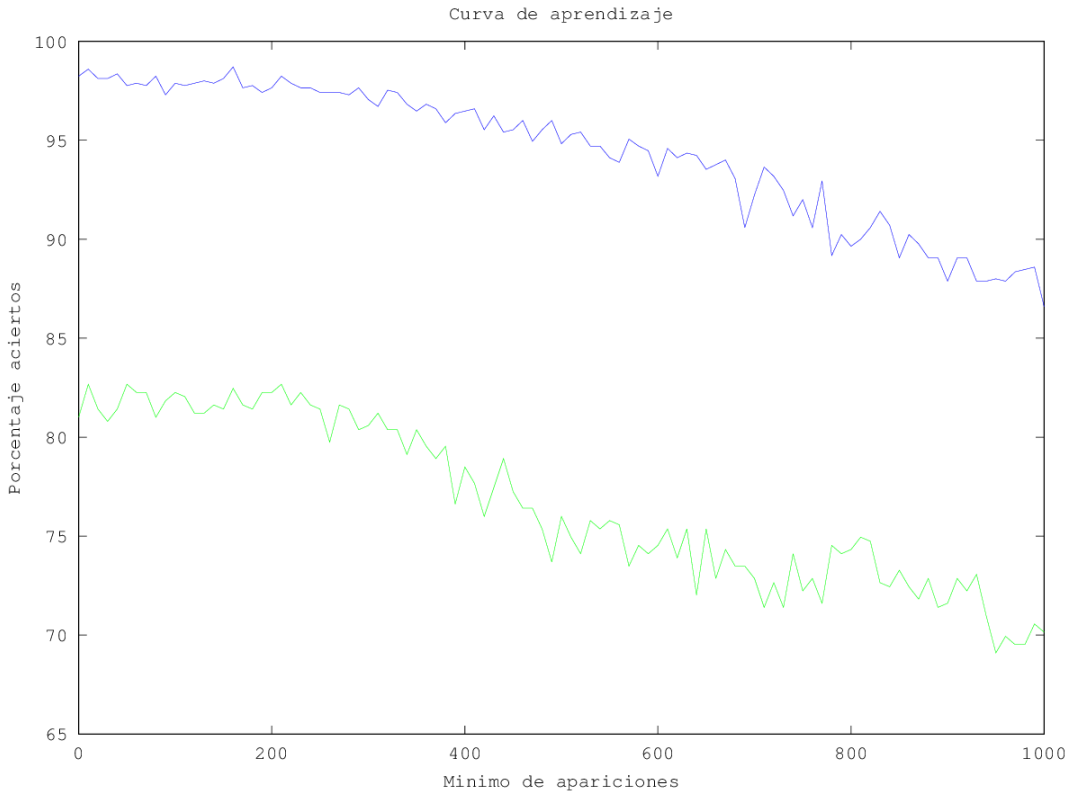



Fig. 7

En Fig.6 se observa el resultado de cambiar el tamaño de la capa oculta utilizada. Se observa que con un número muy bajo, ~20, se obtienen los mismos resultados que con capas más grandes. Para 2000 el resultado era el mismo pero tardaba mucho más.

En Fig.7 se observa el efecto de eliminar palabras que aparezcan menos de x veces, y luego aplicar el algoritmo TF-IDF con las que nos han quedado.

# REGRESIÓN LOGÍSTICA

Tras probar con redes neuronales, el siguiente paso fue la regresión logística multiclase. El código para la prueba de apariciones de las palabras es el siguiente:

```
% Párametros del test %
mtrain = 850;

usingFminunc = false;

lambda = 50;
iteraciones = 50;

num_etiquetas = 5;
i = 1;
# Palabras que aparezcan menos de min veces se eliminaran

for min=0:10:1000

    # Palabras que aparezcan menos de min veces se eliminaran
    % Carga estática de datos  con tfidf %
    load dmoz-web-directory-topics.mat;
    [X y] = cargaDatostfidf(data, label, num_etiquetas, min);
    clear data label mldata_descr_ordering;

    num_entradas = size(X)(2);
    m = size(X)(1);
    n = size(X)(2);

    Xtrain = X(1:mtrain,:);
    ytrain = y(1:mtrain);
    #X_train_unos = [ones(mtrain, 1), Xtrain];

    Xval = X(mtrain+1:end,:);
    yval = y(mtrain+1:end);
    #X_val_unos =  [ones(size(Xval)(1), 1), Xval];

    Thetas = oneVsAll(Xtrain, ytrain, num_etiquetas, lambda,
iteraciones, usingFminunc);
    printf("Con %d ejemplos de entrenamiento, %d ejemplos de
validacion, %d palabras\n\t%% aciertos sobre validacion: %.4f\n",
mtrain, length(y), n, porcentajeAciertosLog(Thetas , Xval , yval ));

    dataSave(i).palabras = n;
    dataSave(i).min = min;
    dataSave(i).porcentajeVal = porcentajeAciertosLog(Thetas, Xval,
yval);
    dataSave(i).porcentajeEnt = porcentajeAciertosLog(Thetas, Xtrain,
ytrain);
    i++;

endfor
```

A partir de los datos guardados con ese código, y con códigos similares para la prueba de lambda y número de ejemplos de entrenamiento, se obtienen las siguientes gráficas:

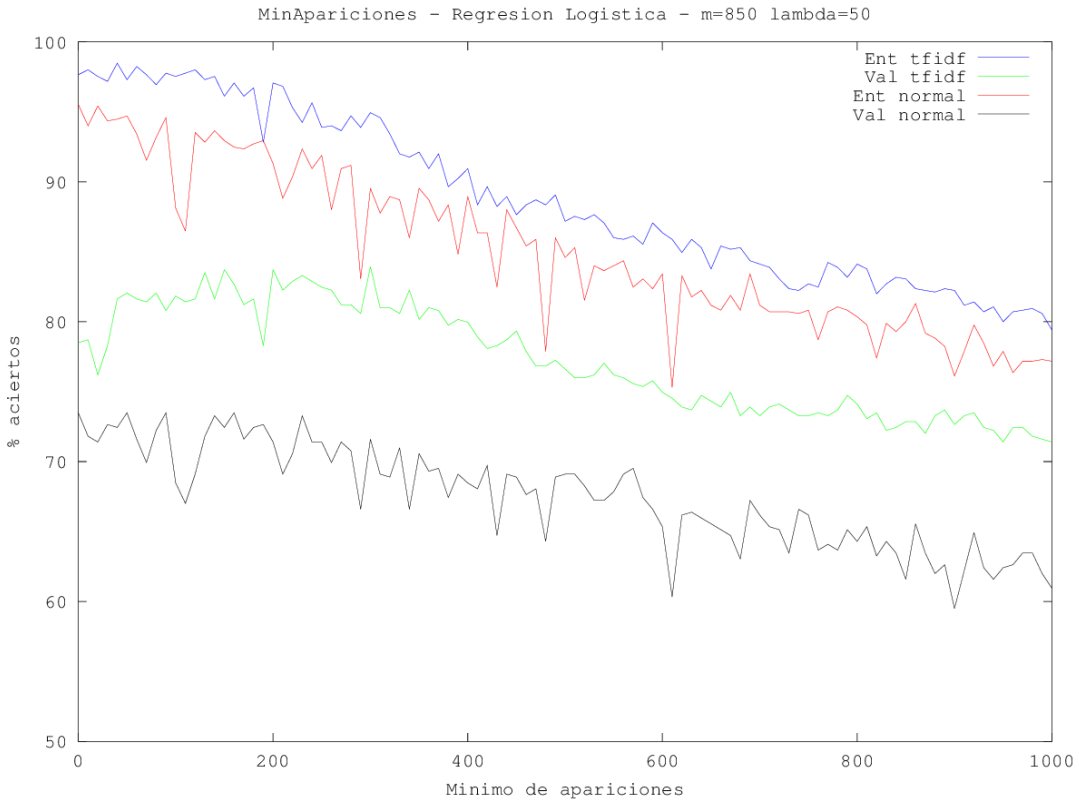

MinApariciones - Regresion Logistica - m=850 lambda=50
Ent tfidf
Val tfidf
Ent normal
Val normal
% aciertos
Minimo de apariciones
100
90
80
70
60
50
0
200
400
600
800
1000

Fig. 8

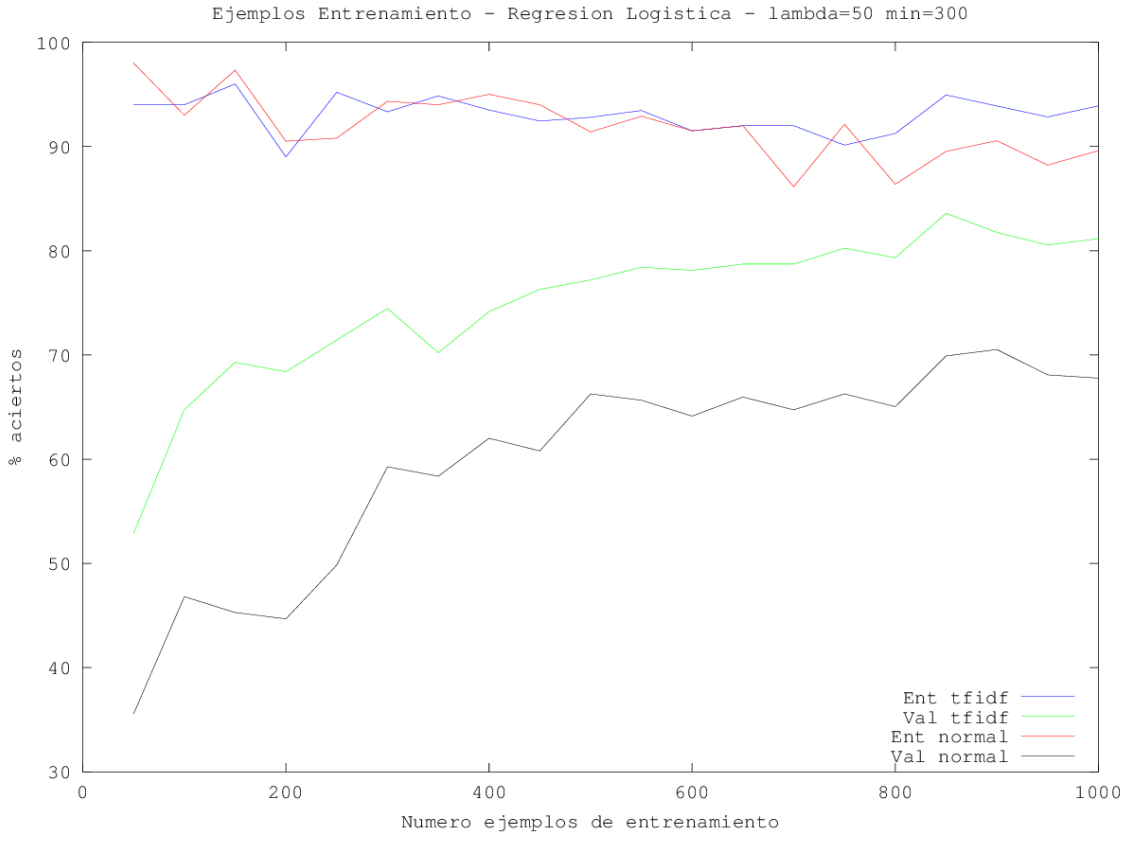

Ejemplos Entrenamiento - Regresion Logistica - lambda=50 min=300
Ent tfidf
Val tfidf
Ent normal
Val normal
% aciertos
Numero ejemplos de entrenamiento
100
90
80
70
60
50
40
30
0
200
400
600
800
1000

Fig. 9

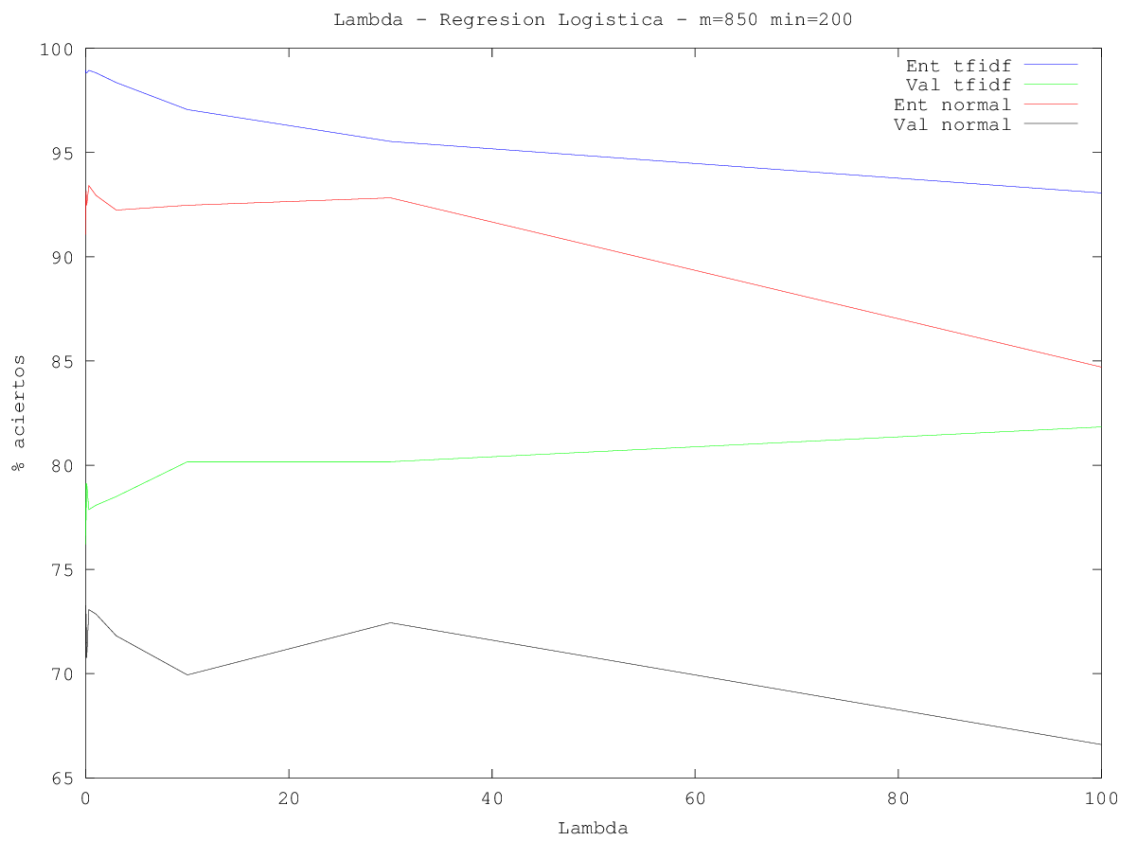



Fig. 10

En general, se deduces de las gráficas que el comportamiento, con o sin TF-IDF aplicado es el mismo, salvo porque TF-IDF consigue porcentajes de acierto bastante mayores.

En Fig.8 se observa que se puede eliminar las palabras que aparecen menos de 350 veces sin perder rendimiento, a partir de ahí el porcentaje de aciertos empieza a caer.

En cuanto al número de ejemplos en Fig.9 se obtiene un pico en aproximadamente 850, básicamente cuantos más se usen parece mejorar.

Sobre el parámetro Lambda, a partir de 15, en TF-IDF, los resultados no varían hasta llegar a 100, a partir de ahí empieza a decaer.

## RESUMEN

En general, el rendimiento en cuanto a porcentaje de aciertos en el caso mejor es muy parejo al de las redes neuronales, así como el tiempo de entrenamiento (cuando usamos redes pequeñas, que obtienen el mismo rendimiento que las grandes) es el mismo, casi instantáneo.

# SUPPORT VECTOR MACHINES

Al tratar de ejecutar el código utilizado en clase para entrenar support vector machines, por alguna razón (posible bug) la función de entrenamiento devuelve todo estructuras vacías o ceros, algo obviamente inviable.

Ante esta situación, decidí utilizar LIBSVM, una librería para múltiples lenguajes de programación, entre ellos octave/matlab, con multitud de utilidades para SVM.

Utilizando las funciones análogas a SVMTrain y SVMPredict, con kernel lineal (por las características del dataset parece más sensato que un kernel gaussiano) queda el siguiente código:

```
% Párametros del test %
mtrain = 850;

c = 0.01;
num_etiquetas = 5;

# Palabras que aparezcan menos de min veces se eliminaran


i = 1;
for min = 0:10:1000
% Carga estática de datos  con tfidf %
load dmoz-web-directory-topics.mat;
[X y] = cargaDatostfidf(data, label, num_etiquetas, min);
clear data label mldata_descr_ordering;

num_entradas = size(X)(2);
m = size(X)(1);
n = size(X)(2);

X_train = X(1:mtrain,:);
y_train = y(1:mtrain);

X_val = X(mtrain+1:end,:);
y_val = y(mtrain+1:end);

    opts = sprintf("-t 0 -q -c %f", c);
    model = svmtrain(double(y_train), double(X_train), opts);
    [void, trainPorA, void] = svmpredict(double(y_train),
double(X_train), model, '-q');
    [void, valPorA, void] = svmpredict(double(y_val), double(X_val),
model, '');
fflush(1)
    trainPor(i) = trainPorA(1);
    valPor(i) = valPorA(1);
    i++;
endfor
```

Este código sirve para la prueba del número mínimo de apariciones con TF-IDF, las otras pruebas son muy similares.  Las gráficas obtenidas son las siguientes:

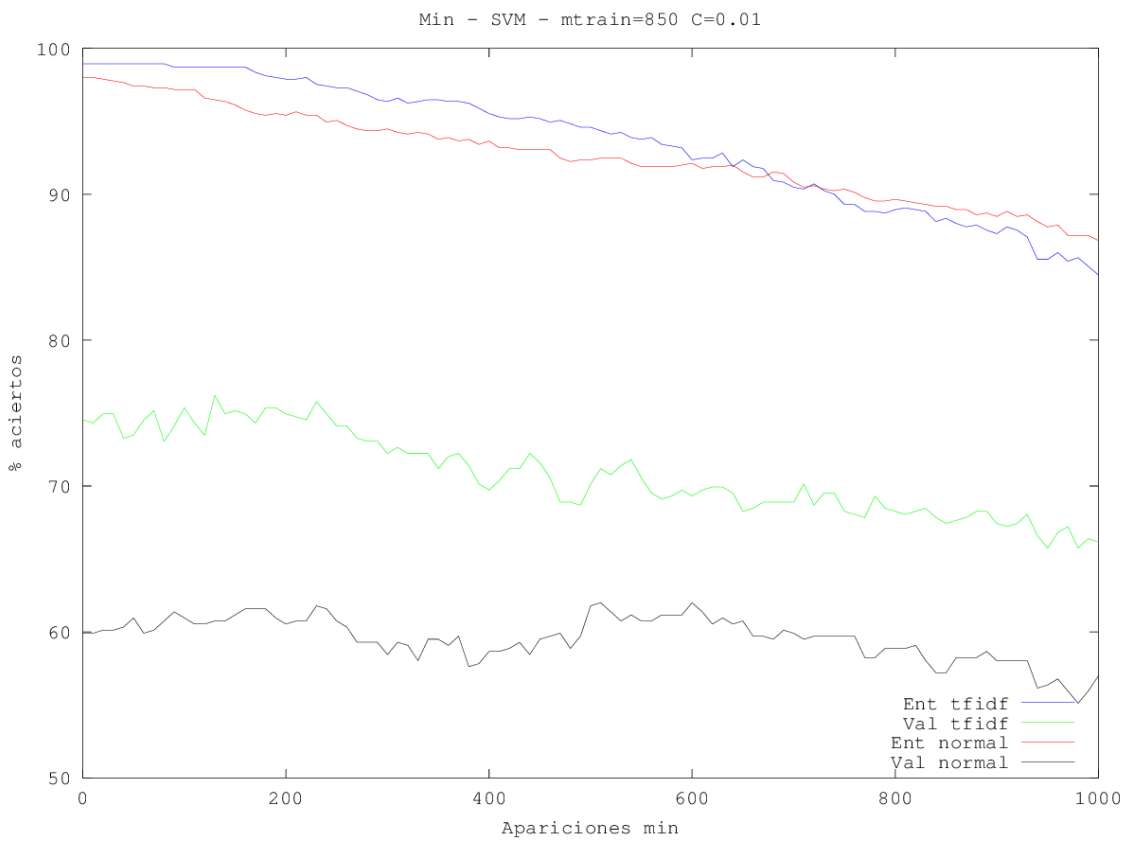

Min - SVM - mtrain=850 C=0.01
% aciertos
Apariciones min
Ent tfidf
Val tfidf
Ent normal
Val normal
100
90
80
70
60
50
0
200
400
600
800
1000

Fig. 11

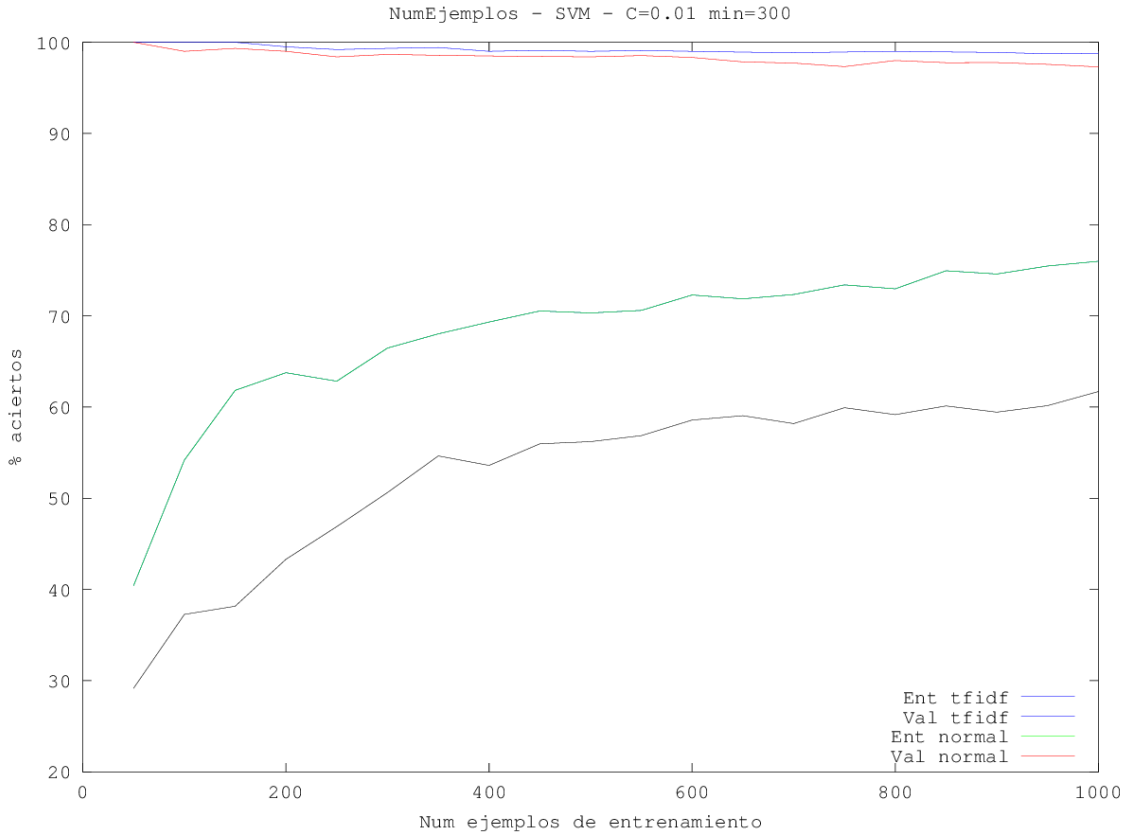

NumEjemplos - SVM - C=0.01 min=300
% aciertos
Num ejemplos de entrenamiento
Ent tfidf
Val tfidf
Ent normal
Val normal
100
90
80
70
60
50
40
30
20
0
200
400
600
800
1000

Fig. 12

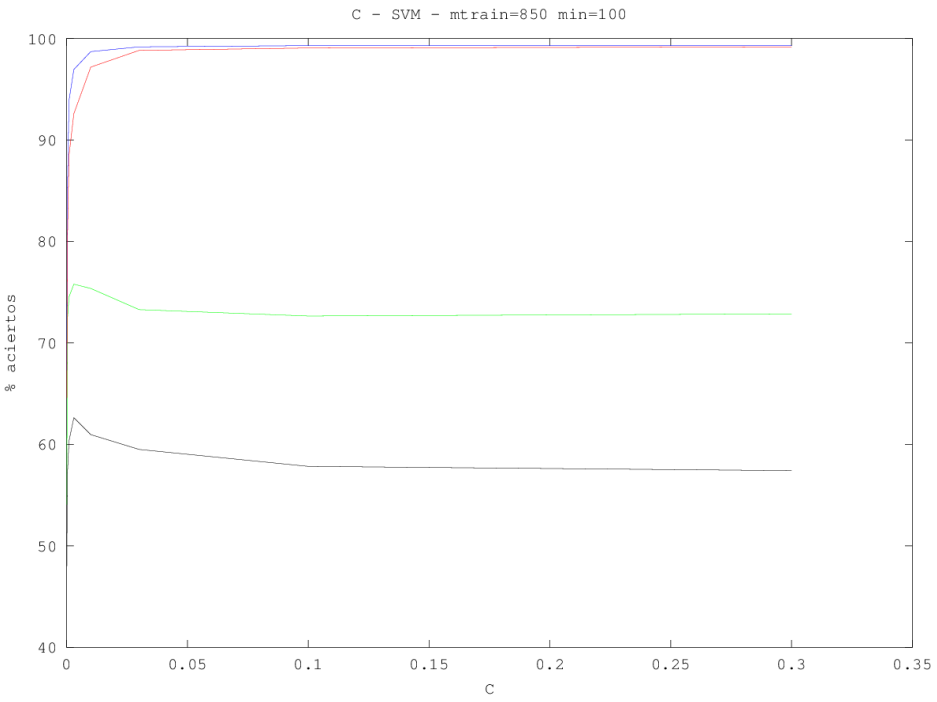

C - SVM - mtrain=850 min=100
% aciertos
C
100
90
80
70
60
50
40
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35

Fig. 13

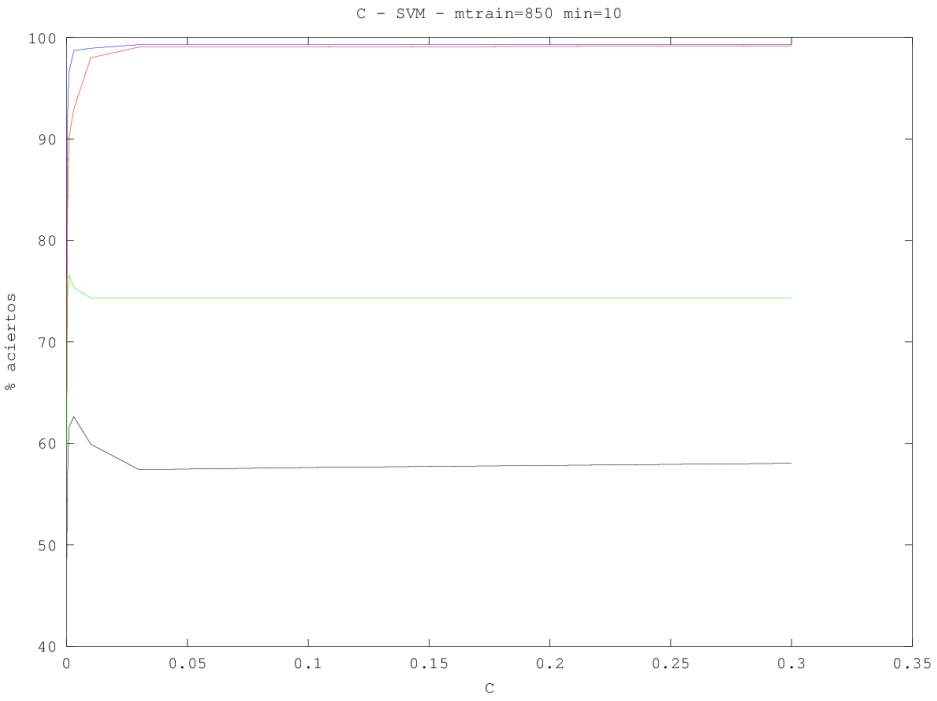

C - SVM - mtrain=850 min=10
% aciertos
C
100
90
80
70
60
50
40
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35

Fig. 14

En Fig.11 se observa que se pueden eliminar palabras que aparezcan menos de 200 veces sin que se deteriore el porcentaje de aciertos conseguido.

En cuanto a Fig.12 se observa que a más ejemplos de entrenamiento, mejores resultados se obtienen.

Comparando Fig.13 y 14, se observa que en Fig.13 el valor del parámetro C con el que se consigue el mejor porcentaje es más grande que en Fig.14. Teniendo en cuenta que la primera tiene menos palabras (pues se eliminan aquellas palabras que aparecen menos de 100 veces, frente a la 14 que son las que aparecen menos de 10), esto parece indicar que cuantas más palabras haya, se necesita un valor menor de C. Otras pruebas parecen confirmar esto.

## RESULTADOS

Con esta implementación de SVM, utilizando kernel lineal, el máximo porcentaje de aciertos alcanzado fue de un 73%, tardando unos pocos minutos en entrenarse.

# APRENDIZAJE BAYSESIANO

Para finalizar, decidí probar el aprendizaje baysesiano, algo bastante apropiado para clasificación de textos. Sin embargo, los resultados obtenidos no eran nada buenos con el siguiente código:

```
addpath Funcs/;
clear;
% Párametros del test %
mtrain = 850;

min = 0; # Palabras que aparezcan menos de min veces se eliminaran
% Carga estática de datos %
num_etiquetas = 5;
load dmoz-web-directory-topics.mat;
[X y] = cargaDatos(data, label, num_etiquetas, min);
clear data label mldata_descr_ordering;

X = double(X);
y = double(y);

num_entradas = size(X)(2);
m = size(X)(1);
n = size(X)(2);

Xtrain = double(X(1:mtrain,:));
ytrain = double(y(1:mtrain));

Xval = double(X(mtrain+1:end,:));
yval = double(y(mtrain+1:end));

% Indices para cada uno de los topics

ytrainTrans = transformaEtiquetas(ytrain, num_etiquetas);
for i=1:num_etiquetas
    idx(i).a = find(ytrainTrans(:,i));
endfor

% Calcular la probabilidad de cada topic

for i=1:num_etiquetas
    prob_topic(i) = length(find(ytrainTrans(:,i))) /
length(ytrainTrans);
endfor

% Calculamos la longitud de cada web

webs_lengths = sum(Xtrain, 2);

% Longitud de cada web para cada topic

for i=1:num_etiquetas
    t_wc(i).a = sum(webs_lengths (idx(i).a));
endfor

% Calcular la probabilidad de cada palabra para cada topic

for i=1:num_etiquetas
```

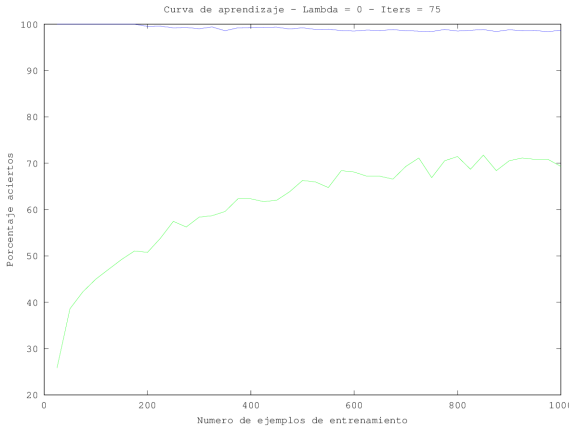
```
    prob_tokens(i).a = (sum(Xtrain(idx(i).a, :)) + 1) ./ (t_wc(i).a +
m);
endfor

% prob_tokens(i).a(k) = P(k|y=i), esto es, la probabilidad de que la
palabra k-esima sea del topic i

for i = 1:num_etiquetas
    bayesProb(:,i) = Xval * ((log(prob_tokens(i).a))' +
log(prob_topic(i)));
endfor

porcentajeAciertosBayes(bayesProb, Xval, yval )

% prob_tokens(i).a(k) = P(k|y=i), esto es, la probabilidad de que la
palabra k-esima sea del topic i
Xval = Xval > 0;
for i = 1:num_etiquetas
    for j = 1:size(Xval)(1)
        idxval = find(Xval(j,:));
        p(i,j) = prod(prob_tokens(i).a(idxval)) * prob_topic(i);
    endfor
endfor
```

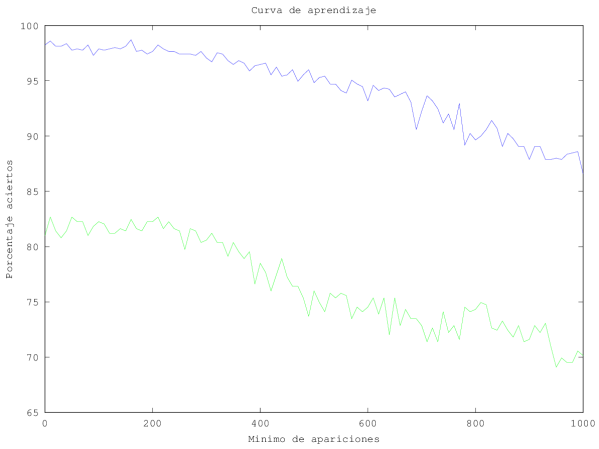
Aparentemente, palabras que no aparecen en el corpus, o en determinados documentos lleva a problemas.

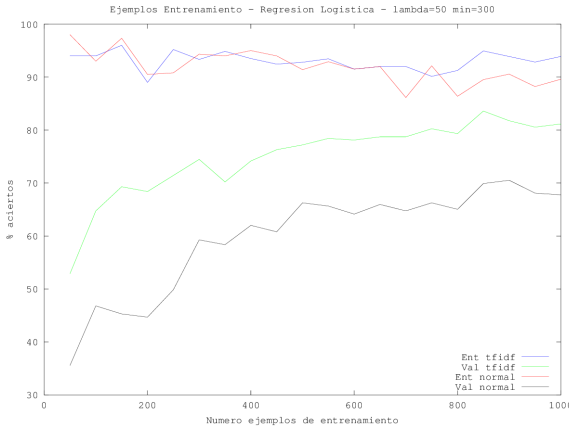
En cuanto a tiempo de ejecución, era instantáneo.

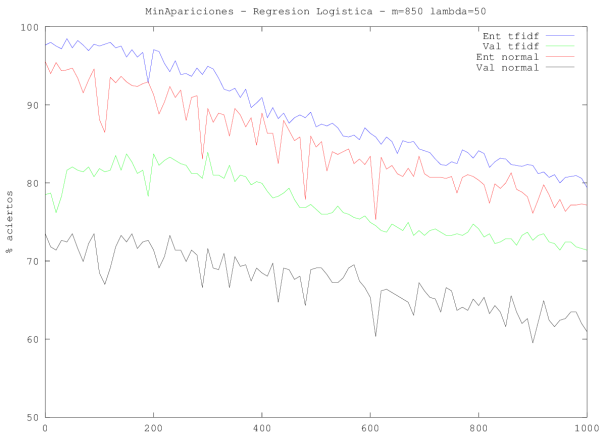
# CONCLUSIONES FINALES

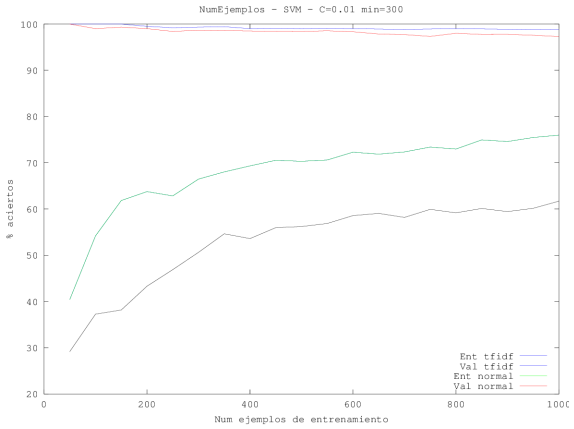
A partir de las gráficas generadas, se observa que mantienen comportamientos similares en Redes Neuronales, Regresión Logística y SVM:

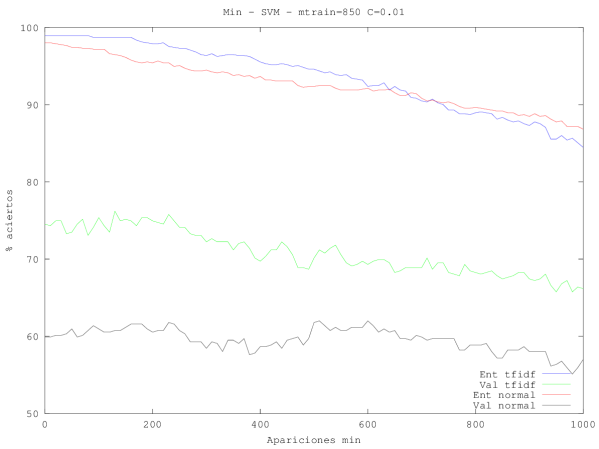


## COMPARATIVAS DE RENDIMIENTO

| | Porcentaje | Tiempo |
|---|---|---|
| **Redes** | 84 | Segundos* |
| **Logistica** | 84 | Segundos |
| **SVM** | 73 | Minutos |

*Para capas intermedias pequeñas (que tardan mucho menos que las grandes, con el mismo porcenatje de aciertos)

Claramente, es preferible el uso de Redes Neuronales o de Regresión logística frente a SVM, pues obtienen un porcentaje de aciertos mucho mejor, en menos tiempo.

## METODOS UTILIZADOS

Para la obtención del mejor porcentaje de aciertos para cada una de los diferentes métodos, he realizado pruebas sobre una de las variables (lambda, numero de ejemplos de entrenamiento…), fijando el resto, y he buscado el valor de la que he variado que daba el mejor porcentaje. A continuación, he usado el valor que acabo de encontrar para probar sobre otras variables, y así he iterado el proceso un par de veces. No es la búsqueda óptima por fuerza bruta (probar todo con todo, a lo backtracking), es algo más parecido a escalada por máxima pendiente, se obtienen unos resultados muy buenos en poco tiempo. Además, probé a comparar el "backtracking" contra este método con redes neuronales, se observaba que prácticamente todos los valores obtenidos por backtracking se diferenciaban en apenas un 1%, por lo que no merece la pena usar un método tan costoso (horas con varias ejecuciones simultáneas).

Aun así, seguramente se pueda sacar un poco más de rendimiento a lo que yo he obtenido, aunque ciertamente dudo que mucho más.