Binary Classification Using Fast Gaussian Filtering Algorithm

Kentaro Imajo, Keisuke Otaki, and Akihiro Yamamoto

Graduate School of Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto, 606-8501, Japan.
{imajo,ootaki}@iip.ist.i.kyoto-u.ac.jp,akihiro@i.kyoto-u.ac.jp

Abstract. We propose a new algorithm for binary classification based on the Fast Gaussian Filtering Algorithm (FGFA), which can compute Gaussian-filtered values fast in low-dimensional spaces. It is originally developed for dense data like images. Extending the algorithm with range trees to deal with sparse data, we propose a new classifier. We show that the classifier is the same as a C-SVM whose parameter C approaches 0. We give experimental results to show that our classifier has as good accuracy as C-SVMs, and we can reduce computational time of predictions of C-SVMs when the number of suppor vectors is larger than 10,000.

Keywords: Binary classification, Support vector machines, Gaussian filtering, Gaussian function

1 Introduction

In this paper, we propose a new algorithm for binary classification tasks based on the Fast Gaussian Filtering Algorithm (FGFA) proposed by Imajo [1]. We also give experimental results for evaluating our algorithm.

It is well-known that SVMs have reputations for outperforming other methods [2, 3]. The basic idea of SVMs is to maximize margins between two classes using quadratic programming problems, and they can predict a class of an unseen sample based on hyperplanes that are constructed with the answer of a quadratic programming problem. In applications of SVMs like the case of LIBSVM [4], however, the computational time for each prediction gets longer than linear time of the number of training samples. Thus, we need to decrease the cost if we use a lot of support vectors for classification. This is our motive of adopting approximate computations for classification. By FGFA [1], we can speed up the computation of the Gaussian filtering for low-dimensional dense data such as image data. We extends FGFA using the range trees [5] in order to apply FGFA to sparse data. Our classifier is constructed by the extended FGFA. In addition, we show that our method can be regarded as C-SVMs, which are known as soft-margin SVMs, whose parameter C approaches 0.

Section 2 shows preliminaries, and Section 3 gives our algorithm. Section 4 compares our algorithm with C-SVMs. Section 5 evaluates our algorithm by accuracy and computational time comparing with LIBSVM, which implements C-SVMs. Finally, we conclude our study in Section 6.

2 Preliminaries

2.1 FGFA (Fast Gaussian Filtering Algorithm)

We proposed FGFA in the last paper [1]. The paper revealed that we can compute a Gaussian-filtered value fast also in low-dimensional sparse spaces. In this paper, we use the second-order approximation of the Gaussian function:

$$\tilde{\psi}(x) = 3(x+11)_+^2 - 11(x+3)_+^2 + 11(x-3)_+^2 - 3(x-11)_+^2, \tag{1}$$

$$\simeq \psi(x) = 2.657 \times 10^2 \exp\left(-\frac{x^2}{5.2720^2}\right), \tag{2}$$

where $(\cdot)_+ \equiv \max(0, \cdot)$. The approximation error of the second-order approximation is about 2%, which is calculated as follows:

$$\frac{\int_{\mathbb{R}} \left|\psi(x) - \tilde{\psi}(x)\right| dx}{\int_{\mathbb{R}} |\psi(x)|\, dx} \approx 0.02009. \tag{3}$$

Let ψ be the Gaussian function and I be a $2D$ image, a Gaussian-filtered value $(\psi * I)(x, y)$ at the pixel (x, y) can be written as:

$$(\psi * I)(x, y) = \sum_{(\Delta x, \Delta y) \in \mathbb{Z}^2} \psi(\Delta x, \Delta y) I(x - \Delta x, y - \Delta y). \tag{4}$$

The proposed method by Imajo [1] for speeding up the calculation of Equation 4 is represented as follows:

$$(\psi * I)(x, y) \simeq \sum_{i=1}^{m} \sum_{j=1}^{m} a_i a_j J(x + b_i, y + b_j), \tag{5}$$

where $a = \{3, -11, 11, -3\},\ b = \{11, 3, -3, -11\},\ m = 4$, and $J(x, y)$ is calculated by precomputation represented as follows:

$$J(x, y) = \sum_{(\Delta x, \Delta y) \in \{(0,0), \cdots, (x,y)\}} \Delta x^2 \Delta y^2 I(x - \Delta x, y - \Delta y). \tag{6}$$

Thus, the second-order approximation takes $O(n \log d)$-time for precomputation, and it takes $O(4^d)$-time for computation of a Gaussian-filtered value, where n is the number of pixels, and d is a dimension.

2.2 *d*-dimensional Range Queries

In this paper, we use range trees [5] to process d-dimensional range queries such as summation of values in a d-dimensional rectangular area. A d-dimensional range tree can be constructed in $O(n \log^d n)$ time, where n is the number of points stored in the tree, and every d-dimensional range query can be processed in $O(\log^d n)$ time.

3 FGFA for Sparse Data

FGFA assumes that an input data consists of dense data such as an image. The precomputation of FGFA takes linear time on the number of pixels of an input image. If we deal with continuous values for coordinates as is, a large number of pixels are required to approximate. Therefore, this section extends the d-dimensional range query algorithm to calculate integrated values of sparse data, and it applies the extension to FGFA.

3.1 Extension of d-dimensional Range Tree

The d-dimensional range tree algorithm provides a method to calculate summation of a rectangular area of d-dimensional spaces. Using the method, this section shows a method to calculate J in Equation 6. The d-dimensional range tree algorithm can compute a value $K_{i,j}(x, y)$ in $O(\log^d n)$-time that is written as:

$$K_{i,j}(x,y) = \sum_{(\Delta x, \Delta y) \in \{(0,0), \cdots, (x,y)\}} \Delta x^i \Delta y^j I(\Delta x, \Delta y). \tag{7}$$

Using the values K, J can be rewritten as:

$$J(x,y) = \begin{pmatrix} x^2 \\ -2x \\ 1 \end{pmatrix} \begin{pmatrix} y^2 \\ -2y \\ 1 \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} K_{0,0}(x,y) & K_{1,0}(x,y) & K_{2,0}(x,y) \\ K_{0,1}(x,y) & K_{1,1}(x,y) & K_{2,1}(x,y) \\ K_{0,2}(x,y) & K_{1,2}(x,y) & K_{2,2}(x,y) \end{pmatrix}. \tag{8}$$

Therefore, J can be computed in $O(\log^{2d} n)$ time.

3.2 Extension of FGFA

Since the extension of d-dimensional range tree offers a method to calculate J in $O(\log^{2d} n)$ time, we reveal that FGFA can be computed in $O(4^d \log^{2d} n)$ time.

3.3 Proposed Classifier

Using the extension of FGFA, the proposed classifier approximates the following function y_{test} for binary classification:

$$y_{\mathrm{test}} = \mathrm{sgn}\left(\sum_{i=1}^{n} y_i \exp\left(-\frac{|x_{\mathrm{test}} - x_i|^2}{\sigma^2} \right) \right). \tag{9}$$

4 Similarities With C-SVM

We show that the proposed algorithm is the same classifier as a C-SVM [2] with the Gaussian kernel whose parameter C approaches $+0$. C-SVMs with the Gaussian kernel can be constructed by solving the following quadratic programming problem.

$$\begin{aligned} \max: & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \exp\left(-\frac{|x_i - x_j|^2}{\sigma^2} \right), \\ \text{s.t.}: & \ 0 \leq \alpha_i \leq C \ (i = 1, \cdots, n), \ \sum_{i=1}^{n} \alpha_i y_i = 0, \end{aligned} \tag{10}$$

where x_i is a vector of parameters, y_i is a class for x_i, C and σ are predefined parameters of a C-SVM and α_i is a variable. If C approaches $+0$, the first term of Equation 10 can be ignored. We can maximize Equation 10 by $\alpha_i = C$ if two classes have the same number. Therefore, we can consider the proposed algorithm a C-SVM whose parameter C approaches $+0$.

5 Experiments

In this paper, we evaluate the proposed method comparing with a popular C-SVM software LIBSVM [4]. First, we compare accuracy of classification. Second, we compare computational time of prediction. The proposed method is implemented by C++ language, and we compute experiments of this section on a computer that installs Core i7 2.8GHz, 16 GB memory, Mac OS X 10.7 and G++ 4.2.

5.1 Comparison of Accuracy

We evaluate the proposed method comparing accuracy of classification of the Iris flower data set [6] with C-SVMs. Since the data set has three classes and we want to evaluate binary classification, we use two classes Versicolor and Virginica in the Iris flower data set. Both of the proposed method and C-SVMs require a parameter σ, and C-SVMs require another parameter C. We evaluated them for each of $C \in \{0.1,\ 1,\ 10,\ 10^6\}$, $\sigma \in \{0.1,\ 0.3,\ 1,\ 3,\ 10\}$. Table 1 is the result of 5-fold cross-validation.

5.2 Comparison of Computational Time

We evaluate the proposed method comparing computational time of prediction with LIBSVM. Training data consist of random real numbers between 0 and 10. Since the proposed method requires integer attributes, we round off a fraction to three decimal places. Table 2 shows that computational time to predict of the proposed method is less linear on the number of training data while LIBSVM

Table 1. Accuracy of classification of the Iris flower data set
(Classification between two species with 5-fold cross validation)

	Proposed Method	C-SVM			
		$C = 0.1$	$C = 1$	$C = 10$	$C = 10^6$
$\sigma = 0.1$	93%	58%	68%	69%	69%
$\sigma = 0.3$	93%	83%	94%	93%	93%
$\sigma = 1$	93%	93%	95%	93%	92%
$\sigma = 3$	86%	90%	94%	95%	89%
$\sigma = 10$	84%	85%	90%	93%	92%

Table 2. Computational time to predict a class
(Every vector has two real numbers between 0 and 10, σ is 1)

# of training data		$1,000$	$10,000$	$100,000$	$1,000,000$
Proposed method	Prediction (Precomputation)	2.43 ms (0.13 s)	2.48 ms (1.48 s)	2.55 ms (17.8 s)	2.96 ms (203 s)
LIBSVM	Prediction (Training)	0.02 ms (0.05 s)	0.22 ms (4.47 s)	2.29 ms (442 s)	23.6 ms (1000+ s)

predicts a class by calculating all the distances between a target and training data. Precomputation of the proposed method takes nearly linear on the number of training data. Training of LIBSVM takes time that is nearly proportional to the square of the number of training data.

6 Conclusions

This paper showed that FGFA can be applied to sparse data using the d-dimensional range tree algorithm. This is the same classification as a C-SVM whose parameter C approaches 0, and it has as good accuracy as C-SVMs. Additionally, we revealed that this can reduce time of prediction of C-SVMs when the number of support vectors is larger than 10,000.

References

1. Kentaro Imajo. Fast Gaussian Filtering Algorithm Using Splines. In *The 21st International Conference on Pattern Recognition*, 2012.
2. Koji Tsuda. Overview of Support Vector Machine. *The Journal of the Institute of Electronics, Information, and Communication Engineers*, 83(6):460–466, 2000.
3. Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.
4. Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27, 2011.
5. Jon Louis Bentley. Decomposable Searching Problems. *Information Processing Letters*, 8(5):244–251, 1979.
6. R.A. Fisher. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Human Genetics*, 7(2):179–188, 1936.