THE WEKA MULTILAYER PERCEPTRON CLASSIFIER

Daniel I. MORARIU[1], Radu G. CREŢULESCU[1], Macarie BREAZU[1]

[1]"Lucian Blaga" University of Sibiu, Engineering Faculty, Computer Science and Electrical and Electronics Engineering Department

Abstract

Automatic document classification is a must when dealing with large collection of documents. WEKA, and especially Weka Knowledge Flow Environment, is a state-of-the-art tool for developing classification applications, even with no programming abilities. We continue our WEKA project presented in a previous paper but changing the classification step, now using the Multilayer Perceptron Classifier. The used dataset is one based on documents from the Reuters Corpus and with vector space model representation, the number of features being reduced by using the InformationGain method. The theoretical bases for Multilayer Perceptron neural networks are presented, both for the architecture and for the backpropagation learning algorithm. In order to evaluate the performance of the Multilayer Perceptron Classifier experiments were done, first with the default network architecture. Results are presented and prove valuable, but for a large number of features the performances decrease. In order to improve the obtained results we test different fine-tuned architectures by changing the number of neurons in the hidden layer. Therefore, the Weka Multilayer Perceptron Classifier is a classifier that deserves attention, but mainly when time requirements are not important at all..

Keywords: Document classification, WEKA framework, Multilayer Perceptron Classifier

1. Introduction

The large amount of data, which is generated by the communication process, represents important information that is accumulated daily and which is stored in form of text documents, databases etc. Retrieving of this data is not simple and therefore data mining techniques were developed for extracting information and knowledge. These are represented in patterns or concepts that are sometimes not obvious.

A complete retrieved process starts with the document representation part, where the data is extracted from flat files and represented into a manner that can be understand by the machine. There are several representation methods each offering the possibility to keep more or less syntactic and semantics of documents. The process continues with the step of selecting the most relevant features. In the text mining process, a very large number of features are obtained, and too much of them can disturb the learning process. Therefore, this step is very important, and proposes to select the most relevant features. Only after this step the learning algorithms are applied in order to get the results.[3,6,9]

In this paper we focus on the learning step and propose the use of a Multilayer Perceptron Classifier. The dataset used for testing this classifier is the same as in the previous article ([2] and [8]) to make a better comparison between classifiers. In [2] we evaluate the Naïve Bayes classifier and in [8] we evaluate the Support Vector Machine Classifier. The idea of this paper is to evaluate a learning algorithm based on Multilayer Perceptron for different number of features extracted form text documents. We evaluate the classification performance from accuracy, precision and recall point of view. The experiments have been described and implemented using the Weka framework [10].
The section 2 presents some theoretical aspects important to the paper regarding multilayer perceptron. In the section 3 we present the Weka framework and the components used. The section 4 presents the experimental results performed using the WEKA framework. Section 5 contain the conclusions and further work of this paper.Editorial Board

1 Theoretical Aspects

The model of the artificial neuron was proposed by McCulloch and Pitts in the 40's and was generalized later in many ways. The most popular approach is:

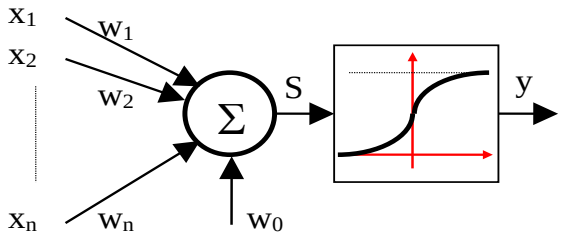

Figure.1 Artificial neuron

The neuron computes the weighted sum of *n* inputs, adds a threshold value and then applys an activation function to the result in order to compute the output.

$$S = \sum_{i=1}^{n} x_i w_i + w_0 \quad y = f(S) \tag{1}$$

As activation function the most used is the sigmoid function, defined as:

$$sigmoid(x) = \frac{1}{1+e^{-x}} \tag{2}$$

The nonlinearity of that function is essential for the power of the neural networks model. Also, the function scales the output to the [0-1] range.
The previous described **perceptron** can classify only linear separable input vectors (XOR being the classic counterexample). This was proven since 1969 by Minsky and Papert and has reduced the interest of researchers for neural networks. To solve the problem **multilayer perceptron** had to be used but it was not known how to update the weights of hidden (intermediate) layers. The updating rule for the weights (briefly described below) was discovered only in the late 80's and was the basis of the boom of neural networks field.

The most popular architecture for neural networks is the multilayer perceptron where each neuron is connected to all the neurons from the previous layer. The only exception is the first layer whose units only repeat their inputs. In the following figure we exemplify the most common approach with one single hidden layer (proven theoretical to be sufficient).

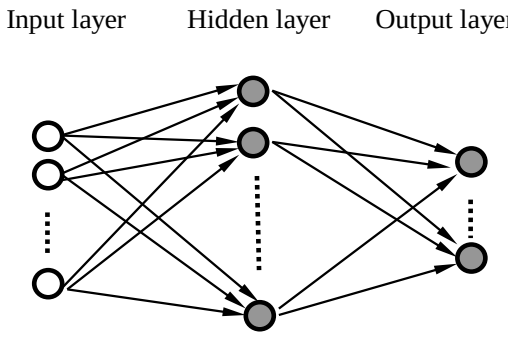

Figure 2 Architecture for neural networks is the multilayer perceptron

In the **forward step** equation (1) is applied for each neuron, first for the hidden layer and then for the output layer (therefore the "**feed-forward**" name) in order to obtain the output value. Being in a case of supervised learning we have also the desired output for each input vector. Therefore, the representation error *E* that appears can also be computed (defined as common Euclidian distance between obtained output and desired output vectors).
The learning rule falls in the category of "**error-correction rules**". The most general rule to update a weight w (from **any** layer!) is:

$$\Delta w = -\eta \frac{\partial E}{\partial w} \tag{3}$$

where *E* is the error (as a function of *w*) and η is the learning rate. The evolution is opposite to the gradient of the error, therefore decreasing the error. Even if not plausible from the biological point of view, it looks like the error propagates back through the network (in the **backward step**) and updates the weights, hence the "**backpropagation**" name of the learning algorithm. The forward and the backward steps are repeated until the error is sufficiently reduced. Complete formulas for each weight update can be found in [1]. Sometimes, in order to increase the chance to find global minima, a (selectable) fraction of the Δw from the previous step of the learning is added to the Δw for the current step (the added part being known as the **momentum** term).

2 WEKA framework

2.1 General information

In developing autonomous document classification systems for text documents, a series of individual subsystems are included. These subsystems must be harmonized together to produce a good and performant classification system. For designing these systems, there are many frameworks that help us, using a small set of information, to optimize the flow between these subsystems. After this optimization, for a real

problem solving, we need to implement those subsystems as they were configured in the framework. One of this framework that contains a collection of machine learning algorithms is WEKA (Waikato Environment for Knowledge Analysis [9]). This framework has been developed to be able to design a series of data mining solutions using subsystems that are already developed and made available. The WEKA framework offers a lot of algorithms that are written in java and are available open source for integrate in your projects. But, for avoiding the programing part, WEKA offers also a framework that permits you to describe your data mining application as a flow of actions and to evaluate it, without the need to write code. The only thing to be done is to write code to transform your specific dataset into a format that is accepted by WEKA.
WEKA offers four different options for implementing your data mining process. The WEKA ***Knowledge Explorer*** is an easy to use framework with a graphical user interface that offers all the facilities of WEKA package. Another framework is Weka ***Experiment Environment*** that permits you to create, run and modify an experiment in a simple manner. The experiment can be described into a text file and tested with the WEKA framework. WEKA ***KnowledgeFlow Environment*** permits you to describe your experiment as a flow of steps with some visual connections between them. The WEKA ***Workbench*** contains a lot of state of the art data preprocessing and machine learning algorithms. In this framework the user can quickly try out existing machine learning methods on new datasets in a very flexible way.

2.2 The flowchart of the system

In our experiment we use WEKA ***KnowledgeFlow Environment*** and the flowchart for our experiment if presented in the next figure.

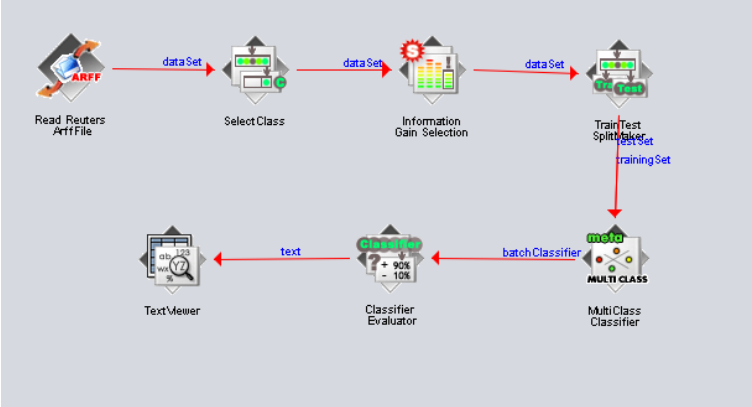

Figure 3 The Multilayer Perceptron Classifier flowchart

In this flowchart we change only the Classifier so that a lot of components used in this flowchart were already presented in previous articles [2] and [8]. Here we present those components only briefly and more detailed the MultiClass Classifier Component.
The ArffLoader Component is used to load our data file that contains the entire dataset, both the training part and testing part. The dataset is saved in one file in the arff format that contain the vector representation of all 7051 Reuters files [7]. Each document has 7000 different features-attributes.
After all 7000 attributes for each document vector, we have a special attribute that represents the class where the document belongs. This class is the expected class in

the dataset (because we talk about a supervised learning). In the presented experiments, we have considered that our documents belong to the class (and labeled with "yes") or not in the class (and labeled with "no"). Thus, we have considered a binary classification.
In all experiments we evaluate the classification accuracy for different number of features between 200 to 5500 features from a total of 7000 features. For selecting the most relevant attributes we use the *AttributeSelection* component. This component is a supervised attribute filter that can be used to select the desired number of attributes. It is very flexible and allows various search and evaluation methods to be combined. For attribute selection we have chosen the Information Gain Attribute Evaluation method. In our experiments, we have used the default value for the threshold and have changed the *numToSelect* as desired in the range 200-5500 as presented in the experimental results section.
After selecting the features, we use *TrainTestSplitMaker* component that permits us to split randomly our dataset into a training part and a testing part. In the configuration, we specificity that 70% of data will be used for training and the rest of 30% to be used for testing.
In order to evaluate the classification performance, we use the *ClassifierPerformanceEvaluator* component that is designed to evaluate classifier results. The WEKA has a lot of evaluation metrics already implemented, as *accuracy*, *precision*, *recall*, *f-measure*, *TrueRate*, *NegativeRate* [9]. In our experiments we use only precision, recall, accuracy and f-measure [12].
For visualizing the results, the WEKA propose a lot components and we chose *TextViewer* component that permits to write the classification results into a text file.

2.3 Multiclass Classifier

The WEKA framework contains a lot of learning algorithms, as classifier, clustering and association algorithms. The classifier algorithms have a specific tab with the *Classifier* name where a lot of algorithms from different categories (as multilayer perceptron, *Bayes*, *rules*, *trees*, *lazy* and more) can be found [4, 5]. WEKA has also a *Clusterers* tab with learning algorithms as *EM*, *Hierarchical*, *Simple KMeans* and more. For our experiments, we use a classifier algorithm because we have a dataset that is already classified. We chose the Multilayer perceptron classifier component.
The MultilayerPerceptron component is in the tab weka.classifiers.functions and has many parameters for configuration. In the following, we describe briefly these characteristics.
We chose the option SYNOPSIS that represents a classifier that uses backpropagation as learning method to classify instances. This network can be built by hand, created by an algorithm or both. The nodes in this network are all sigmoid (except for when the class is numeric in which case the output nodes become linear units without threshold).
The parameters that can be used to configure this component are[10,11]:

- *seed* - used to initialize the random number generator. Random numbers are used for setting the initial weights of the connections between nodes, and for shuffling the training data.
- *momentum* – is the value that is applied to the weights during updating.

- *nominalToBinaryFilter* – represents the filter that will preprocess the instances. This could help to improve performance if there are nominal attributes in the dataset.
- *hiddenLayers* - This defines the number hidden layers of the neural network and the numbers of neurons from each layer. This is a list of positive integer numbers, one for each hidden layer, comma separated. To specify no hidden layers you need to put a single 0 here, and this will only be used if auto build is set. There are also wildcard values 'a' = (attributes + classes) / 2, 'i' = attributes, 'o' = classes, 't' = attributes + classes. We have used the 'a' wildcard.
- *validationThreshold* - Used to terminate the learning process. The value here dictates how many times in a row the validation set error can get worse before training is terminated.
- *GUI* - Brings up a graphic user interface. This will allow the pausing and altering of the neural network during training. Can add a node, create new connections between nodes., remove a connection or a node. If this option is activated then the network is automatically paused at the beginning and the user can reconfigure the network. Once the network configuration is done, it will pause again and either wait to be accepted or trained more. If the GUI is not set the network will not require any interaction.
- *normalizeAttributes* - This will normalize the attributes. This could help to improve the performance of the network. This is not reliant when we have numeric classes. This will also normalize nominal attributes as well (after they have been run through the nominal to binary filter if that is in use) so that the nominal values are between -1 and 1
- *numDecimalPlaces* - The number of decimal places to be used for the output of numbers in the model.
- *batchSize* - The preferred number of vector instances kept in cache once if batch prediction is being performed. More or fewer instances may be provided, but this gives implementations a chance to specify a preferred batch size.
- *decay* - This will cause the learning rate to decrease. This will divide the starting learning rate by the epoch number, to determine what the current learning rate should be. This may help to stop the network from diverging from the target output, as well as improve general performance. Note that the decaying learning rate will not be shown in the GUI, only the original learning rate. If the learning rate is changed in the GUI, this is treated as the starting learning rate.
- *validationSetSize* - The percentage size of the validation set. The training will continue until the error on the validation set has been consistently getting worse, or if the training time is reached.
- *trainingTime* - The number of epochs to train through. If the validation set is non-zero then it can terminate the network learning.
- *debug* - If set to true, classifier may output additional info to the console.
- *autoBuild* - Adds and connects up hidden layers in the network.
- *normalizeNumericClass* - This will normalize the class if it is numeric. This could help improve performance of the network, It normalizes the class to the range [-1,1]. Note that this is done only internally, the output will be scaled back to the original range.
- *learningRate* -- The amount the weights are updated.
- *doNotCheckCapabilities* -- If set, classifier capabilities are not checked before classifier is built (Use with caution to reduce runtime).

- *reset* - This will allow the network to reset with a lower learning rate. If the network diverges from the answer this will automatically reset the network with a lower learning rate and begin training again. This option is only available if the GUI is not set. Note that if the network diverges but isn't allowed to reset it will fail the training process and return an error message.

If the GUI is activated for an input vector having 10 attributes and 6 neurons on the hidden layer the network looks like the following figure.

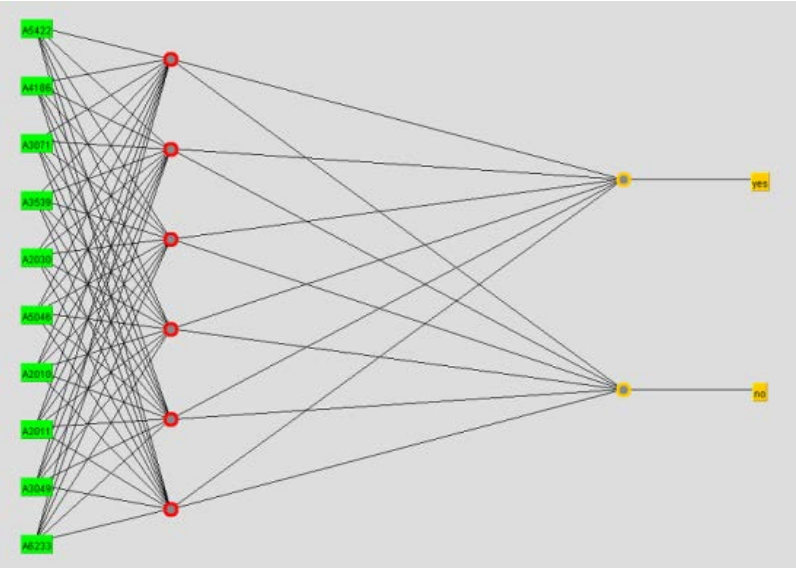

Fig. 4. GUI – representation on 6 neurons

3 Experimental Results

The experimental results have been performed in the WEKA framework as described previously. For the *AttributSelection* component, we have used a parameter that represents the number of features that we want to be retained. In Table 1 we present the results obtained for a number of features between 200 and 1000 by the neural learning algorithm. We have evaluated the following measures: Precision, Recall, F-measure and Accuracy. For comparison reasons, we present also the results taken from our previous papers [8] and [2]. For the Multilayer Perceptron a new parameter appears: the number of neuron in the hidden layer. We chose to have only one hidden layer and in majority cases we keep the default number of neurons in the hidden layer. Only for 500 and 1000 features we make tests for a different number of neurons on hidden layer (smaller and greater) to see if the learning quality increases or not significantly.

No of features	Multilayer Perceptron					Naïve Bayes				SVM
	No. of neurons on hidden layer	Precision	Recall	F-Measure	Accuracy (%)	Precision	Recall	F-Measure	Accuracy (%)	Accuracy (%)
200	101def	**0.881**	**0.855**	**0.853**	**85.50**	0.839	0.830	0.829	82.98	92.45
300	151def	**0.903**	**0.888**	**0.887**	**88.80**	0.846	0.840	0.839	84.02	92.87
400	201def	**0.872**	0.833	0.829	83.32	0.846	**0.842**	**0.842**	**84.23**	93.37
500	150	0.751	0.493	0.328	49.29	**0.838**	**0.836**	**0.835**	**83.56**	92.99
	251def	0.772	0.573	0.484	57.33	**0.838**	**0.836**	**0.835**	**83.56**	92.99
	550	0.750	0.510	0.345	50.99	**0.838**	**0.836**	**0.835**	**83.56**	92.99
600	301def	0.750	0.510	0.345	50.99	**0.836**	**0.834**	**0.834**	**83.40**	93.12
1000	250	0.260	0.509	0.344	50.95	**0.843**	**0.842**	**0.842**	**84.23**	93.16
	501def	0.241	0.491	0.323	49.05	**0.843**	**0.842**	**0.842**	**84.23**	93.16
	1024	0.750	0.510	0.345	50.99	**0.843**	**0.842**	**0.842**	**84.23**	93.16

Table 1. Experimental Results

From the experimental results, we notice that this type of classifier algorithm works very well with a small number of features, but when the number of features increases, the training time increase also or, worse, the network cannot learn. So that for 200 and 300 features the network returns results better than Naïve Bayes (close to SVM) but, for more features, the learning quality decreases significantly.

4 Conclusions

In this paper, we use Information Gain as feature selection method (that was proven in one of our previous papers to be the best) and we evaluate an algorithm – Multilayer Perceptron with one hidden layer. This tested classifier is not so fast as Naïve Bayes but much faster comparatively with Support Vector Machine.

As we might expect experimental results are not of the same quality comparatively with SVM (but are close for a small number of features) but are better comparatively with Naïve Bayes results.

As further work we propose to classify large text data sets (the complete Reuters dataset) in order to see the behavior of Information Gain feature selection method and the Multilayer Perceptron in an industrial text classification problem. We try to make the representation and classification into two steps, in first step make a pre-classification of all documents, obtain fewer representative samples and after that we'll use only the obtained samples as input vectors for an information retrieval system.

References

[1] Breazu, M. *Tehnici fractale şi neuronale în compresia de imagini,* Editura Universitatii „Lucian Blaga" din Sibiu, ISBN 973-739-251-5, Sibiu, 2006.

[2] Cretulescu, R., Morariu, D., Breazu, M. - Using WEKA framework in document classification, The 7th International conference on Information Science and Information Literacy, ISSN 2067-9882, April 2016, Sibiu.

[3] Han, J., Kamber, M., - *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, 2001;
[4] Manning, C., - An Introduction to Information Retrieval, Cambridge University Press, 2009
[5] Mitchell, T., M. *Machine Learning*, The McGrow-Hill Companies, 1997
[6] Mitkov R., *The Oxford Handbook of Computational Linguistics*, Oxford University Press, 2005;
[7] Misha Wolf and Charles Wicksteed – *Reuters Corpus:* http://trec.nist.gov/data/reuters/reuters.html, accessed in 03.2016
[8] Morariu D., Cretulescu R., Breazu, M. - *Feature Selection in Document Classification*, The fourth International Conference in Romania of Information Science and Information Literacy, ISSN-L 2247-0255, April 2013, Sibiu
[9] Witten, I. H., , Hall, E. F., Pal, C.J., *Data Mining – Practical Machine Learning Tools and Techniques with Java Implementation*, Morgan Koufmann Press, 2000
[10] http://www.cs.waikato.ac.nz/ml/weka/, accessed in 03.2016
[11] http://www.cs.waikato.ac.nz/ml/weka/documentation.html, accessed 03.2016
[12] https://en.wikipedia.org/wiki/Precision_and_recall, accessed 03.2016