Kernel Combination Through Genetic Programming for Image Classification

Yuri H. Ribeiro, Zenilton K.G. do Patrocínio Jr.(✉), and Silvio Jamil F. Guimarães

Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte, Brazil
yuri.ribeiro@sga.pucminas.br, {zenilton,sjamil}@pucminas.br

Abstract. Support vector machine is a supervised learning technique which uses kernels to perform nonlinear separations of data. In this work, we propose a combination of kernels through genetic programming in which the individual fitness is obtained by a K-NN classifier using a kernel-based distance measure. Experiments have shown that our method KGP-K is much faster than other methods during training, but it is still able to generate individuals (*i.e.*, kernels) with competitive performance (in terms of accuracy) to the ones that were produced by other methods. KGP-K produces reasonable kernels to use in the SVM with no knowledge about the distribution of data, even if they could be more complex than the ones generated by other methods and, therefore, they need more time during tests.

Keywords: Genetic programming · Support vector machines · Kernel combination · Image classification

1 Introduction

Support vector machine (SVM) is a supervised learning technique conceived by [2] as a binary linear classifier. Given a vector space H and a set of data such as $S = (x_i, y_i)$ in which $x_i \in H, y_i \in \pm 1$, a SVM calculates an optimal hyperplane that separates the data from two classes.

However, according to [9], such hyperplane might not exist since a single outlier in the training data can impact negatively on the calculation of the optimal hyperplane. Therefore, it is desirable that SVM tolerates a certain degree of error to deal with outliers. In order to cope with this issue, the authors in [2] introduced slack variables during the calculation of the optimal hyperplane in order to relax the separation constraints. Since large values of those variables could lead to trivial solutions, they also proposed the use of a margin weight to control the size of the margin.

Since SVM is a linear classifier, in order to perform nonlinear separations of data, kernel functions can be used to transform the given data to a higher

The authors are grateful to PUC Minas, CNPq, CAPES and FAPEMIG for the partial financial support of this work.

A. Pardo and J. Kittler (Eds.): CIARP 2015, LNCS 9423, pp. 314–321, 2015.
DOI: 10.1007/978-3-319-25751-8_38

dimensional feature space in which they are linearly separable. A kernel function (represented by $k(\cdot,\cdot)$) are continuous symmetrical functions. According to [10], it is difficult to identify if a function is a kernel, thus any function that satisfies Mercer's theorem [6] could be used. Moreover, a closure property for some operations between kernels could also be explored to generate new kernels [9,10].

Although there are ways of solving efficiently the primal form of the SVM problem (*i.e.*, the calculation of the optimal hyperplane) [8], most of the literature addresses the following dual form of the SVM problem, which is considered more conveniently solvable, as follows

$$\underset{\alpha\in\mathbb{R}^m}{\text{maximize}}\ W(\alpha) = \sum_{i=1}^{m}\alpha_i - \tfrac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j k(x_i,x_j), \tag{1}$$

$$\text{subject to} \tag{2}$$

$$\sum_{i=1}^{m}\alpha_i y_i = 0 \tag{3}$$

$$0 \le \alpha_i \le \tfrac{C}{m} \quad ,\forall i = 1,\ldots,m \tag{4}$$

in which W is the vector of hyperplane coefficients, C and ξ are the aforementioned margin weight and slack variables, respectively, and α is the vector of dual variables corresponding to each separation constraint. According to [3], the decision function can be written as

$$f(x) = sgn\left(\sum_{i=1}^{m}\alpha_i y_i k(x_i,x) + b\right) \tag{5}$$

in which b is the bias of the separating hyperplane.

In general, the choice of an appropriate kernel is one important design decision when SVM is used; however, that task is nontrivial. The usual approach is based on a set of predefined kernels from the literature when there is no knowledge about the distribution of data for a given application. Another way is to analyze the application data in order to obtain insights about their distribution. Either ways the usage of SVM gets to be limited.

Evolutionary algorithms (EAs) have presented good results in evolving kernels by making combinations of kernels or searching for kernel parameters in diverse applications [3,10]. In [10], the authors proposed the combination of kernels through genetic programming in which the fitness of each individual (representing a kernel function) is set to the accuracy of a SVM classifier, *i.e.*, at each iteration of their evolutionary method – hereafter called KGP, they have trained and evaluated each kernel function using part of the dataset. However, in [10] training sets are small and the execution time is neglectable, thus it is not possible to infer the consequences of using KGP over large (web scale) datasets.

In this work, we also propose the use of genetic programming (GP) to combine kernels, but, in our GP method, the fitness of each individual is obtained by a K-Nearest Neighbor (K-NN) classifier using a kernel-based distance measure. Experiments have shown that our method, so-called KGP-K, is much faster than KGP during training, but it is still able to generate individuals (*i.e.*, kernels) with

competitive performance (in term of accuracy) to the ones that were produced by other methods.

This work is organized as follows. Section 2 discussed genetic programming and presents its use for kernel combination. Our proposed method is presented in Section 3. Some experimental results performed on a well known image dataset are given in Section 4. Finally, in Section 5, some conclusions are drawn and future works are pointed out.

2 Kernel Genetic Programming

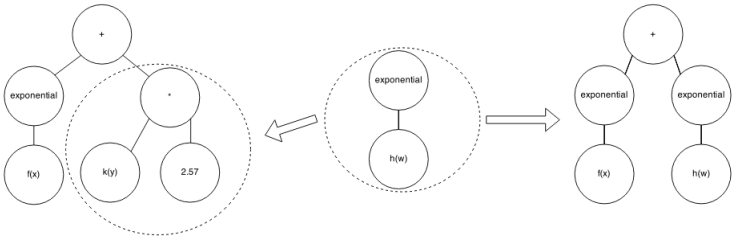
Genetic programming (GP) [4], similar to genetic algorithm (GA), is an evolutionary method based upon the principles of biological evolution. The major difference between GP and GA is that GP individuals are generally represented by trees whose size constantly change during execution.

In those, a number of solutions (or individuals) to a given problem are randomly generated, and they together are called a population. The individuals are made in a way such that two operations – called genetic operators – can be applied to then: *mutation* and *crossover*. In the first, a randomly selected part of an individual is exchanged by a new one that is randomly generated; in the latter, randomly selected parts of two different individuals are exchanged between them. The algorithm runs for a number of iterations (each one is called a *generation*). In each generation, the individuals are evaluated in some way (which is dependent on the application domain). The result of such evaluation is called *fitness* and it is used to determine to which individuals the genetic operators will be applied and which of them are the best from their generation (*i.e.*, the best solutions to the problem). The algorithm stops when some condition is reached, such as a certain value of fitness, a maximum number of generations, or both. Fig. 1(a) shows an example of mutation operator, while Fig. 1(b) illustrates a crossover operator.

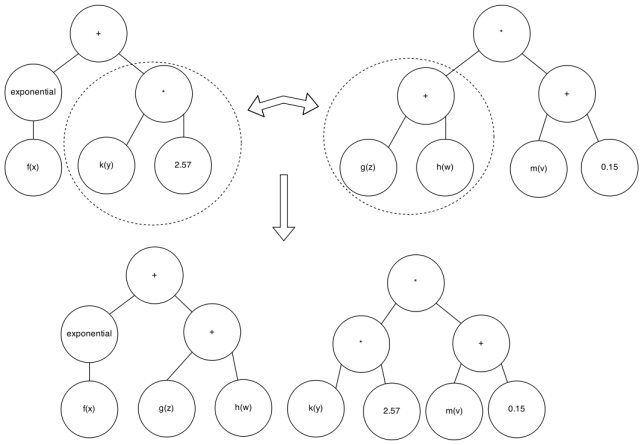
The KGP algorithm proposed in [10] uses GP to generate valid kernels. It starts with a set of Mercer's kernels randomly generated and uses the aforementioned closure property to produce new kernels. The operations used are exponential, addition or multiplication by another kernel and the multiplication by a real number. Each individual has its fitness set to the accuracy a SVM classifier that is generated by using the kernel represented by the individual itself.

3 Proposed Method

Given a training dataset, whose classes are known, and an observation of unknown class. A K-NN classifier decides the class of the observation based on the classes of the K nearest neighbors in the training set by using some statistic such as the mode. In order to generate a K-NN classifier, we only need a distance measure between the data elements.



(a) Mutation.



(b) Crossover.

Fig. 1. Examples of genetic operator of GP.

A kernel distance [12] is a distance measure between objects based on a kernel. Given a definite positive kernel function k and two points p and q, a kernel distance d_k is given by:

$$d_k(p,q) = k(p,p) + k(q,q) - 2k(p,q) \tag{6}$$

Therefore, the K-NN classifier using the kernel distance d_k can be used to obtain a fitness for each individual during KGP algorithm. Since there is no off-line learning in the K-NN classifier, its use is expected to be inferior to the use of SVM in terms of accuracy. But since the calculations of a K-NN classifier are simpler, it is also expected to be faster. A simplified diagram of the process can be seen in Fig. 2

4 Experiments and Results

In this section, we present the results of experiments made to assess the performance of the proposed method. During experiments, the **FGComp2013**[1]

[1] https://sites.google.com/site/fgcomp2013/

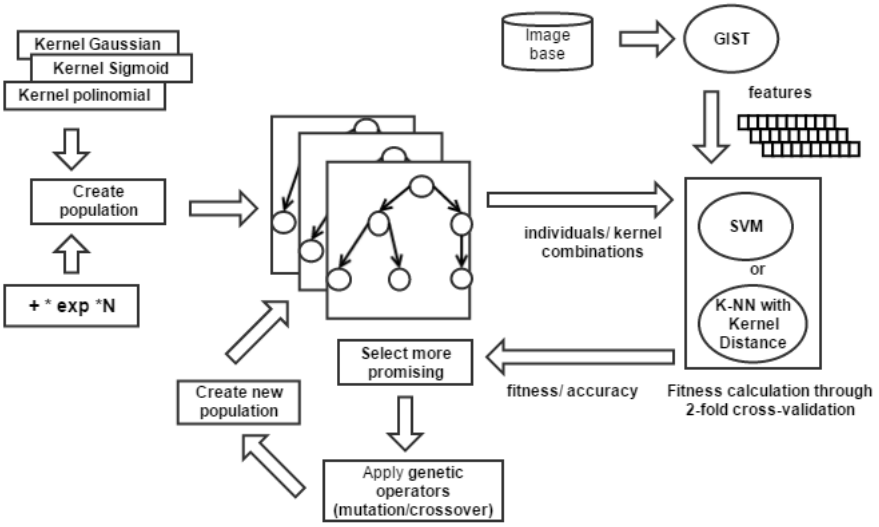


Fig. 2. KGP

dataset was used. It is a 5-domain subset of the **ILSVRC2013** (*Imagenet Large Scale Visual Recognition Challenge*[2]) and contains 75,533 images (49,052 for training and 28,481 for test).

The global descriptor GIST [7] was used to describe the visual content of the images x_i in H addressing the semantic perception. According to [11], the GIST can be defined as a feature vector g, in which each component g_k is given by

$$g_k = \sum_{x,y} w_k(x,y) \times |I(x,y) \otimes h_k(x,y)|, \tag{7}$$

in which $\otimes$ and $\times$ denote convolution and multiplication of values related to the pixels, respectively; $I(x,y)$ is the luminance channel for the input image; $h_k(x,y)$ is a multiscale Gabor filter bank and $w_k(x,y)$ is a spatial window which is used for computing the average energy of each filter in different positions at the image.

In order to compare our method to [10], we implement it using ECJ [5]. For individual evaluation, we use both LIBSVM [1], so-called KGP, and our own implementation of K-NN using kernel distance, so-called KGP-K (which considers 5 nearest neighbors). In both cases, we have 5 generations and 30 individuals for GP, with a probability of 0.9 for crossover operation and a probability of 0.1 for mutation. We also take advantage of multicore technology and explore the parallelization during individual evaluation using 10 threads to do that (which generates a processor occupancy of almost 90%).

[2] http://www.image-net.org/challenges/LSVRC/2013/

Table 1. Experimental results for train step

	Accuracy		Time (in min)	
	average	Δ	average	Δ
KGP	93.86	0.08	2131	211
KGP-K	89.24	0	230	35
Grid	93.69	0	471	0
Random Walk	92.83	0.41	1080	49
Monte Carlo	93.36	0.10	1305	390

We have also compare KGP and KGP-K to: (i) a Grid search of LIBSVM, which is a strategy to explore multiple combinations of C and γ for the Gaussian kernel using cross validation and takes the best as the solution; (ii) a Monte Carlo's method which generates 150 random kernel combinations and takes the best one as the solution; and (iii) a Random Walk method using 30 random combinations of two kernels, followed by a test to choose the best as "current kernel combination". In our experiments, due to restriction of time, we have only three runs for each method in training and test steps (*e.g.*, KGP has taken an average time of one day and a half during training step).

Table 1 presents the average and standard deviation concerning the train step, while Table 2 presents the average and standard deviation for the test step. The experiments were done by using a Core i7 5820k with 16G of memory running Windows 8.1. It is worth to mention that, during training with KGP and KGP-K, each kernel (*i.e.*, individual) is evaluated (*i.e.*, has its fitness calculated) using a 2-fold cross validation over the training data with a SVM classifier and a K-NN classifier, respectively. At end of the training step, the best kernel should be tested, and we do that using a 2-fold cross validation over the testing data with a SVM classifier for both: KGP and KGP-K.

The results of Grid and KGP are very similar. The proposed method KGP-K is inferior during training but it presents a great improvement during tests. It is worth to notice that KGP-K spends a very short time in training. Monte Carlo

Table 2. Experimental results for test step

	Accuracy		Time (in min)	
	average	Δ	average	Δ
KGP	95.34	0.07	433	107
KGP-K	94.82	0.12	626	47
Grid	95.38	0	246	0
Random Walk	94.74	0.18	989	138
Monte Carlo	95.11	0.15	606	376

is superior to KGP-K in terms of accuracy but it spends more time than KGP-K during training. Finally, KGP-K represents a feasible alternative for kernel evolution in scenarios that training time is an important issue. KGP-K produces reasonable kernels to use in the SVM with no knowledge about the distribution of data, but they could be more complex than the ones generated by KGP and, therefore, they need more time during tests.

5 Conclusion

In this work, we propose the combination of kernels through genetic programming in which the individual fitness is obtained by a K-NN classifier using a kernel-based distance measure. Experiments have shown that KGP-K is much faster than KGP during training, but it is still able to generate individuals (*i.e.*, kernels) with competitive performance (in term of accuracy) to the ones that were produced by other methods.

Thus, KGP-K represents a feasible alternative for kernel evolution in scenarios that training time is an important issue. KGP-K produces reasonable kernels to use in the SVM with no knowledge about the distribution of data, but they could be more complex than the ones generated by KGP and, therefore, they need more time during tests.

In future works, we will explore ways to control the individuals during the evolutionary process to prevent the generation of complex (and time-consuming) individuals. It will be also interesting to assess the impact on our method of descriptors with more semantic information about the problem domain.

References

1. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology **2**, 27:1–27:27 (2011)
2. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning **20**(3), 273–297 (1995)
3. Gönen, M., Alpaydın, E.: Regularizing multiple kernel learning using response surface methodology. Pattern Recognition **44**(1), 159–171 (2011)
4. Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection, vol. 1. MIT press (1992)
5. Luke, S.: Ecj 13: A java EC research system (2005)
6. Mercer, J.: Functions of positive and negative type, and their connection with the theory of integral equations. Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character **209**, 415–446 (1909)
7. Oliva, A., Torralba, A.: Modeling the shape of the scene: A holistic representation of the spatial envelope. International Journal of Computer Vision **42**(3), 145–175 (2001)
8. Shalev-Shwartz, S., Singer, Y., Srebro, N., Cotter, A.: Pegasos: Primal estimated sub-gradient solver for SVM. Mathematical Programming **127**(1), 3–30 (2011)
9. Smola, A.J., Schölkopf, B.: Learning with kernels. Citeseer (1998)

10. Sullivan, K.M., Luke, S.: Evolving kernels for support vector machine classification. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp. 1702–1707. ACM (2007)
11. Torralba, A., Murphy, K., Freeman, W.: Using the forest to see the trees: exploiting context for visual object detection and localization. Communications of the ACM **53**(3), 107–114 (2010)
12. Wu, G., Chang, E.Y., Panda, N.: Formulating distance functions via the kernel trick. In: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, pp. 703–709. ACM (2005)