Introducción al Reconocimiento de Patrones 2015

Proyecto Final

Clasificación de bosques por información cartográfica

Santiago Martinchich Pablo Soto

Instituto de Ingeniería Eléctrica

Facultad de Ingeniería
Universidad de la República

Diciembre de 2015

Índice

1. Introducción

1.1. Descripción del problema

El problema consiste en clasificar el tipo mayoritario de árboles en zonas determinadas de bosque nativo. El estudio incluye cuatro áreas de bosques naturales, situados en el norte del estado de Colorado, en EEUU.

Se trabaja con observaciones que corresponden a áreas de 30x30 metros de bosque. Las observaciones están compuestas por las siguientes características:

- Elevación (en metros)
- Azimut del terreno (en grados)
- Pendiente del terreno (en grados)
- Distancia horizontal a la fuente de agua superficial más cercana (en metros).
- Distancia vertical a la fuente de agua superficial más cercana (en metros).
- Distancia horizontal a la ruta más cercana (en metros).
- Índice de sombra a las 9am en el solticio de verano (valor de 0 a 255).
- Índice de sombra al mediodía en el solticio de verano (valor de 0 a 255).
- Índice de sombra a las 3pm en el solticio de verano (valor de 0 a 255).
- Distancia horizontal a puntos donde hubo comienzo de incendios (en metros).
- Nombre del área de bosque natural a la que pertenece (4 posibles).
- Tipo de suelo (40 tipos posibles).

Se cuenta con un conjunto de datos de entrenamiento compuesto por 7560 observaciones, conteniendo las características y el tipo de cobertura de árboles presente en cada parcela, y se cuenta con un conjunto de datos de test de tamaño similar.

Los datos disponibles fueron extraídos de la siguiente base pública:

`https://archive.ics.uci.edu/ml/datasets/Covertype`

1.2. Objetivos

Diseñar un sistema de reconocimiento de patrones que permita, a partir de las características de una determinada área de 30x30 metros de bosque, determinar cuál es el tipo de cobertura de árboles mayoritario en dicha área.

El sistema debe maximizar el porcentaje de acierto asumiendo que las clases de tipos de bosque son equiprobables. Se debe trabajar, en una primera instancia, con un conjunto de datos de entrenamiento y hacer una posterior evaluación con un conjunto de test que posee una cantidad igual de elementos por clase.

2. Análisis preliminar

2.1. Análisis de los datos

Una primera aproximación a los 7560 datos de entrenamiento disponibles arroja que estos están ligeramente desbalanceados en cuanto a su distribución a lo largo de las distintas clases:

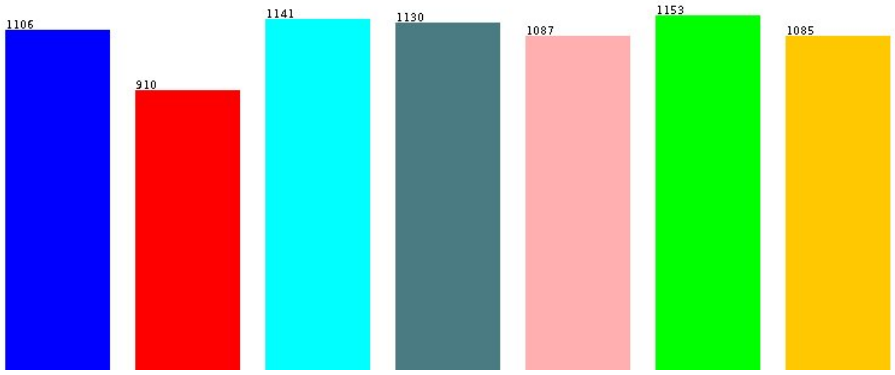
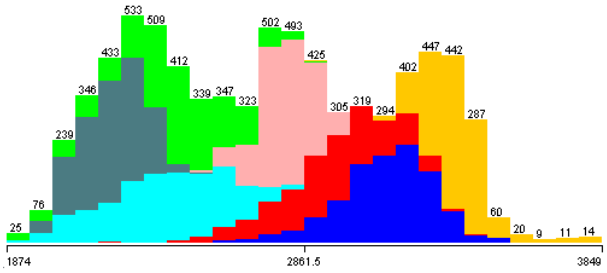


Figura 1: Cantidad de datos de entrenamiento para cada clase

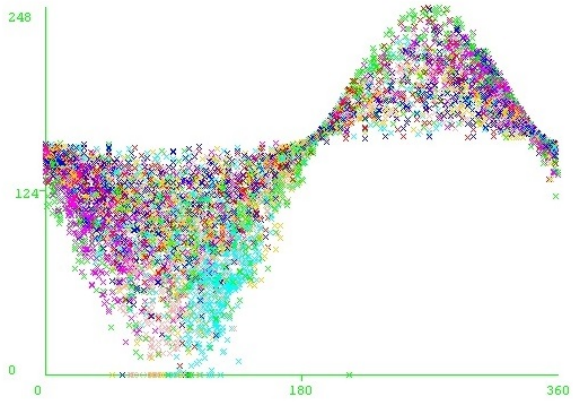
Haciendo un análisis visual de cada característica por separado, observamos que existen algunas que tienen por sí mismas cierto poder de discriminación, sin llegar ninguna de ellas a ser determinantes. Un estudio de a pares de características arroja resultados similares, obteniéndose que, como es esperable, existen ciertas correlaciones entre características, y que las "nubes de puntos"que se obtienen muestran una gran mezcla de elementos de distintas clases.

En general, se observa que el problema presenta una considerable complejidad, que deberá ser abordada haciendo uso de un número grande de características para su satisfactoria resolución.

A modo ilustrativo, se muestran a continuación las suguientes gráficas:



(a) Distribución de clases según elevación del terreno.



(b) Distribución de clases respecto al azimut del terreno y el índice de sombra a las 3pm.

2.2. Pruebas con distintos clasificadores

En una primera instacia, se realizaron pruebas de clasificación midiendo el resultado de varios tipos de clasificadores. Se ingresaron los datos al software Weka y se realizaron las pruebas con los datos originales de entrenamiento sin ningún tipo de preproceso, realizando validación cruzada con 10 particiones (10-fold CV) para estimar el desempeño.

Las técnicas utilizadas fueron:

- **NaiveBayes.** Clasificador probabilístico paramétrico basado en el teorema de Bayes. Se implementó con núcleos de tipo gaussiano.
- **K-NN.** Método de clasificación no paramétrico que clasifica a cada nuevo patrón teniendo en cuenta, en una votación ponderada, a los K vecinos (datos de entrenamiento) más cercanos. Previa normalización de los datos y trabajando con la distancia euclídea, se varió entre 1 y 10 el valor de K obteniéndose los mejores resultados para K=1.
- **C4.5.** Algoritmo que genera un árbol de decisión que tiene en cuenta el criterio de ganancia de información para elegir la división que se realiza en cada nodo. Distintos árboles fueron inicializados a partir de distintas semillas. Se usó 0.25 como *confidence factor* del podado.
- **RandomForest.** Técnica que combina una gran cantidad de árboles de decisión generados aleatoriamente. Se realizaron pruebas con bosques de 500 árboles.

Cada clasificador fue corrido 10 veces distintas, previo reordenamiento (resample) de los datos. La Tabla 1 resume los resultados obtenidos para cada clasificador.

Clasificador	Media de acierto en 10-fold CV (%)	Varianza del acierto
Naive Bayes	65.85	0.03
K-NN (K=1)	79.03	0.04
C4.5	77.42	0.12
Random Forest	84.10	0.02

Tabla 1

Observar que con *RandomForest* se logra un acierto superior al 82 %, alcanzando de esta forma uno de los objetivos de la tarea.

2.3. Elección de técnica a utilizar

En vista de los resultados satisfactorios obtenidos con las anteriores técnicas, del interés particular de los participantes del proyecto y con la idea de centrarse es una única técnica para desarrollarla con cierta profundidad, se decidió continuar el trabajo centrándose en la técnica de Redes Neuronales.

3. Experimentación con Redes Neuronales

Como punto de partida se evaluaron los resultados generados a partir de una Red Neuronal construida con Weka. Para esto, se construyeron diferentes redes modificando tanto la cantidad de capas ocultas como el *learning rate* de las mismas. En la siguiente tabla se muestran los resultados, en media, luego de 3 iteraciones con validación cruzada de 10 particiones:

		Capas ocultas	
		$n_1 = 30$	$n_1 = 50$, $n_2 = 20$
Learning rate	0.1	49.59	76.39
	0.01	49.81	76.85
	0.001	67.85	60.46

Nota: n_i denota la cantidad de nodos en la capa i.

Cabe destacar que Weka no permite especificar las funciones de activación, ya que utiliza activación *Sigmoid* por defecto para todos los nodos de las capas ocultas. Tampoco provee muchas herramientas para evitar problemas tales como el sobreajuste de la red.

A partir de estas y otras consideraciones, se decidió continuar el trabajo programando las redes en Python haciendo uso de la biblioteca Keras[1].

Durante la experimentación se buscó probar distintas posibilidades para gran parte de los componentes que hacen a una Red Neuronal. Se trabajó, en su mayoría, particionando randómicamente el conjunto de entrenamiento original en una proporción de 70:30, dejando el conjunto mayor como conjunto de entrenamiento y el restante como conjunto de test.

Se reseñan a continuación las pruebas realizadas con sus respectivos resultados.

3.1. Preprocesamiento de datos

3.1.1. Estandarización

Este procedimiento tiene como objetivo preparar los datos para que la Red Neuronal sea capaz de aprender y clasificar de forma más eficiente. Se busca que los valores de cada característica estén distribuidos de modo que su media sea igual a 0 y su varianza igual a 1. Al tener media 0, se logra centrar los valores de cada característica con respecto a la función de activación. Por otro lado, la varianza 1 evita que aquellos valores de la entrada que se encuentren muy alejados de 0 provoquen cambios significativos en las actualizaciones de

[1] `http://keras.io/`. Keras es una biblioteca de Python que tiene como objetivo permitir la experimentación y prototipado de Redes Neuronales de forma simple.

los pesos, uniformizando la escala intercaracterísticas. Ver, por ejemplo, *Efficient BackProp* sección 4.3 [1] para una justificación más detallada.

De esta forma, se transformó cada patrón $X = (X_1, .., X_{54})$ en un patrón $Y = (Y_1, .., Y_{54})$ tal que:

$$Y_i = \frac{X_i - \hat{\mu_i}^{train}}{\hat{\sigma_i^{train}}} \quad \forall\, i = 1, \ldots, 54$$

donde

$$\hat{\mu_i}^{train} = \frac{1}{N^{train}} \sum_{j=1}^{N^{train}} X_i^j \quad \text{y} \quad \hat{\sigma_i}^{train} = \sqrt{\frac{1}{N^{train}} \sum_{j=1}^{N^{train}} (X_i^j - \mu_i)},$$

siendo N^{train} la cantidad de patrones de entrenamiento. Es decir, se estandarizaron los patrones de acuerdo al conjunto de patrones de entrenamiento.

3.1.2. Extracción y creación de características

Se investigó generar nuevas características a partir de las ya existentes. En particular, se hizo énfasis en la utilización de las descripciones de cada tipo de suelo. A partir de estas descripciones se pudieron definir nuevas características agrupando suelos pertenecientes a una misma familia o suelos que compartieran iguales propiedades como 'rocoso', 'muy rocoso', etc.

Una vez generadas las nuevas características, se evaluó la red, por un lado, sustituyendo las entradas correspondientes al tipo de suelo por las nuevas recientemente creadas, y por otro, agregando estas nuevas entradas a las ya existentes. En ninguno de los dos casos se observaron mejoras, por lo que se optó por continuar con las entradas originales.

Vale la pena observar que al agregar las nuevas entradas a las ya existentes, se corre el riesgo de aumentar la complejidad del problema. Esto se debe a que al ser generadas unas a partir de otras, las mismas quedan correlacionadas, lo que puede resultar en un problema más difícil de resolver. Ver, nuevamente, *Efficient BackProp* sección 4.3 [1] para más detalles.

Por otro lado, se observó que ciertos tipos de suelo no se encontraban presentes en ninguno de los datos de entrenamiento. Al no tener poder de discriminación, las características correspondientes a dichos tipos de suelo fueron removidas.

3.1.3. Reducción de dimensionalidad

Se realizaron pruebas de reducción de dimensionalidad como PCA, LDA y Autoencoders. Esta última es una técnica propia de Redes Neuronales que consiste en entrenar una red tal que la cantidad de unidades en su entrada y salida sea igual a la dimensión de los patrones originales, y que la cantidad de unidades en su (única) capa oculta sea igual a la dimensión a la que se desea llegar. Luego de entrenar con el conjunto de entrenamiento de forma tal que

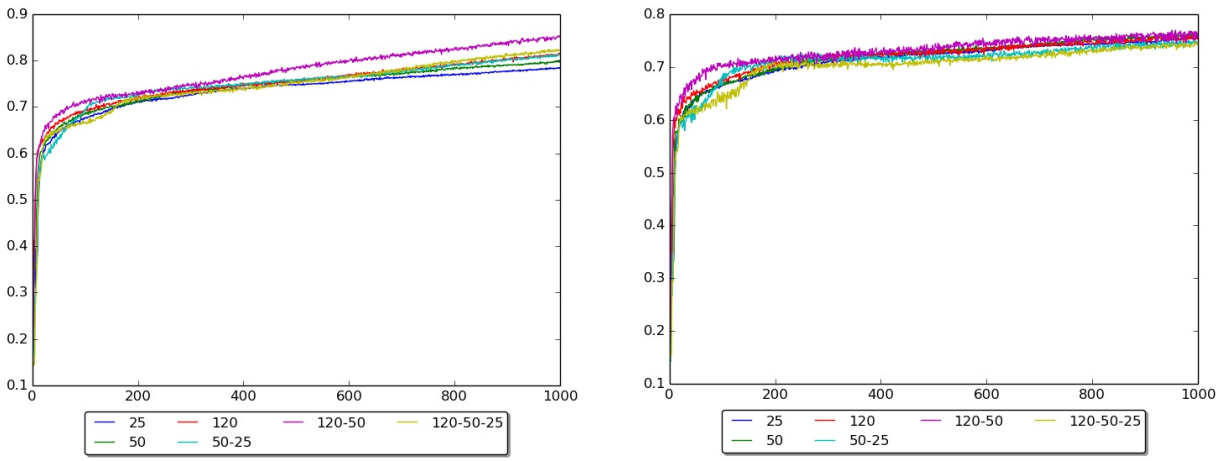
la salida esperada para cada patrón sea el mismo patrón, se obtiene que la transformación entre la capa de entrada y la oculta es una reducción de la dimensionalidad que preserva, hasta cierto punto, la información contenida el conjunto de entrenamiento original.

Ninguna de estas técnicas arrojó mejoras en el desempeño de las redes realizadas y por tanto fueron descartadas.

3.2. Capas ocultas

Se probaron redes con una, dos y hasta tres capas ocultas, con distintas cantidades de unidades cada una.

En las siguiente gráficas se muestra la evolución del porcentaje de acierto a lo largo de 1000 iteraciones de entrenamiento. En este caso, a partir de la experiencia empírica obtenida (ver sección 3.3), se utilizó *Sigmoide* y *Softmax* como funciones de activación y *Mean Square Error* como función a minimizar:



(a) Entrenamiento

(b) Test

Figura 3: Evolución del porcentaje de acierto de diferentes configuraciones de las capas ocultas a lo largo de 1000 iteraciones de entrenamiento.

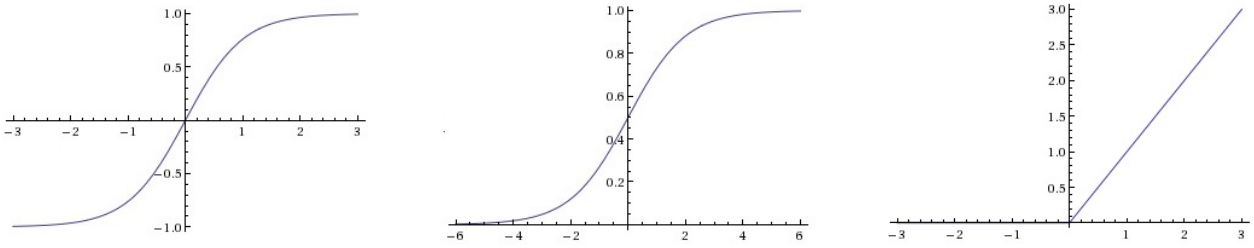
Se obtuvieron los mejores resultados con una red de 120 unidades en la primer capa oculta y 50 unidades la segunda. Con un desempeño levemente inferior se ubicó una red con una única capa oculta de 120 unidades.

3.3. Activaciones

Se evaluaron diferentes combinaciones de funciones de activación, utilizando la cantidad y dimensiones de capas ocultas que dieron mejores resultados en la sección anterior.

Se probaron las siguientes funciones de activación:

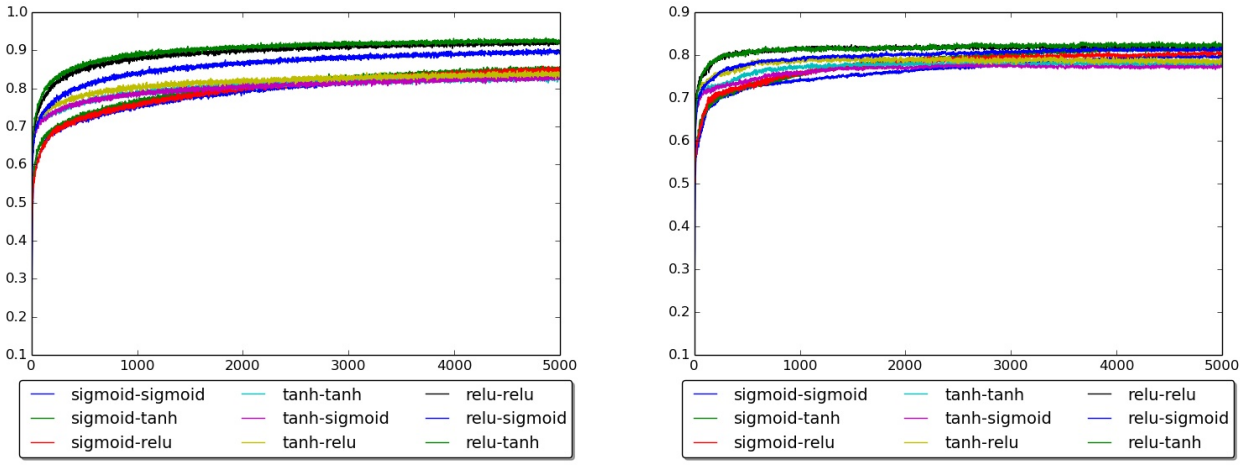
- *Sigmoide:* $S(t) = \frac{1}{1+e^{-t}}$
- *Tangente hiperbólico:* $tanh(t) = \frac{1-e^{-2t}}{1+e^{-2t}}$
- *ReLU (Rectified Linear Unit) :* $f(t) = \text{máx}\{0, t\}$



(a) Sigmoid (b) Tangente hiperbólico (c) ReLU

Figura 4: Funciones de activación

En las siguientes gráficas se puede observar la evolución del porcentaje de acierto para las distintas funciones de activación a lo largo de 1000 iteraciones de entrenamiento:



(a) Entrenamiento (b) Test

Figura 5: Evolución del porcentaje de acierto de diferentes configuraciones de funciones de activación a lo largo de 1000 iteraciones.

En la tercer capa se optó por usar *Softmax* como función de activación debido a que posee propiedades muy favorable para la capa de salida en un problema de clasificación de más de

dos clases. La función *Softmax* está definida como

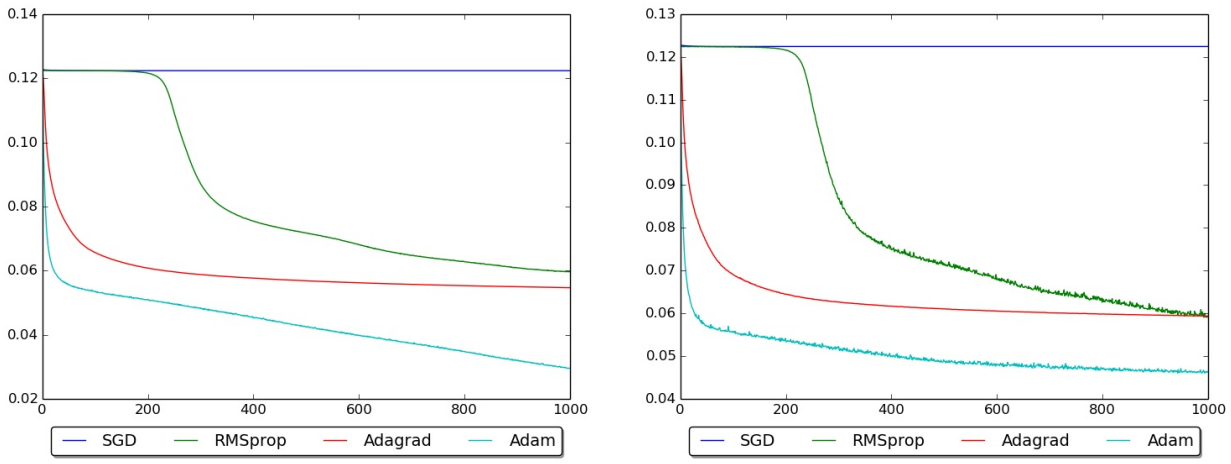
$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{n_s} e^{z_k}} \qquad \forall j = 1, .., n_s,$$

donde z_j es la entrada al nodo j de la capa de salida y n_s es la cantidad de nodos en la salida, en nuestro caso, 7. Esta función provee dos características importantes, por un lado genera una salida entre 0 y 1, y por otro, la suma de todas las salidas es igual 1. De esta forma, la salida se puede ver como una distribución que indica cuál es la probabilidad de una muestra de pertenecer a cada clase.

Como se puede observar en la Figura 5, si bien ciertas configuraciones se destacan en su porcentaje de acierto con lo datos de entrenamiento, ninguna se distingue particularmente con los datos de test. Asimismo, en varios casos existe un sobreajuste en los datos de entrenamiento. Debido a esto, se decidió definir las mejores funciones de activación una vez definidos los mecanismos que permitieran enfrentar mejor el problema del sobreajuste. Las pruebas de estos mecanismo se desarrollan en la sección 3.5.

3.4. Métodos de optimización

Se evaluaron distintos métodos de optimización con el objetivo de encontrar el que hiciera converger mejor y más rápido a la red. Los métodos analizados fueron *SGD*, *RMSprop*[2], *Adagrad*[3] y *Adam*[4]. En la siguiente gráfica se muestra, para una de las redes que brindaron mejor desempeño, la evolución del error (*Mean Square Error*) durante 1000 iteraciones de entrenamiento:



(a) Entrenamiento (b) Test

Figura 6: Evolución del error en 1000 iteraciones para diferentes métodos de optimización.

Se obtuvo de esta forma que el método *Adam* resultó superior a los restantes, tanto con los datos de entrenamiento como con los de tests.

Adam es un método basado en descenso por el gradiente que actualiza los valores de los pesos considerando el historial de cambios para cada parámetro a partir de la siguiente regla de actualización de pesos:

$$W_{t+1} = W_t - \alpha \frac{\sqrt{1-\beta_2}}{1-\beta_1} \frac{m_t}{\sqrt{v_t} + \epsilon},$$

donde $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ y $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$, siendo $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$ y $v_t = \frac{v_t}{1-\beta_2^t} g_t^2$, con $g_t = \nabla f(W_{t-1})$ y f la función objetivo que se desea minimizar. El parámetro α denota el tamaño de cada paso y los parámetros β_1 y β_2 los decaimientos exponenciales.

Los valores sugeridos para los parámetros son: $\alpha = 0{,}001$, $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$ y $\epsilon = 10^{-8}$. Los mismos son los recomendados en el artículo [4] que presenta el método y fueron los usados en el presente trabajo.

3.5. Técnicas de generalización

A partir de las primeras pruebas realizadas, se generó el problema del sobreajuste. Es decir, a medida que avanzaba el entrenamiento de la red, se llegaba a un punto en el que el desempeño de la misma sólo mejoraba para el conjunto de entrenamiento, y no lo hacía, o incluso empeoraba, para el conjunto de test.

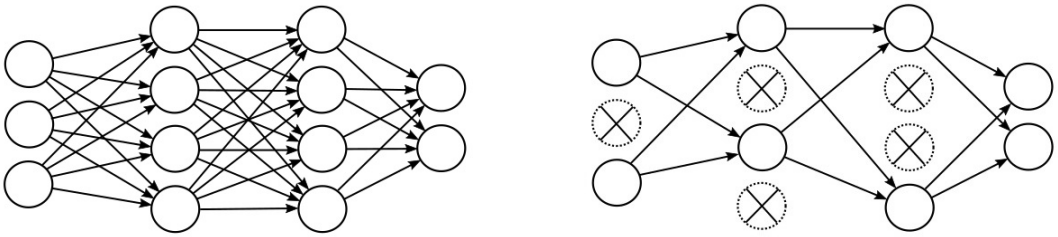
Con el objetivo de tratar está problemática, se probaron las siguientes dos técnicas:

- **Gaussian Noise.** Consiste en entrenar la red a partir de los datos de entrenamiento, pero agregando cierto ruido gaussiano a los mismos. Es decir, en cada paso de entrenamiento se modifica la entrada sumando al valor de cada caracerística una cantidad producida de forma aleatoria a partir de una distribución normal $N(0, \sigma^2)$.

- **Dropout (ver [5]).** Se basa en entrenar sucesivamente subredes de la red original generadas aleatoriamente, de forma tal que al final del entrenamiento la red original actúe como una reunión de todas estas subredes.

  En cada paso de entrenamiento, y para cada unidad de la red, se sortea con probabilidad $p \in (0, 1)$ si dicha unidad es conservada. En caso de ser, temporalmente, descartada (dropped out), esa unidad y sus respectivas conexiones no son tenidas en cuenta para ese paso de entrenamiento. Las unidades y conexiones que sí se conservan son las que forman la subred que sí es entrenada (ver Figura 7). A continuación, los pesos de esa subred son actualizados mediante backpropagation respecto a la salida esperada (dentro de los patrones de un determinado *batch*), y los pesos corresponientes a unidades descartadas

no son modificados. En la siguiente pasada de entrenamiento, se vuelve a sortear una nueva subred a partir de la original y se repite el procedimiento. Se continua de esta forma hasta completar la cantidad de iteraciones fijadas como entrenamiento.



(a) Red completa (b) Subred en un paso de dropout

Figura 7

Luego en la etapa de test, se clasifica usando la totalidad de la red entrenada, sin descartar ninguna unidad o conexión. Sin embargo, dado que en cada paso se tiene, en media, una proporción de p unidades y conexiones respecto de las que posee la red original, se recomienda al finalizar el entrenamiento multiplicar por p los pesos obtenidos luego del entrenamiento, de forma tal de asegurar que la salida esperada de cada unidad durante el entrenamiento se corresponda con la que se obtendrá durante el test.

De esta forma, la red obtenida hace uso del potencial que proporcionan las 2^n subredes de la red original (con n la cantidad de unidades de la red), manteniendo el número total de parámetros (pesos de las conexiones) en $O(n^2)$.

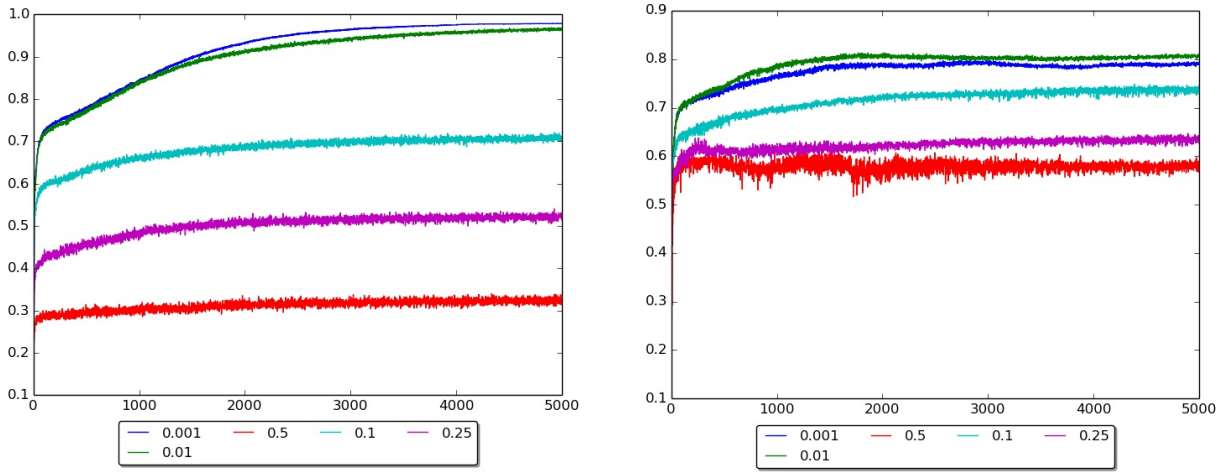
A modo ilustrativo, se presenta en la Tabla 2 los desempeños en datos de entrenamiento y test de dos redes distintas alternando: no usar método de generalización, usar *Gaussian Noise* y usar *Dropout*. La Red 1 está compuesta por una capa oculta de 120 unidades con activación *ReLU* y la Red 2 por dos capas ocultas de 120 y 50 unidades con activaciones *Sigmoide*, ambas con activación *Softmax* en la capa de salida.

	Sin técnica		Con *Gaussian Noise* ($\sigma = 0{,}01$)		Con *Dropout* ($p = 0{,}7$)	
	train (%)	test (%)	train (%)	test (%)	train (%)	test (%)
Red 1	85.66	80.42	84.30	78.45	80.18	78.41
Red 2	93.39	79.11	91.14	79.72	85.67	80.03

Tabla 2

Por otro lado, se estudió el desempeño de estas dos técnicas de generalización con distintos parámetros de σ (desviación en *Gaussian Noise*) y de p (probabilidad de conservación en

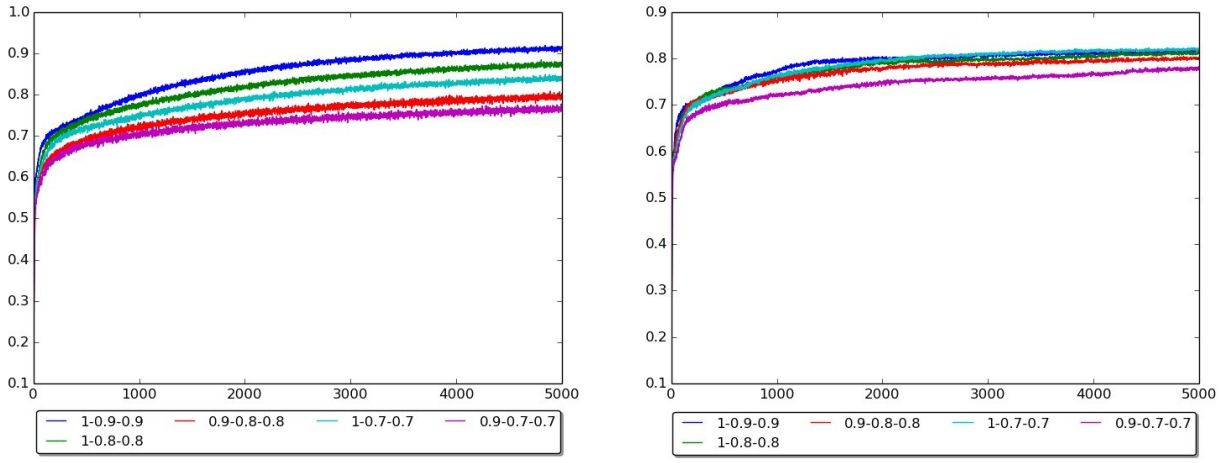
Dropout). Las Figuras 8 y 9 ilustran los resultados obtenidos para una red con dos capas ocultas con las dimensiones y activaciones que arrojaron mejores resultados en secciones anteriores.



(a) Entrenamiento

(b) Test

Figura 8: *Gaussian Noise* con distintos parámetros de σ.



(a) Entrenamiento

(b) Test

Figura 9: *Dropout* con parámetros $p_1 - p_2 - p_3$, es decir, parámetro p_i en la capa i.

A partir de las pruebas, se obtuvieron resultados óptimos con parámetros $\sigma = 0{,}01$ para entrenamientos con *Gaussian Noise* y $p = 0{,}7$ para entrenamientos con *Dropout*.

Si bien ambas técnicas mejoraron el acierto con los datos de tests, al combinarlas no se obtuvo un mejor desempeño.

4. Red Neuronal elegida

A partir de las experimentaciones realizadas, se consideró que la Red Neuronal más adecuada para resolver el problema planteado es la construida de la siguiente manera:

- Una capa de entrada con 51 unidades (una por cada característica no nula en entrenamiento).
- Estandarización de las entradas.
- Dos capas ocultas de 120 unidades la primera y 50 la segunda.
- Una capa de salida con 7 unidades (una por clase).
- *ReLU* como función de activación en las capas ocultas y *Softmax* en la capa de salida.
- *Dropout* con parámetro $p = 0{,}7$ en las dos capas ocultas.
- *Mean Square Error* como función a minimizar durante el entrenamiento.
- *Adam* como método de optimización durante el entrenamiento.
- Inicialización de los pesos al azar con distribución normal $N(0, \frac{1}{20}^2)$.
- Entrenamiento de la red con 70 % del conjunto de entrenamiento (seleccionado al azar), en grupos (*batches*) de 128 patrones consecutivos.
- Criterio de parada: Cuando se alcanzan 300 iteraciones consecutivas sin mejora en la función de error, evaluando la misma sobre el 30 % de datos reservados para validación.

Se entrenaron 10 de estas redes sobre el 70 % de los datos de entrenamiento elegidos randómicamente, testeando el desempeño de las mismas con el 30 % restante. La siguiente tabla resume los resultados de esos tests:

Entren. No.	1	2	3	4	5	6	7	8	9	10
Acierto (%)	81.65	82.48	81.17	82.09	81.91	82.83	80.16	80.69	83.27	81.87

Tabla 3: Desempeño de 10 redes como la elegida, evaluadas en el 30 % de datos separados para validación.

Se obtuvo de esta forma una media de 81,81 % de acierto sobre el conjunto de validación, con una varianza de 0,81 %. Por tanto, se estima que la Red Neuronal seleccionada tendrá un desempeño en el conjunto de test (desconocido a priori) en el entorno de 81,81 %.

5. Resultados

Se particionó randómicamente el conjunto original de test en una proporción de 70:30. Se entrenó la red elegida con el conjunto grande (70 %) de acuerdo al criterio de parada respecto al conjunto chico (30 %). Con la red ya entrenada, se clasificó el conjunto final de test obteniéndose un acierto de 80,76 %.

6. Conclusiones

Durante el presente trabajo, el problema planteado fue abordado, en una primera instancia, evaluando el desempeño de distintos clasificadores mediante validación cruzada sobre el conjunto de entrenamiento dado.

Posteriormente, se profundizó el abordaje del problema con la técnica de Redes Neuronales de tipo perceptrón multicapa. Se experimentaron diversas opciones para la gran mayoría de los componentes que hacen a una Red Neuronal de este tipo, logrando mejorar los resultados preliminares obtenidos con esta técnica dentro del software Weka. Para esto se evaluaron diversas configuraciones de redes en forma sistemática y se pusieron en práctica satisfactoriamente nóveles métodos que asceleran el proceso de entrenamiento de las redes (*Adam*), o que logran, hasta cierto punto, atenuar problemas como el del sobreajuste (*Dropout*).

Referencias

[1] Yann LeCun, Leon Bottou, Genevieve B. Orr y Klaus-Robert Muller, *Efficient BackProp*, 1998.

[2] Tieleman, T. and Hinton, G. Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. Technical report, 2012.

[3] John Duchi, Elad Hazan, Yoram Singer *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization* `http://www.magicbroom.info/Papers/DuchiHaSi10.pdf`

[4] Diederik P. Kingma, Jimmy Lei Ba, *Adam: A method for stochastic optimization.* `http://arxiv.org/pdf/1412.6980v8.pdf`

[5] Nitish Srivastava, Georey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overtting*, Journal of Machine Learning Research, 15, 2014.
`https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf`

[6] Duda, Hart and Stork, *Pattern Classification*, John Wiley & Sons (ISBN-10- 0471056693)-2001.