Sparse Fuzzy Techniques Improve Machine Learning

Reinaldo Sanchez[1], Christian Servin[1,3], and Miguel Argaez[1,2]
[1]Computational Science Program and [2]Department of Mathematical Sciences
University of Texas at El Paso, 500 W. University, El Paso, TX 79968, USA
[3]Information Technology Department, El Paso Community College
919 Hunter, El Paso TX 79915, USA
reinaldosanar@gmail.com, christians@utep.edu, margaez@utep.edu

***Abstract*—On the example of diagnosing cancer based on the microarray gene expression data, we show that fuzzy-technique description of imprecise knowledge can improve the efficiency of the existing machine learning algorithms. Specifically, we show that the fuzzy-technique description leads to a formulation of the learning problem as a problem of sparse optimization, and we use ℓ_1-techniques to solve the resulting optimization problem.**

I. Formulation of the Problem

Machine learning: a typical problem. In machine learning, we know how to classify several known objects, and we want to learn how to classify new objects.

For example, in a biomedical application, we have microarray data corresponding to healthy cells and microarray data corresponding to different types of tumors. Based on these samples, we would like to be able, given a microarray data, to decide whether we are dealing with a healthy tissue or with a tumor, and if it is a tumor, what type of cancer does the patient have.

In general, each object is characterized by the results

$$x = (x_1, \ldots, x_n)$$

of measuring several (n) different quantities. So, in mathematical terms, machine learning can be described as a following problem (see, e.g., [1]):

- we have K possible labels $1, \ldots, K$ describing different classes;
- we have several vectors $x(j) \in R^n$, $j = 1, \ldots, N$, each of which is labeled by an integer $k(j)$ ranging from 1 to K; vectors labeled as beloning to the k-th class will be also denoted by
$$x(k,1), \ldots, x(k, N_k);$$
- we want to use these vectors to assign, to each new vector $x \in R^n$, a value $k \in \{1, \ldots, K\}$.

Machine learning: original idea. The first machine learning algorithms were based on the assumption that for each class C_k, the set of the vectors x corresponding to this class is *convex*, i.e., that for every two vectors $x, x' \in C_k$, and for every number $\alpha \in (0,1)$, their *convex combination*

$$\alpha \cdot x + (1 - \alpha) \cdot x'$$

also belongs to the class C_k.

It is known that if different classes $C_1, \ldots, C_K$ are convex, then we can separate them by using linear separators; see, e.g., [6]. For example, if we only have two classes C_1 and C_2, then there exists a linear function

$$f(x) = c_0 + \sum_{i=1}^{n} c_i \cdot x_i$$

and a threshold value y_0 such that:

- for all vectors $x \in C_1$, we have $f(x) < y_0$, while
- for all vectors $x \in C_2$, we have $f(x) > y_0$.

Thus, if we know that an object characterized by a vector x belongs to one of these two classes, and we want to decide to which of these classes it belongs, we can compute the value of the linear function $f(x)$, and then:

- if $f(x) < y_0$, conclude that x belongs to the class C_1, and
- if $f(x) > y_0$, conclude that x belongs to the class C_2.

If we have more than two classes, then, to classify an object, we can use linear functions separating different pairs of classes.

Machine learning: current development. In practice, the classes C_k are not convex. As a result, it is often impossible to use simple linear separators $f(x)$ to separate different classes, we need *nonlinear* separating functions.

The first such separating functions came from an observation that in the biological neurons – cells that enable us to classify objects – the dependence of the output on the input is nonlinear. The resulting Artificial Neural Networks simulating this nonlinearity turned out to be an efficient machine learning tool.

Even more efficient algorithms became possible when researchers realized that a general nonlinear separating function $f(x_1, \ldots, x_n)$ can be represented, e.g., by its Taylor series

$$f(x_1, \ldots, x_n) = c_0 + \sum_{i=1}^{n} c_i \cdot x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_i \cdot x_j + \ldots$$

This expression becomes linear if, in addition to the original values $x_1, \ldots, x_n$, we also add their combinations such as $x_i \cdot x_j$. The corresponding *Support Vector Machine* (SVM) techniques [1], [10] are, at present, the most efficient in machine learning. For example, SVM is implemented in the GEMS-SVM software which is used to automatically diagnose cancer based on the microarray gene expression data [9].

Remaining problem. While the SVM methods lead to efficient classification results, these results are not perfect. If we:

- divide the original samples into a training set and a training set,
- train an SVM method on the training set, and then
- test the resulting classification on a testing set,

then we get, depending on the type of tumor, 90 to 100% correct classifications. 90% is impressive, but it still means that up to 10% of all the patients are misclassified. How can we improve this classification?

Our idea. As we have mentioned, linear algorithms are based on an assumption that all the classes C_k are convex. Since these classes are not exactly convex, SVM techniques abandon the efficient linear separation algorithms and use less efficient general nonlinear techniques.

In reality, while the classes are *not exactly convex*, they are *somewhat* convex, in the sense that for many vectors x and x' from each class C_k and for many values α, the convex combination $\alpha \cdot x + (1-\alpha) \cdot x'$ still belongs to C_k.

In this paper, we use fuzzy techniques to formalize this imprecise idea of "somewhat" convexity, and we show that the resulting machine learning algorithm indeed improves the efficiency.

II. Main Idea: From Formalization of "Somewhat" Convexity to Sparse Optimization

Need to use degrees. The usual ("crisp") description of convexity is that if the two vectors x, x' belong to the set C_k of all the vectors corresponding to the k-th class, then their convex combination

$$\alpha \cdot x + (1-\alpha) \cdot x'$$

also belongs to this class C_k.

"Somewhat" convexity means that if $x, x' \in C_k$, then we can only conclude with a certain *degree of confidence* that the object corresponding to the vector

$$\alpha \cdot x + (1-\alpha) \cdot x'$$

belongs to the k-th class. So, to get an adequate description of our knowledge about the k-th class, we need to assign, to each possible vector x, a degree $\mu_k(x)$ to which we are confident that an object corresponding to this vector x belongs to this class. The corresponding function $x \to \mu_k(x) \in [0,1]$ constitutes a *fuzzy set* [4], [5], [11].

For each vector x, we can then assign the corresponding object to the class k for which we are most confident that x belongs to this class, i.e., to the class k for which the degree of confidence $\mu_k(x)$ is the largest possible.

Using "somewhat" convexity to estimate degrees of confidence. For each class k, we know that several "sample" vectors that for sure belong to this class – namely, the vectors

$$x(k,1), \ldots, x(k, N_k)$$

which have been originally labeled as belonging to the j-th class.

We also know that a convex combination

$$x'' \stackrel{\text{def}}{=} \alpha \cdot x + (1-\alpha) \cdot x'$$

of two vectors x and x' belongs to the k-th class if both x and x' belong to the class, and if the convexity rule holds for this situation. Thus, the degree $\mu_k(x'')$ to which x'' belongs to the k-th class is greater than or equal to the degree

$$\mu((x \in C_k) \,\&\, (x' \in C_k) \,\&\, \text{rule holds})$$

with which all these three conditions hold. A simple way to estimate the degree to which all three conditions hold is to take a product of the degree of the component statements:

- the product corresponds to the case when degrees are (subjective) probabilities and all three events are reasonably independent; see, e.g., [8];
- the product is also one of the simplest and most widely used ways of combining degrees of confidence in fuzzy applications.

So, if we denote, by r, our degree of confidence in the convexity rule, we conclude that

$$\mu_k(\alpha \cdot x + (1-\alpha) \cdot x') \geq r \cdot \mu_k(x) \cdot \mu_k(x').$$

This means that for vectors x'' which are convex combinations of two sample vectors, the degree $\mu_k(x'')$ is great than or equal to r.

A vector x''' which is a convex combinations of *three* sample vectors x, x', and x'' can be represented as a convex combination of x and a convex combination y of x' and x''; since $\mu_k(y) \geq r$, we thus get

$$\mu_k(x''') \geq r \cdot \mu_k(x) \cdot \mu_k(y) \geq r \cdot 1 \cdot r = r^2.$$

Similarly, for a vector y which is a convex combination of four sample vectors, we get $\mu_k(y) \geq r^3$. In general, for a vector y which is a combination of v sample vectors, we get $\mu_k(y) \geq r^{v-1}$. In other words, for a vector

$$y = \sum_{j=1}^{N_k} \alpha_j \cdot x(k,j),$$

where $\alpha_j \geq 0$ and $\sum\limits_{j=1}^{N_k} \alpha_j = 1$, we conclude that

$$\mu_k(y) \geq r^{\|\alpha\|_0 - 1},$$

where the ℓ_0-norm $\|\alpha\|_0$ of a vector $\alpha = (\alpha_1, \ldots, \alpha_{N_k})$ is defined as the number of non-zero components of this vector:

$$\|\alpha\| \stackrel{\text{def}}{=} \#\{i : \alpha_j \neq 0\}.$$

Using closeness. Another natural idea is that if a vector x is *close* to a vector y corresponding to the object from the k-th class, then it is reasonable to conclude (with some degree of confidence) that the object corresponding to x also belongs to the same class. The smaller the distance

$$\|x-y\|_2 \stackrel{\text{def}}{=} \sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

between the vectors x and y, the larger our degree of certainty that x belongs to this class.

In this paper, we assume that this degree of certainty is described by a Gaussian expression $\exp\left(-\frac{\|x-y\|_2^2}{\sigma^2}\right)$ for an appropriate value σ. Similarly to selecting a product, we selected the Gaussian formula for two reasons:

- the Gaussian distribution is ubiquitous in probability theory, since due to the Central Limit Theorem, every time when the effect is caused by a large number of small independent factors, the distribution is close to Gaussian; see, e.g., [8];
- in fuzzy logic, Gaussian membership functions have been successfully used in many practical applications to describe closeness.

As a result, for every two vectors x and y, we have

$$\mu_k(x) \ge \mu_k(y)\cdot\exp\left(-\frac{\|x-y\|_2^2}{\sigma^2}\right).$$

Resulting formula for the degree of confidence. By combining the two formulas describing “somewhat” convexity and closeness, we conclude that for every vector x and for every vector α, we have

$$\mu_k(x) \ge \exp\left(-\frac{\left\|x-\sum\limits_{j=1}^{N_k}\alpha_j\cdot x(k,j)\right\|_2^2}{\sigma^2}\right)\cdot r^{\|\alpha\|_0-1}.$$

This is true for every α, so we can conclude that the desired degree of confidence is larger than or equal than the largest of the right-hand sides:

$$\mu_k(x) \ge \max_{\alpha}\exp\left(-\frac{\left\|x-\sum\limits_{j=1}^{N_k}\alpha_j\cdot x(k,j)\right\|_2^2}{\sigma^2}\right)\cdot r^{\|\alpha\|_0-1}.$$

This right-hand side is all we can state about the vector x, so we can thus conclude that

$$\mu_k(x) = \max_{\alpha}\exp\left(-\frac{\left\|x-\sum\limits_{j=1}^{N_k}\alpha_j\cdot x(k,j)\right\|_2^2}{\sigma^2}\right)\cdot r^{\|\alpha\|_0-1}.$$

Thus, to classify a vector x, we need to compute the values of the right-hand side corresponding to different classes k, and select the class k for which the value $\mu_k(x)$ is the largest.

Formula simplified. Maximizing the value $\mu_k(x)$ is equivalent to minimizing a simpler expression

$$d_k(x) \stackrel{\text{def}}{=} -\ln(\mu_k(x))$$

which can be described as follows:

$$d_k(x) = \min_{\alpha}\left(\frac{\left\|x-\sum\limits_{j=1}^{N_k}\alpha_j\cdot x(k,j)\right\|_2^2}{\sigma^2} + (\|\alpha\|_0-1)\cdot R\right),$$

where $R \stackrel{\text{def}}{=} -\ln(r)$. To use this formula, for each k, we need to minimize the expression

$$\frac{\left\|x-\sum\limits_{j=1}^{N_k}\alpha_j\cdot x(k,j)\right\|_2^2}{\sigma^2} + (\|\alpha\|_0-1)\cdot R$$

with respect to vectors α. By adding a constant R to this expression and dividing the resulting objective function by R, we get an equivalent problem of optimizing the objective function

$$\mathcal{C}\cdot\left\|x-\sum_{j=1}^{N_k}\alpha_j\cdot x(k,j)\right\|_2^2 + \|\alpha\|_0,$$

where

$$\mathcal{C} \stackrel{\text{def}}{=} \frac{1}{R\cdot\sigma^2}.$$

III. From the Main Idea to Its Efficient Algorithmic Implementation

The above optimization problem is hard to solve exactly. The main algorithmic problem with the above optimization is that optimization problems involving the ℓ_0-norm $\|\alpha\|_0$ are, in general, NP-hard; see, e.g., [3]. This means, crudely speaking, that unless P = NP (which most computer scientists believe to be impossible), there is no hope to have an efficient algorithm for exactly solving all particular cases of such an optimization problem.

ℓ_1-optimization as a good approximation to ℓ_0-optimization problems. In terms of the Lagrange multiplier method, the above optimization problem is equivalent to minimizing the ℓ_0-norm $\|\alpha\|_0$ under the constraint

$$\left\|x-\sum_{j=1}^{N_k}\alpha_j\cdot x(k,j)\right\|_2 \le C,$$

for an appropriate constant C. One of the reasons why this problem is NP-hard is that while the set of all the vectors α which satisfy this constraint is convex, the ℓ_0-norm $\|\alpha\|_0$ is not convex. To make the problem computationally efficient, researchers proposed to approximate it by a convex objective function, namely, approximate it by an ℓ_1-norm

$$\|\alpha\|_1 \stackrel{\text{def}}{=} \sum_{j=1}^{N_k} |\alpha_j|.$$

It turns out that not only we have a good approximation in many practical applications, but in many cases, ℓ_1-optimization is *equivalent* to ℓ_0-optimization; see, e.g., [2].

We therefore propose to use the replace each original NP-hard optimization problem with a feasible convex optimization problem of minimizing

$$C' \cdot \left\| x - \sum_{j=1}^{N_k} \alpha_j \cdot x(k,j) \right\|_2^2 + \|\alpha\|_1$$

for an appropriate constant C'.

Taking the specific problem into account. The above formulation works for any "somewhat" convex case.

For microarray analysis, there is a specific property – that the actual values of the vector x depend on the efficiency of the microarray technique. In other words, with a less efficient technique, we will get $\lambda \cdot x$ for some constant λ. From this viewpoint, it is reasonable to use not just *convex* combinations, but arbitrary *linear* combinations of the original vectors $x(k,j)$.

Towards even more efficient computations. While ℓ_1-optimization is efficient, it still takes a large amount of computation times.

In our formulation, we need to repeat this optimization as many times as there are classes; this repetition further increases the computation time.

To decrease computation time, we propose the following idea:

- instead of trying to represent the vector x as a linear combination of vectors from each class,
- let us look for a representation of x as a linear combination of *all* sample vectors, from all classes.

In other words, instead of solving K different optimization problems, let us solve a single problem of minimizing the expression

$$C' \cdot \left\| x - \sum_{j=1}^{N} \alpha_j \cdot x(j) \right\|_2^2 + \|\alpha\|_1.$$

Then, for each class k, we only take the components belonging to this class, and select a class for which the resulting linear combination is the closest to the original vector x, i.e., for which the easy-to-compute distance

$$\left\| x - \sum_{j:k(j)=k} \alpha_j \cdot x(j) \right\|_2$$

is the smallest possible.

Observation. Interestingly, this time-saving idea not only increased the efficiency of our method, it also improve the quality of classification.

We think that this improvement is related to the fact that all the data contain measurement noise. On each computation step, we process noisy data and the results combine noise from different inputs of this step – and thus, get noisier and noisier with each computation step. From this viewpoint, the longer computations, the more noise we add.

So, while our approximate method does not lead to a perfect minimum in the original optimization problem, it saves on the noise, and these savings overwhelm the errors introduced by replacing the original K times repeated optimizations with a single "approximate" ones.

IV. Results

Detailed results are presented in [7]. (It should be mentioned that the paper [7] does not include any justification of our proposed algorithm; such a justification is presented here for the first time.) The results are good; for example:

- for brain tumor, the probability of correct identification increased from 90% for the best known SVM techniques to 91% for our method;
- for prostate tumor, the probability similarly increased from 93% to 94%.

In some cases, the success probability of our algorithm is slightly lower than for the SVM techniques. However, it should be mentioned that the good SVM results are partly due to the fact that to make SVM efficient, we need to select appropriate nonlinear functions, and there are known algorithms for selecting these functions. If we select arbitrary functions, we usually get not-so-good results; this selection must be expertly tuned to the problem. Hence, the very fact that SVM results (obtained after trying several different sets of nonlinear functions are selecting the most appropriate set) are good does not necessarily mean that we will get similarly good results in other cases.

In contrast, our sparse method has only one parameter to tune – the parameter C'. From this viewpoint, our technique is much less subjective, much more reliable – and leads to similar or even better classification results.

Acknowledgments

The authors are thankful to the anonymous referees for valuable suggestions.

References

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2006.

[2] E. Candès, J. Romberg, and T. Tao, "Robust Uncertainty Principles: Exact Signal Reconstruction from Highly Incomplete Frequency Information", *IEEE Transactions on Information Theory*, 2006, Vol. 52, pp. 489–509.

[3] M. Elad, *Sparse and Redundant Representations*, Springer Verlag, 2010.

[4] G. J. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.

[5] H. T. Nguyen and E. A. Walker, *First Course In Fuzzy Logic*, CRC Press, Boca Raton, Florida, 2006.

[6] R. T. Rockafeller, *Convex Analysis*, Princeton University Press, Princeton, New Jersey, 1970.

[7] R. Sanchez, M. Argaez, and P. Guillen, “Sparse Representation via l^1-minimization for Underdetermined Systems in Classification of Tumors with Gene Expression Data”, *Proceedings of the IEEE 33rd Annual International Conference of the Engineering in Medicine and Biology Society EMBC’2011 “Integrating Technology and Medicine for a Healthier Tomorrow”*, Boston, Massachusetts, August 30 – September 3, 2011, pp. 3362–3366.

[8] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, Chapman & Hall/CRC, Boca Raton, Florida, 2007.

[9] A. Statnikov, I. Tsamardinos, Y. Dosbayev, and C. Aliferis, “GEMS: A system for automated cancer diagnosis and biomarker discovery from microarray gene expression data”, *International Journal of Medical Informatics*, 2005, Vol. 74, pp. 491–503.

[10] V. Vapnik, *The Nature of Statistical Learning Theory*, 2nd ed. Springer-Verlag, New York, 2000.

[11] L. A. Zadeh, “Fuzzy sets”, *Information and control*, 1965, Vol. 8, pp. 338–353.