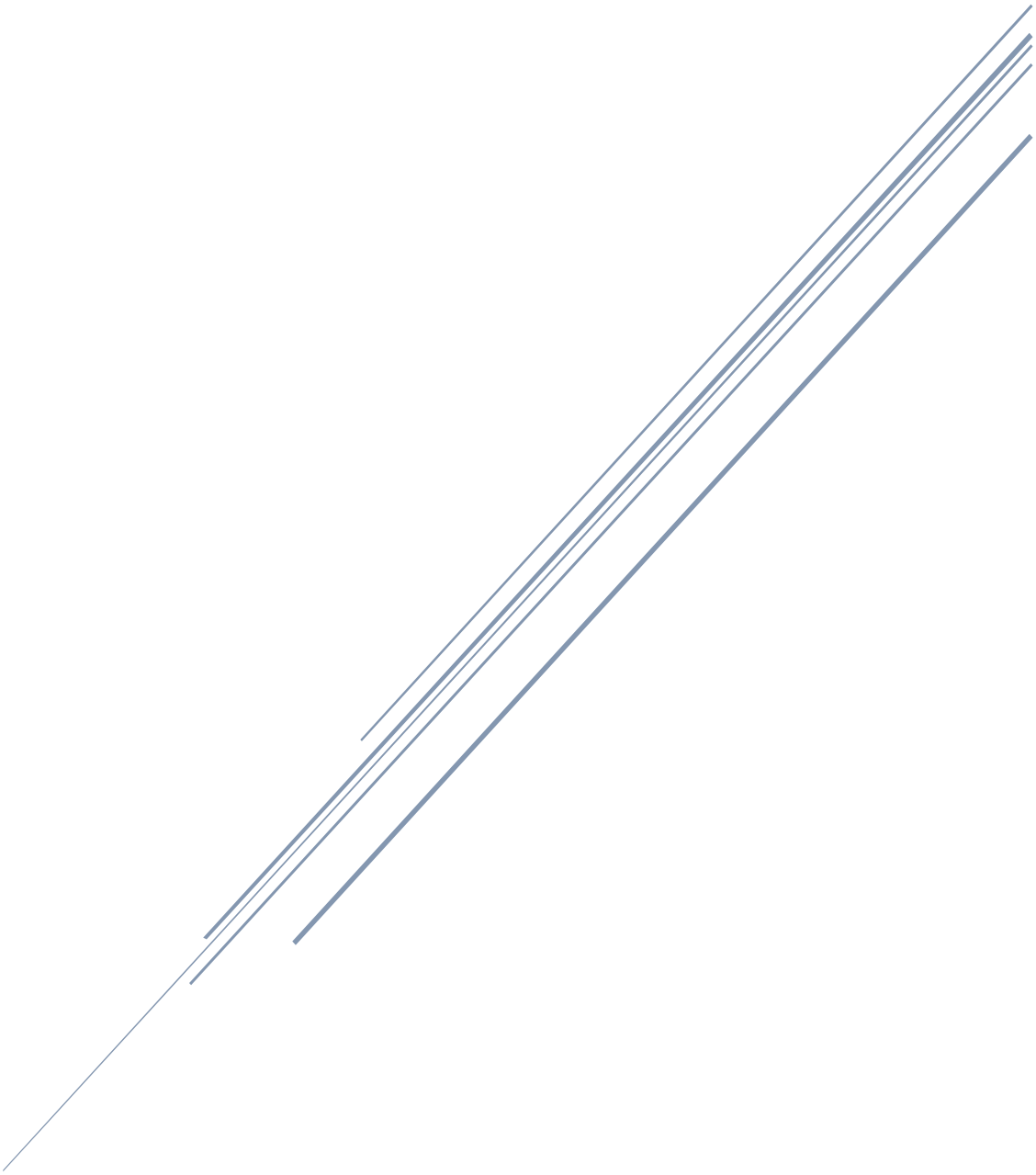
SISTEMAS INTELIGENTES

Practica II



Rafael Soria Diez

Introducción

Esta práctica consiste en implementar un sistema de aprendizaje a través de AdaBoost que consistirá en distinguir dos tipos de imágenes de dimensiones 24*24:

- Imagen con cara
- Imagen sin cara

Para efectuar esta práctica, habrá que efectuar un clasificador fuerte, que estará formado por muchos débiles, este a su vez, tiene hiperplanos. El hiperplano averigua a través del espacio de 576 dimensiones(24x24) en qué lugar de este espacio se encuentra.

Para la realización, he tenido una plantilla, la cual he añadido la siguiente clase.

Hiperplano

En primer lugar, la fórmula del hiperplano es la siguiente:

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n = b$$

La clase hiperplano está formado por:

- Un constructor, en el cual generará hiperplanos aleatorios, es decir, dividiremos el plano en x veces, para ver, en qué punto se aproxima más al resultado deseado. Tras esto normalizaremos para que tengan su misma dirección y sentido, Por ultimo calcularemos el termino el termino independiente.

- Un método denominado “ver”, en el cual mira en qué posición en el plano esta, es decir si es + esta encima, si es – está debajo y si es 0 está contenida en él.

```
public double ver(int[] aux){
    double resultado = 0;

    for(int i = 0; i < aux.length; i++)
        resultado += puntos[i] * aux[i];

    return resultado += D;

}
```

Clasificador Débil

Los métodos que más destacan en el clasificador débil son los siguientes:

- Un constructor, que contiene un hiperplano, del cual, lo crea aleatoriamente.
- Un método que, dado una imagen, determinara si es cara (si el resultado mayor que 1) o si no es cara (si el resultado es menor que 0).

```
public int mirar( Cara c){
    int result = 0;

    if(hiperplano.ver(c.getData()) < 0.0) result = -1;
    else result =  1;

    return result;
}
```

- Un método que, dado unos conjuntos de imágenes, que determina cuanto de bueno es el clasificador débil, para averiguar esto, a través de la clase creada Cara, evalúa si es cara o no, si es distinto, aumentaría el error. Por ultimo calcularía el valor de valor de confianza para la predicción descrita.

```
public void calcularError(ArrayList<Cara> listaCaras){
    error = 0;
    for(int i = 0; i < listaCaras.size();i++){
        Cara cara = listaCaras.get(i);
        if(cara.getTipo() != mirar(cara))
            error+= cara.getPeso();

    }
    // ht: αt= ½ Ln(1-εt/εt)
    valorConfianza = 0.5 * Math.log((1 - error)/error);
}
```

- Por último, un método entrenar, en el cual llamaremos en el método Adaboost, que determina cuál de ellos tienen mejor resultados (menor tasa de error).

```
public void entrenar(ArrayList<Cara> listaCaras,int x){
    ClasificadorDebil mejor = new ClasificadorDebil();
    //para que entre en el primero si o si
    mejor.setError(Double.MAX_VALUE);
    //ClasificadorDebil mejor = null;
    for(int i = 0; i < x;i++){
        ClasificadorDebil aux = new ClasificadorDebil();
        aux.calcularError(listaCaras);
        if(aux.getError() < mejor.getError()){
            mejor = aux;
            hiperplano = mejor.hiperplano;
            valorConfianza = mejor.valorConfianza;
            error = mejor.error;
        }
    }
}
```

Clasificador Fuerte

Los métodos más relevantes son:

- Un constructor, donde el atributo más importante, es un ArrayList de clasificadores débiles, ya que el clasificador fuerte, lo podemos denominar como un conjunto de estos. Además, para sacar pruebas, he añadido otro ArrayList, donde recojo los valores de los porcentajes obtenido en cada iteración, para evaluar los resultados de Adaboost.

- Un método, que, dado una imagen, valora que determina el conjunto de clasificadores débiles, es decir, si la mayoría da como buena que es cara la imagen, el clasificador fuerte, valórala como optimo el resultado,

```
public int determinarCara(Cara c){
    double result = 0;
    int esCara = 0;

    ClasificadorDebil cDebil = new ClasificadorDebil();
    for(int i = 0; i< debiles.size();i++){
        cDebil = debiles.get(i);
        result += cDebil.getValorConfianza() * cDebil.mirar(c);
    }
    if(result < 0.0) esCara =  -1;
    else  esCara =  1;

    return esCara;
}
```

Adaboost

Por último, esta clase, solo contiene el método Adaboost, el cual consiste en entrenar el clasificador para tener el resultado más óptimo:

- En primer lugar, inicializo la distribución de pesos, D 1 (i) = 1/N sobre el conjunto de entrenamiento.

```
for(int i = 0; i <listaCaras.size();i++){
    Cara cara = listaCaras.get(i);
    cara.setPeso((double) 1.0/listaCaras.size());
}
```

- En segundo lugar, entreno el clasificador débil, y recojo el mejor valor de confianza.

```
//entremaos el mejor y lo recojo
ClasificadorDebil mejor = new ClasificadorDebil();
mejor.entrenar(listaCaras,numClasificadores);
fuerte.IntroducirFuerte(mejor);
//System.out.println(mejor.getError());
double valorConfianza = mejor.getValorConfianza();
```

- En tercer lugar, actualizo los pesos, según el clasificador débil obtenido

- Por ultimo miro la cantidad de aciertos obtenido y lo guardo, para evaluar los resultados obtenidos.

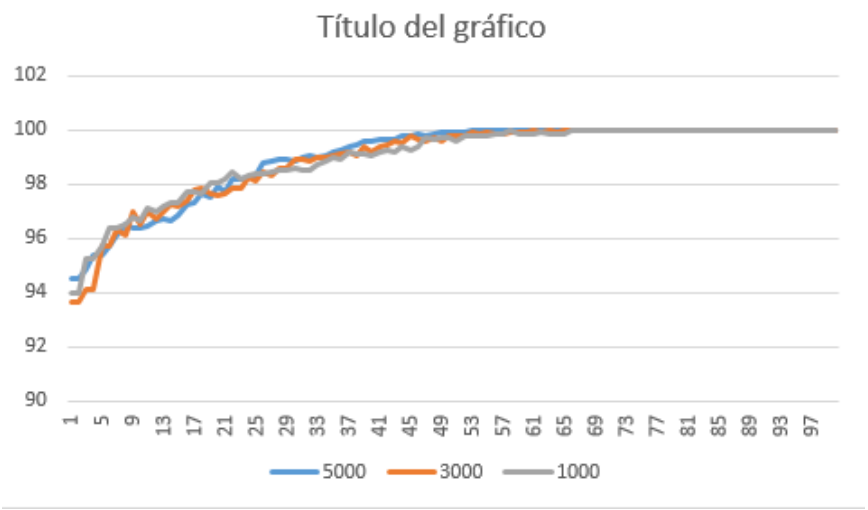
```
double aciertos = 0;
for(int j = 0; j <listaCaras.size();j++){
    Cara cara = listaCaras.get(j);
    if(fuerte.determinarCara(cara) == cara.getTipo()) aciertos++;
}
//guardo el porcentaje
porcentaje = (100.0 * aciertos/listaCaras.size());
fuerte.IntroducirPorcentaje(porcentaje);
```

Pruebas

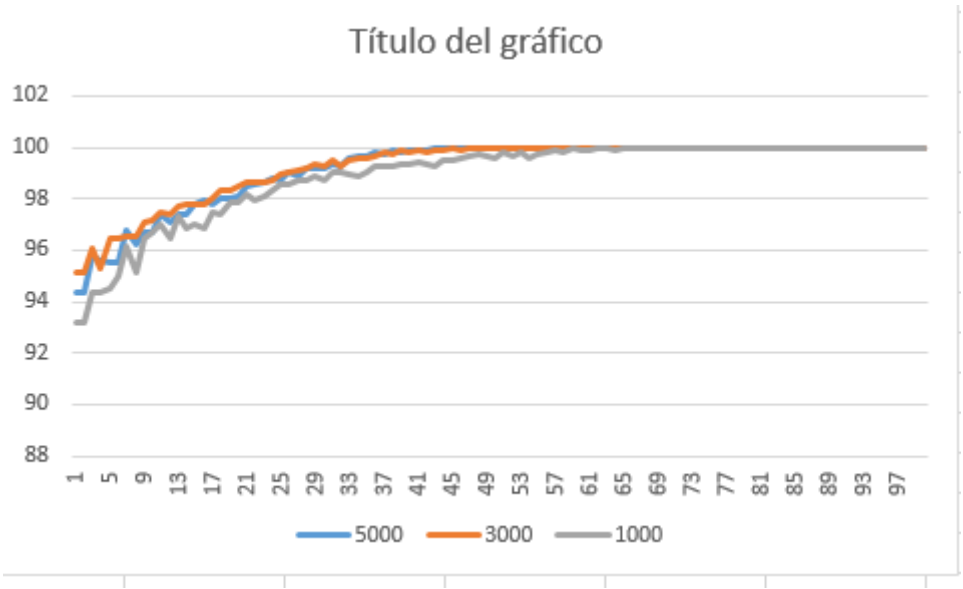
Aprendizaje

He detallado unas clases de pruebas, para ver el funcionamiento del entrenamiento, con iteraciones de 100, y variando el valor testRate y el número de clasificadores débiles (Nota: el eje Y representa el porcentaje acierto, y el eje X las repeticiones).

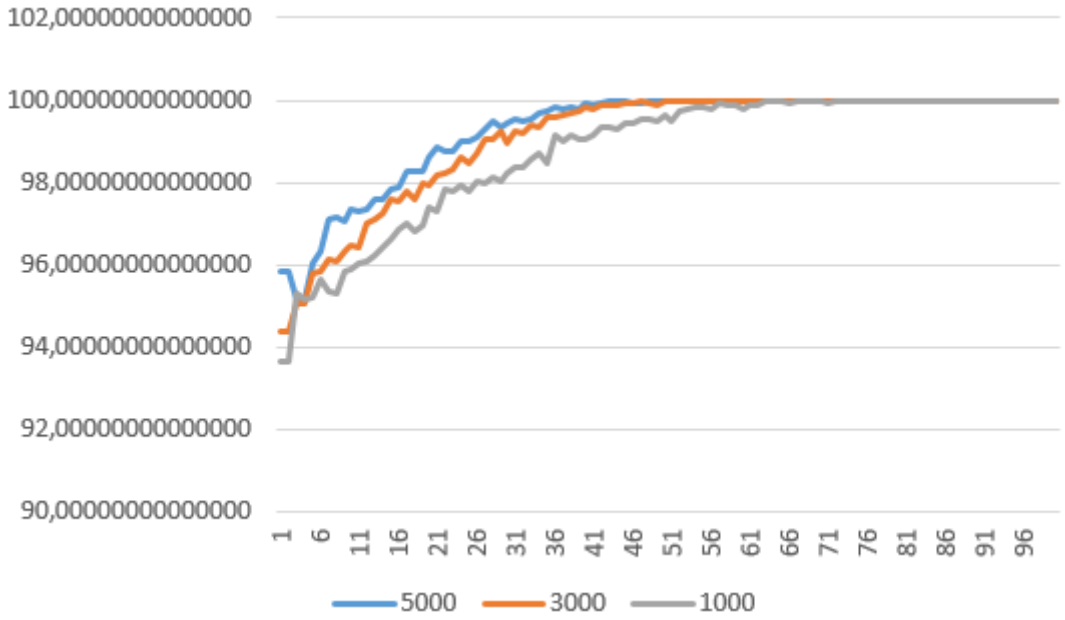
TestRate: 0.1



TestRate: 0.2



TestRate: 0.3

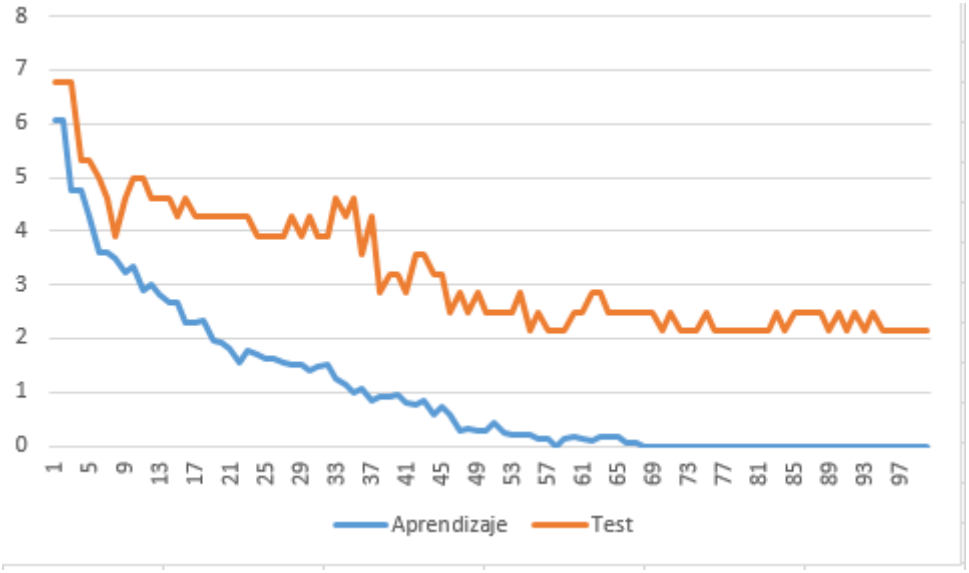


Las tres graficas tienen en común, que cuanto más clasificadores débiles, se introduce, más rápido alcanza el 100%, ya que, al generar más hiperplanos aleatorios, tiene más posibilidades de acertar. Por otro lado, podemos observar, que al aumentar el testRate, es decir hay más imagenes para el test y menos para el aprendizaje, se genera el 100% más rápido. Lo podemos observar, en las gráficas 0.3 y 0.1, en el cual, el 0.1 alcanza el resultado óptimo más rápido.

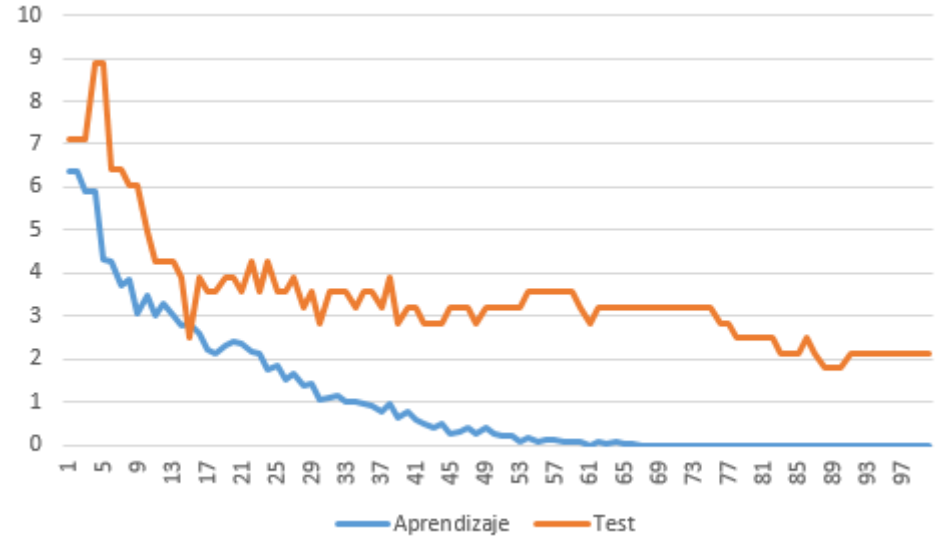
Errores

He detallado unas clases de pruebas, para ver los errores tanto del aprendizaje, como del test, con iteraciones de 100, y variando el valor testRate y el número de clasificadores débiles (Nota: el eje Y representa el porcentaje de error, es decir de 8-1%, y el eje X las repeticiones).

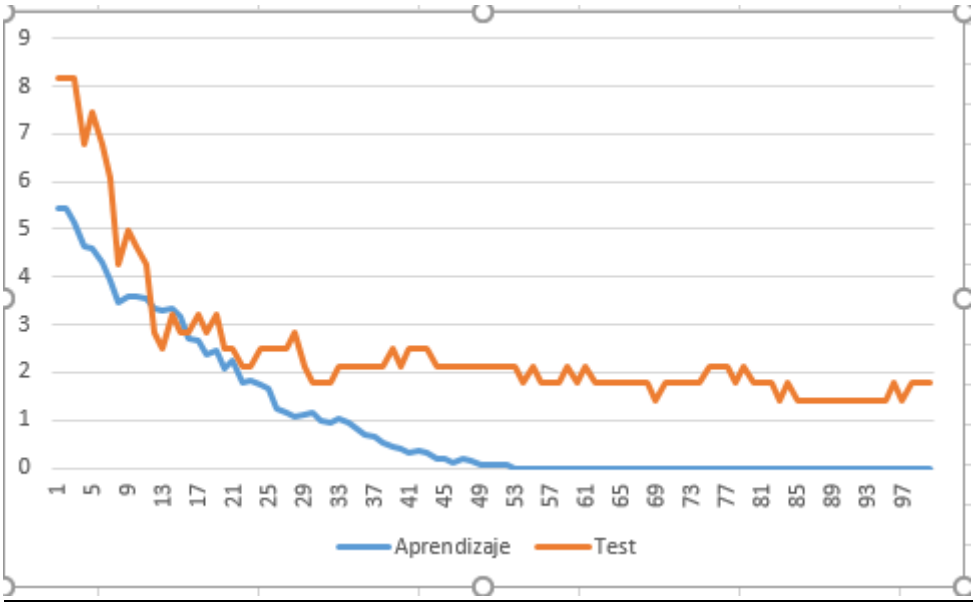
TestRate: 0.1 y 1000 clasificadores



TestRate: 0.1 y 3000 clasificadores

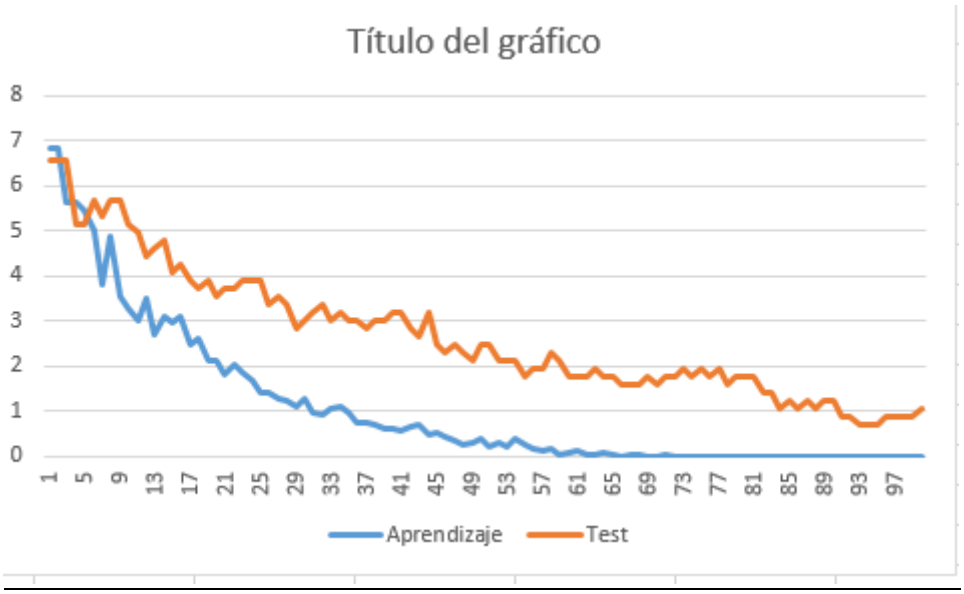


TestRate: 0.1 y 5000 clasificadores

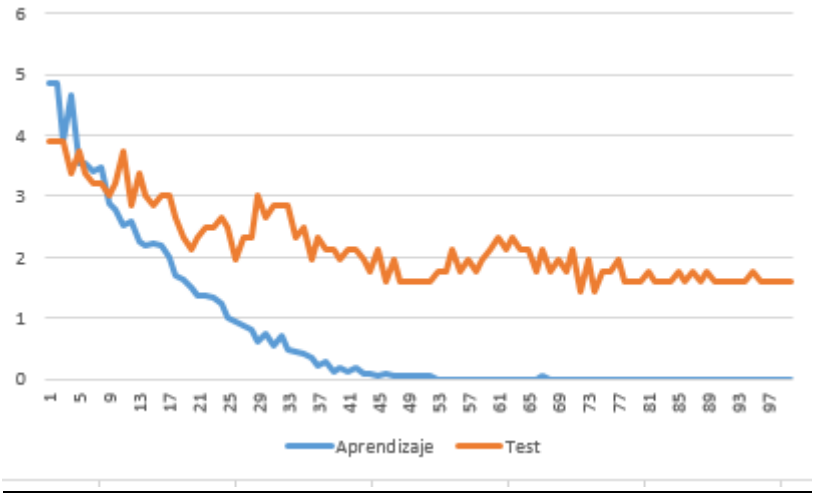


Con TestRate 0.1 podemos observar, que en los errores del test, hay momentos que tienen grandes iteraciones, donde tiene un sobre-entrenamiento, es decir, que memoriza valores ya visto antes, esto puede ser producido, al excesiva cantidad de imágenes de aprendizaje frente al test.

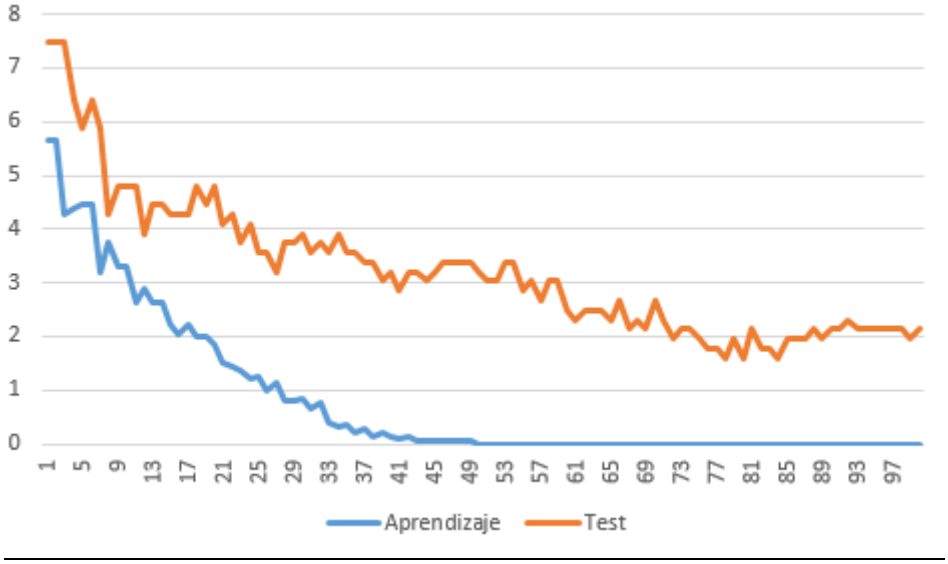
TestRate: 0.2 y 1000 clasificadores



TestRate: 0.2 y 3000 clasificadores

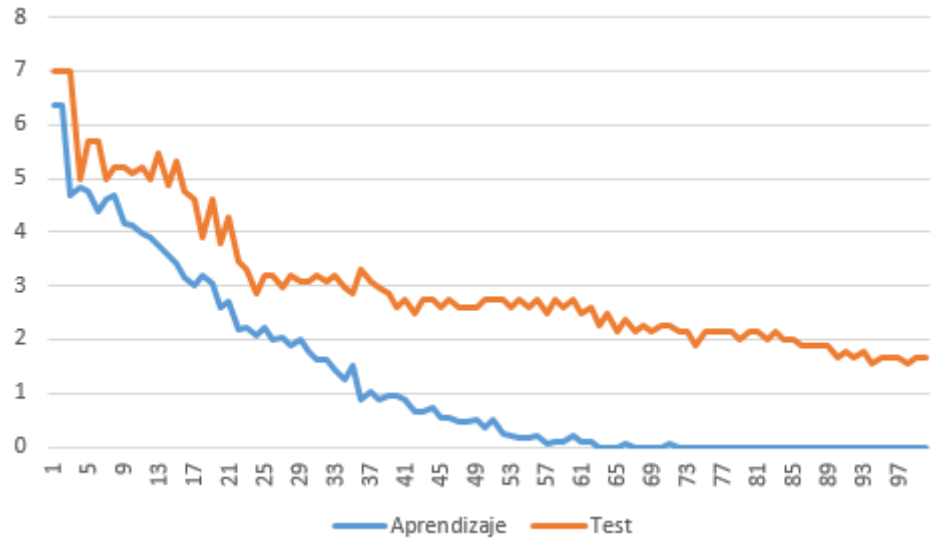


TestRate: 0.2 y 5000 clasificadores

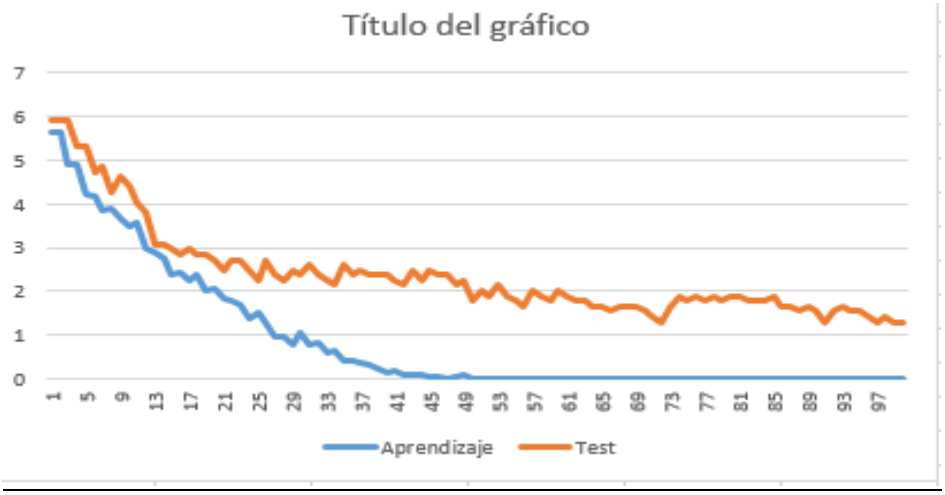


Con TestRate 0.2 podemos observar, que en el caso de 1000 hiperplanos, no tiene sobre-entrenamiento, en cambio, cuantos más hiperplanos aleatorios generamos,mas sobre-entrenamiento contiene. Sin embargo, al aumentar la cantidad de imágenes en el test, la capacidad de memorizar disminuye.

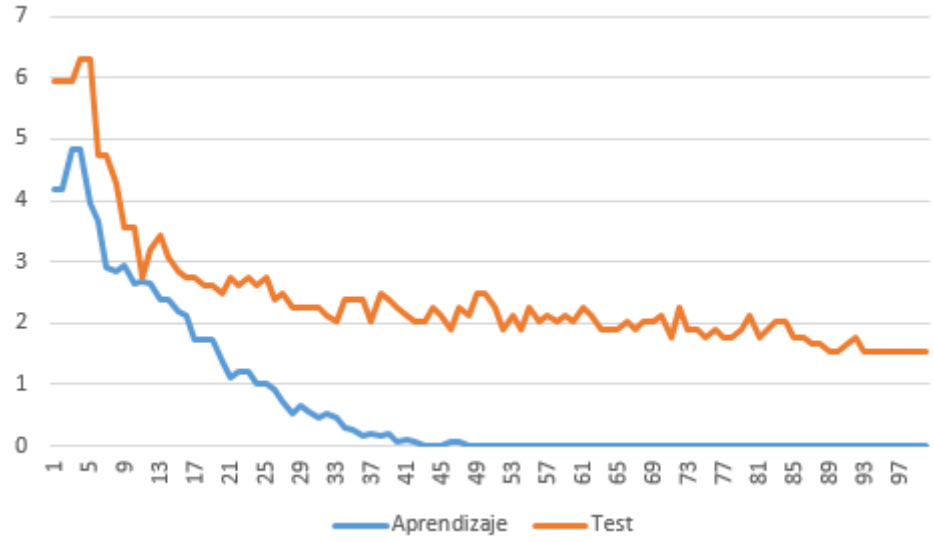
TestRate: 0.3 y 1000 clasificadores



TestRate: 0.3 y 3000 clasificadores



TestRate: 0.3 y 5000 clasificadores



Por último, en testRate con un valor de 0.3, podemos observar, que apenas se produce un sobre-entrenamiento, dando un poco en la gráfica con más hiperplanos generados, es decir la de 5000.

Respecto a los resultados obtenidos, podemos deducir, que la cantidad exacta, para tener un buen resultado, y que no sobre-entrene seria con un valor de tesRate de 0.3 y una cantidad de hiperplanos aleatorios de 1000 y 3000.