Offline Signature Verification Using Virtual Support Vector Machines

Samuel Audet, *ID: 260184380,* Peyush Bansal, *ID: 110223002,* and Shirish Baskaran, *ID: 110232333*

***Abstract*— Offline signature verification is the art of properly classifying between one's real signature and reasonably good forgeries from after-the-fact (scanned or otherwise captured) images. For this project, we implemented a technique of signature verification using Virtual Support Vector Machines. A testing procedure was devised and the results were analyzed and written in this report. When compared to SVM classification without virtual signatures, it was found that the use of Virtual SVM with invariant rotation and translation transformations has a small detrimental effect on the error rate, but reduces the false rejection rate at the expense of the false acceptance rate which increases significantly.**

***Index Terms*— signature verification, image processing, support vector machines, classifiers, sequential minimal optimization, virtual examples, virtual support vector machines, invariant transformation.**

I. Introduction

OFFLINE signature verification is the art of properly classifying between one's real signature and reasonably good forgeries from after-the-fact (scanned or otherwise captured) images. Signing is a very conventional, efficient and simple to use biometric authentication method for humans and has been used extensively for legal purposes for centuries now. With recent advances in computer vision, computational performance and classifier technologies, it might soon be possible to let computers autonomously verify signatures found on checks and other legal documents with limited human intervention.

We chose to implement such an offline signature verification system for our final ECSE 526 course project using trainable Support Vector Machine classifiers. First, we will briefly describe the image processing needed to extract feature vectors. Then, the SVM algorithm itself as well as our SMO implementation are detailed. Finally, we will talk about our testing procedure as well as present and analyze test results.

In this report, we will refer to only three kinds of signatures: genuine signatures, random forgeries and skilled forgeries (Fig. 1). Genuine signatures are the real signatures of a given person. Random forgeries are genuine signatures from other people. Skilled forgeries are actual imitations of the genuine signatures of interest.

II. Signature Image Processing

In the case of normal character recognition, the image of the character itself is fed as the feature vector to the SVM algorithm. However, in the case of signatures, the feature vector would be too large (for example, a typical signature image of 400×200 pixels would make for a feature vector of 80000 components). In order to reduce complexity and dimensionality, and in an attempt to extract important characteristics of the signatures, the images of the signatures are first processed before being fed to the SVM algorithm. In our case, we decided to implement the method of Justino et al. [1], since it was simple and according to the authors gave good results.

Fig. 1. Types of signatures (from [1]).

For this procedure, the image is first segmented into what are called *cells* (Fig. 2), of about 16×25 pixels for signatures scanned at 300 DPI and of about 4×6 pixels for 75 DPI. Within each cell, four features are computed (Fig. 3): the pixel density, the gravity center distance, the segment curvature, and the predominant slant or orientation.

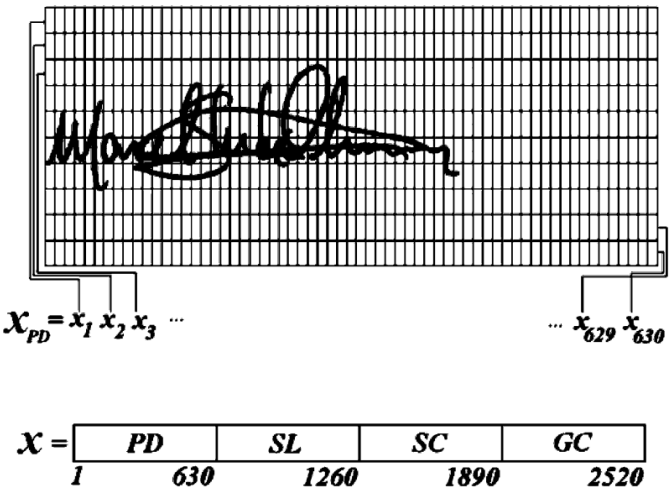

Fig. 2. Feature vector representation for signatures (from [1]).

Report created on April 7, 2006; revised May 7, 2006, McGill University, Montréal, Québec.

Samuel Audet <saudet@cim.mcgill.ca> is with the Centre for Intelligent Machines. Peyush Bansal <peyush.bansal@mail.mcgill.ca> and Shirish Baskaran <shirish.baskaran@mail.mcgill.ca> are with the Electrical and Computer Engineering Department.

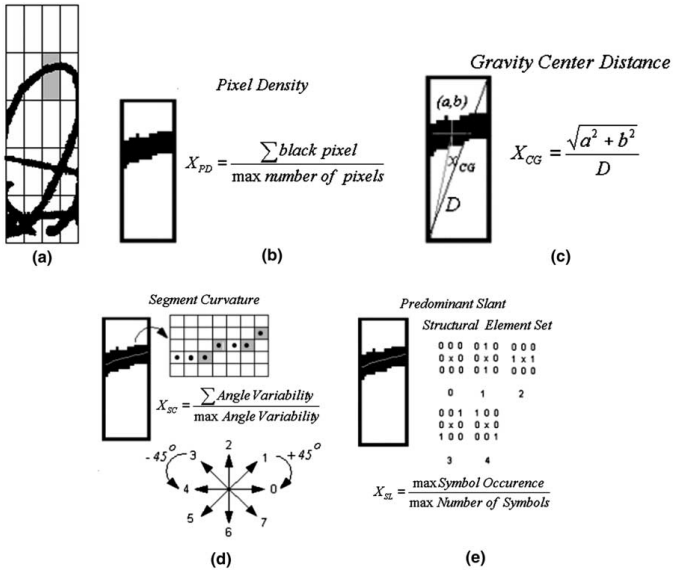

Fig. 3. The feature extraction method (from [1]): (a) segmented cell; (b) pixels density; (c) gravity center distance; (d) segment curvature and (e) predominant slant definition.

The first two features are easy to compute and account for static graphometric features. The pixel density computes the ratio of the number of "black" pixels to the total number of pixels in a cell. The gravity center distance is the distance from the center of gravity (first moment or average position) of all the "black" pixels to one of the corners of the cell. The two last features are a bit more involved and account for semi-dynamic graphometric features (which change with the way a signature is written). The image is first converted into a skeleton image. The application of a thinning procedure [2] is the easiest method of obtaining a skeleton of relatively good quality. The segment curvature is computed by walking around a "black" pixel, and by matching pairs of "black" pixels in the skeleton image. The curvature of a 3×3 pixel neighborhood can be seen as counting the number of steps required to reach one pixel from another pixel by walking around the center pixel. The number of absolute steps required to walk from the deviation of a straight line is the curvature measure we are interested in, such that three aligned pixels would give a curvature measure of 0, and anything else would be greater than 0, but smaller than 8. The average for all pixels in a given cell gives the segment curvature. The predominant slant is a measure of the orientation of the segment in the skeleton image. Each pattern in Fig. 3 (e) is matched for each "black" pixel, and the average over all pixels represents the predominant slant feature for the cell. For empty cells, all features are set to 0.

These features for all cells are concatenated together giving for this signature a feature vector (Fig. 2) which is then directly fed to the SVM algorithm. The dimensionality of such a feature vector is usually of about 3000, but it is always quite sparse.

For Virtual SVM (VSVM), the support vectors found during the course of the training of an SVM classifier are sent back to the image processing module to undergo invariant transformation before retraining [3]. Since support vectors are

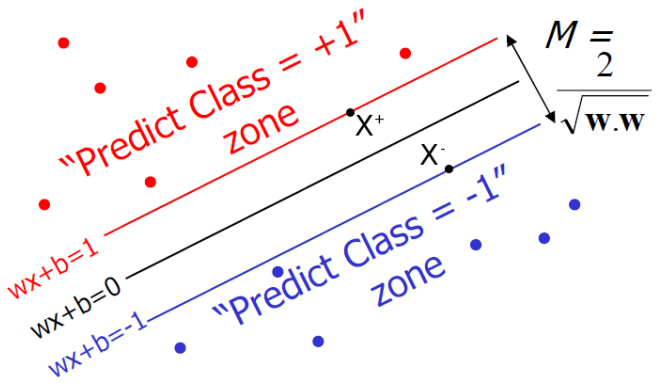

Fig. 4. A sample linear SVM classifier and its margin (from [5]).

the difficult examples to classify, it makes sense to concentrate further computational effort on those examples alone. After some informal testing regarding the performance on a few signatures, it was decided to settle on the following: 3 rotations of -5, 0 and 5 degrees; and 3 translations of a quarter of a cell width in the direction of both axes. This procedure generates $3\times3\times3-1=26$ virtual examples per signature or forgery. The SVM classifier is then retrained on those new virtual examples, in the hope of producing a more robust classifier.

III. Support Vector Machines

To perform signature classification we decided to use the technique of Support Vector Machines since it can handle data sets with large number of attributes and has a better generalization performance [4] than other methods.

Unlike Neural Networks for which the number of sigmoid units that define the structure is fixed before training and for which the network weights are determined by training for minimum mean-squared error between actual and desired outputs (empirical risk minimization), SVM classifiers can learn the optimal structure by optimizing their parameters for minimum classification error (structural risk minimization). Empirical risk minimization relies on the availability of large amounts of training data [4]. Hence, SVM classifier are even more appropriate in our case since the amount of training data available for signatures is usually limited.

The principle behind SVM relies on a linearly separable feature space (Fig. 4). The objective is to find the optimal hyperplane that is uniquely determined by a set of the vectors (or data points) at equal distance from the hyperplane – the support vectors.

The support vectors are selected from the training data so that the distance between the two hyperplanes passing through the support vectors (x^+ and x^-) is at a maximum. The optimal hyperplane maximizes the empty space (or margin) between all training data points.

For the linearly separable case there will not be any training data between the two hyperplanes and the decision plane, which is midway between the two hyperplanes. Moreover, this optimal decision plane should minimize classification error. This has been proved by Vapnik [6].

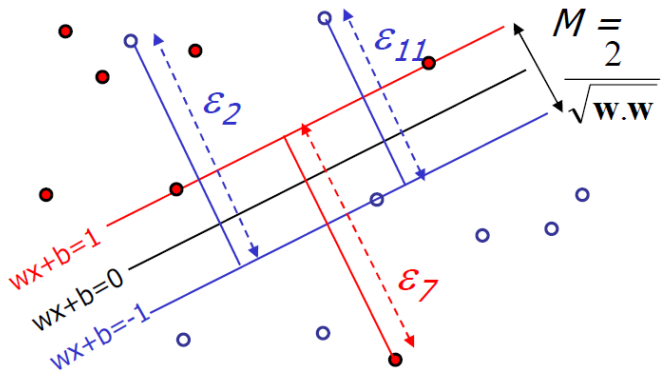

Fig. 5. A sample linear SVM classifier with a soft margin (from [5]).

From Fig 4, the decision hyperplane can be expressed as follows:

$$x \cdot w + b = 0. \tag{1}$$

All the n training data must satisfy the following constraint:

$$y_i(x_i \cdot w_i + b) \geq 1 \quad \text{for} \quad i = 1, ..., n. \tag{2}$$

The SVM approach aims at maximizing the margin of separation which can be calculated as follows (Fig 4):

$$M = \frac{w \cdot (x^+ - x^-)}{\|w\|} = \frac{2}{\|w\|}. \tag{3}$$

For cases where the training data points are not clearly separable the margin can be relaxed to facilitate a more robust decision (Fig. 5).

In this case, the constraint is posed as follows:

$$y_i(x_i \cdot w_i + b) \geq 1 - \epsilon_i \quad \text{for} \quad i = 1, ..., n \tag{4}$$

where ϵ_i are slack variables which measure the deviation of data points from the marginal hyperplane and allow some data points to violate the initial constraint. Therefore, the objective is to minimize:

$$\frac{1}{2}\|w\|^2 + C\sum_i \epsilon_i \quad \text{subject to} \quad y_i(x_i \cdot w_i + b) \geq 1 - \epsilon_i \tag{5}$$

where C is a parameter used to penalize violations of the margin. The value of C is defined constant or chosen by validation.

A typical and good approach to solving the above quadratic minimization problem is by introducing Lagrange multipliers $\alpha \geq 0$ and reformulating the optimization equation as a Lagrangian:

$$\max_\alpha \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n} \alpha_i \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \tag{6}$$

subject to the following constraints:

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \text{ where } 0 \leq \alpha_i \leq C. \tag{7}$$

This formulation helps is two ways. First, the constraints on Lagrange multipliers are much simpler to handle. Secondly,

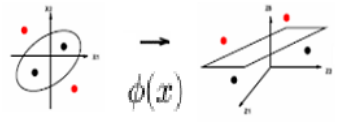

Fig. 6. Mapping of data to a higher dimensional feature space.

the training data now only appears in the form of dot products of vectors [4].

For the cases where the data is not linearly separable, the fact that it only appears in the form of dot products makes the mapping of the data x from the input space to a higher dimensional feature space (Fig. 6) much less complex.

This gives rise to the concept of "kernel tricks". A kernel is defined as follows:

$$K(x, x') = \phi(x) \cdot \phi(x'), \tag{8}$$

where $\phi(x)$ is a function mapping x to a higher dimensional space. However, this function does not need to be known to evaluate $K(x, x')$. The SVM algorithm only uses K and does not need to know what individual ϕ represent. It is for this reason that SVM can handle large dimensional feature vectors without significant effect on performance [4].

A. Sequential Minimal Optimization

Moving into the implementation aspects of SVM, we decided to implement the Sequential Minimal Optimization (SMO) algorithm as outlined by Platt [7] since it is known to be a fast and simple method of realizing SVM classification. The Lagrangian of equation 6 form a quadratic programming (QP) problem that the SMO algorithm can solve. The constraints on the Lagrangian are known as the Karush-Kuhn-Tucker (KKT) conditions within the context of this constrained optimization. A solution is optimal only when all the data points satisfy the KKT conditions which can be reformulated as:

- $\alpha_i = 0$ for linearly separated points $(y_i f(x_i) \geq 1)$
- $0 < \alpha_i < C$ for support vectors $(y_i f(x_i) = 1)$
- $\alpha_i = C$ for points violating the margin $(\epsilon_i > 0$ and $y_i f(x_i) < 1)$

The SMO algorithm breaks the large QP problem into the set of smallest possible QP problems, by selecting and optimizing two Lagrange multipliers at a time while holding the rest constant in order to satisfy the constraints of equation 7.

The advantage of SMO lies in the fact that the optimization of the two selected Lagrange multipliers can be done analytically, thus computationally intensive numerical QP optimization which would have required special QP libraries is avoided. In addition, by avoiding large matrix computations and by simultaneously optimizing only two Lagrange multipliers at a time, we make the training procedure less susceptible to numerical instabilities that can arise when working with large matrices. Also of great importance, the memory required by SMO is linearly proportional to the the size of the training set, allowing the SVM classifier to handle very large amounts of data.

The SMO algorithm can be broken down into three components:

1) a heuristic for choosing which Lagrange multipliers to optimize,
2) an analytical method to solve for the two selected Lagrange multipliers, and
3) a method for computing b.

In order to convey the overall structure of the algorithm, a brief summary of the first component is provided below. However, since the mathematical derivations are quite involved, they are not detailed in this report. For the interested reader, more information about SMO has been published by Platt [7].

The SMO algorithm is guaranteed to converge [7]. However, in order to accelerate the convergence, heuristics are used to detect which Lagrange multipliers should provide the best improvement upon optimization. There are basically two areas of the algorithm that can benefit from heuristics. One is found within the outer loop where the first Lagrange multiplier is picked, while the other is found within the inner loop where the second multiplier is picked.

The processing done within the outer loop goes as follows. First, an iteration over every element in the training set is performed. If an example is found to violate the KKT conditions then it is immediately selected for optimization, and the program enters the inner loop. After the outer loop completes an entire sweep over all examples, the next sweep proceeds only with those examples in the training set which have non-bounded Lagrange multipliers ($0 \leq \alpha_i \leq C$). Again each example is checked against the KKT conditions and violating examples are eligible for optimization. The program loops repeatedly over all non-bound examples until they all obey the KKT conditions within a certain pre-defined tolerance. When this goal is achieved, the outer loop once again sweeps over the entire training set, and this process is repeated until a sweep through the entire training set yields no examples that violates the KKT conditions (within tolerances), upon which the algorithm terminates.

Within the inner loop, the second SMO heuristic chooses which example to select as the second Lagrange multiplier to optimize. This is done by selecting the example which gives the largest step during the optimization, or in other words the example that maximizes $|E_1 - E_2|$. However, the process of calculating the step is a time consuming process involving kernel function evaluations. Thus an improvement was suggested by Platt [7] whereby the error values ($E1, E2$) are stored in an *error cache* and read back when necessary. The algorithm also includes a secondary method of selection, in the event that the maximum step heuristic is unable to make positive progress, leading to a selection that brings no optimization. This method involves first iterating through all the non-bound examples until one is found that leads to positive progress. If this method also fails, then an iteration is performed over all examples in search of suitable data. If this heuristic fails as well, then the example selected by the outer loop is skipped and another example is selected. It should be noted that the starting point of both iterations is randomized. This is to done in order to avoid biasing the classifier toward to the training examples at the beginning of the set.

In an effort to further improve the performance of the SMO algorithm, we added a *kernel cache* as well. The caching mechanism keeps in memory the results of kernel computations which are later retrieved as needed. This optimization has a notable positive impact on performance, although no empirical measurements were made.

Once the SMO has completed, the values of all the Lagrange multipliers (α_i) are known, and the decision boundary for classification can be determined as follows:

$$f(x) = \sum_{i=1}^{n} y_i \alpha_i K(x, x_i) + b = 0. \tag{9}$$

In the linear case, $K(x, x_i) = (x \cdot x_i)$ and the weights needed for classification and to find the margin can be determined as follows:

$$w = \sum_{i=1}^{n} y_i \alpha_i y_i x_i. \tag{10}$$

B. Implementation

For the signature verification system, we implemented linear and non-linear SVM classifiers with soft margins using the SMO algorithm.

For the non-linear case we implemented a Radial Basis Function (RBF) and used it as a kernel function. This function is well known to constitute a valid kernel whose output represents the result of a dot product in an otherwise unknown infinite dimensional space. In fact, it is simply a Gaussian function that behaves like a similarity function. In the RBF kernel equation below one can see that the exponential function is at its maximum of 1.0 only if the two points are the same:

$$K(x_i, x_j) = \exp\left(\frac{-|x_i - x_j|^2}{\sigma^2}\right) = \exp(-\gamma |x_i - x_j|^2). \tag{11}$$

It was proven as well that the linear kernel with parameter C is in fact just a special case of the RBF kernel for given values of C and γ. The reason we only used the RBF kernel as non-linear kernel is because the number of parameters required by the RBF kernel (C, γ), which influences the complexity of model selection during validation, are less than the other choices of non-linear kernels.

Finally, for numerical stability within the SMO algorithm, we normalized all components of the feature vector so that their values are between 0.0 and 1.0.

We compared the results obtained from our SMO implementation of SVM with SVM^{light}, a free software implementation of SVM. We compared the results for both the linear and the RBF kernels and they were identical. This confirmed that our SMO implementation of the SVM algorithm was working correctly and that it could be used for the signature verification task.

IV. Testing

A. Testing Procedure

We devised a testing scheme as detailed by the flow chart of Fig. 7 in order to train optimal classifiers and test their

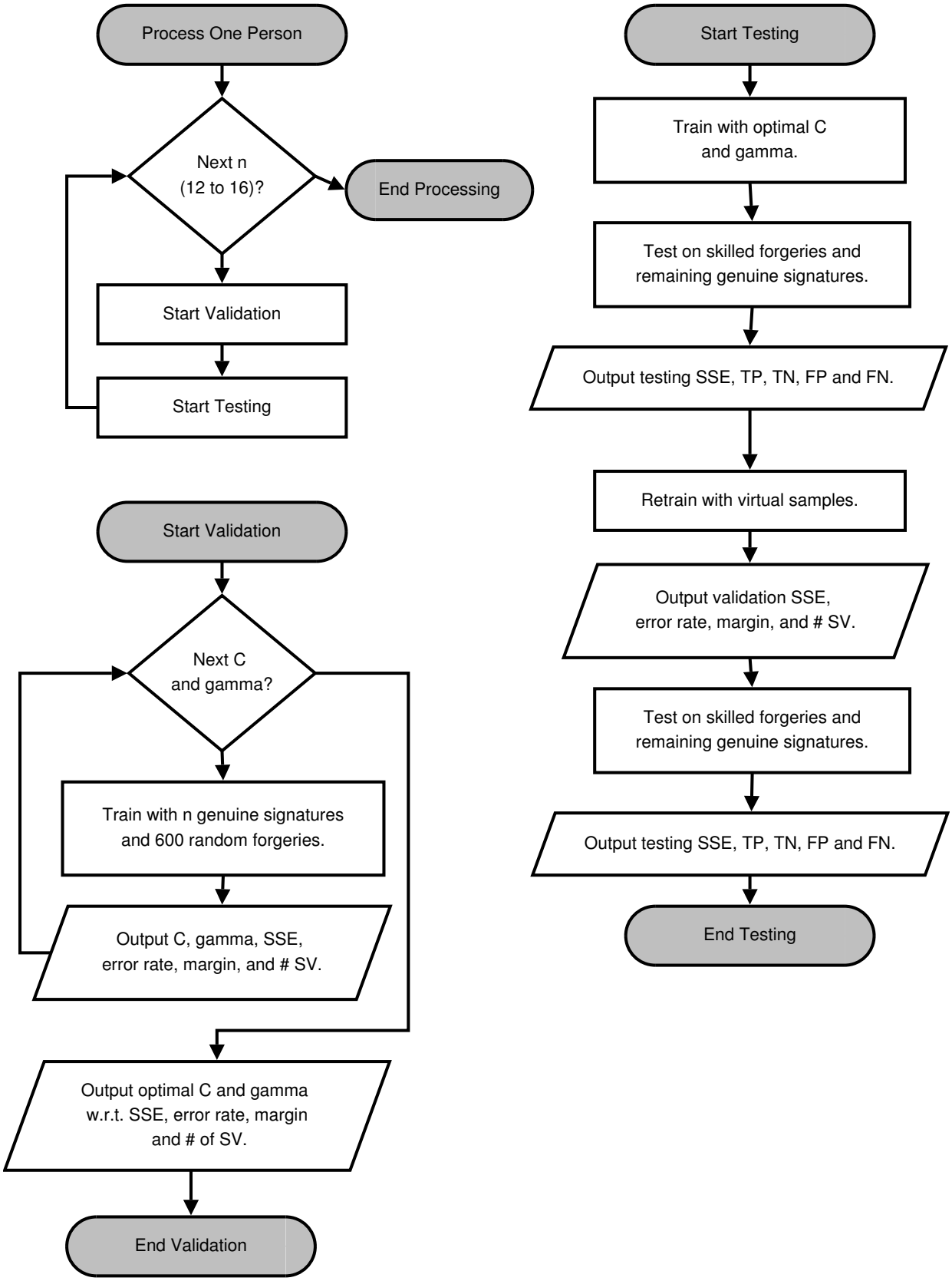

Fig. 7. Flowchart of the full testing procedure for one person.

performances. Summarily, for each person, the program trains the classifier with a varying number (from 12 to 16) of genuine signatures and with a high number (600) of randomly chosen signatures from the rest of the database. As validation, a leave-one-out cross-validation is performed over a range of values for the parameter C in the case of the linear kernel, and for the parameters C and γ, in the case of the RBF kernel. We decided not to use the polynomial kernel since it would require validation of 4 parameters versus only 2 for the RBF. It is probable as well that an RBF kernel with given C and γ is equivalent to a polynomial kernel with given parameters, although no proof of this exists. Also, it has been observed that RBF kernels generally outperform polynomial kernels [8].

Once the optimal kernel and its optimal parameters are known, the program trains the classifier with the full set of examples, with and without VSVM. The test results of both classifiers trained and validated using the same genuine signatures and random forgeries are compared.

Note that in the field of signature verification, it is standard to train only with random forgeries and not with skilled forgeries. This is because skilled forgeries are generally hard to obtain, and any practical system built to verify signatures will need to properly classify out skilled forgeries without access to any skilled forgeries during the training phase.

B. Test Results

The testing procedure described in the previous section is quite involved. In order to capture the highest number of differences between a given person's signatures and everyone else's signatures, it is desirable to train the classifier using a lot of random forgeries (between 200 and 600). Training a classifier on that many examples with a feature vector of about 3000 dimensions is quite computationally demanding, especially in the case of the RBF kernel. The performance of our SMO implementation of was not good enough to proceed through the validation phase in order to find the optimal kernel and its optimal parameters (only C for the linear kernel, and C and γ for the RBF kernel). Therefore, we decided to present here complete test results using a linear kernel with fixed parameter $C = 1024$ as suggested by Justino et al. [1]. The results are shown in Figures 8, 9, and 10, where the error bars indicate the standard deviations and the red dots, the minima and maxima. The 75 DPI database used as input data was taken from work done by Ferrer et al. [9]. This database contains 160 people, 24 genuine signatures and 30 skilled forgeries per person, which were all used for our tests. For the training of a given person, 600 randomly chosen signatures from the rest of the database were used as random forgeries.

The statistics that best describe our results are: the minimum, maximum and average error rate (ER), false rejection rate (FRR or error type I), and false acceptance rate (FAR or error type II) over all persons:

$$\begin{aligned} ER &= \frac{FP+FN}{TP+TN+FP+FN} \\ FRR &= \frac{FN}{FN+TP} \\ FAR &= \frac{FP}{FP+TN} \end{aligned}$$

where TP, TN, FP, and FN are the number of true positives, true negatives, false positives and false negatives respectively.

The abscissa indicates the number of genuine signatures used for training, between 12 and 16. This range was chosen since the database contains only 24 genuine signatures per person. Anything lower than 12 would most likely, statistically, not produce a low FRR. Also we wanted to keep at least 8 signatures for testing. As can be seen on Fig. 8 the average error rate is of about 14% for all cases, but the extrema go from 0% and to almost 50%. Anything higher than 50% would indicate a serious problem with our algorithm since it would indicate that our classifiers were worse than a classifier randomly choosing the validity of a signature. From Fig. 9, the false rejection rate is of around 25% and slightly decreases with the number of genuine signatures used during training, as expected. Again, this statistic has a wide range of values, from 0% to 100%, indicating that for some very rare signatures, our classifier could not recall at all signatures from the person for which it was trained. Next, the false acceptance rate of Fig. 10 is of about 11% and slightly increases with the number of genuine signatures used for training, also as expected, since using more signatures for training can only either leave the decision plane unchanged or push it toward the forgery area of the space. The variability of this rate ranges from 0% to about 63%, indicating that such a system could not be trusted without human supervision for managing access to critical data.

For the second part of the testing procedure, the classifier was retrained with virtual signatures generated as described in the image processing section, and its performance was evaluated. As one can see from the results in figures 11, 12 and 13, the error rate slighly increases by 3%, the FRR of the virtually trained SVM classifiers is better (17%) than the one trained without (25%), but the FAR is higher (18% versus 11%).

However, these results were obtained with a linear kernel. At the time of this writing, the performance of our SMO implementation was not good enough to run a full parameter grid search [8] and validate the parameters C and γ of the RBF kernel. However, in the future, with a better optimized SVM implementation such as SVM^{light}, we plan on finding optimal RBF classifiers using the leave-one-out cross-validation procedure for each person, as described in the previous section, and compare the results with and without virtual examples. Also, invariant transformations other than rotation and translation, and better suited to signature verification might exist and should be investigated.

Although we did not use validation on the classifiers obtained and used for the tests above, we still wanted to show some validation results. So we used leave-one-out cross-validation with 16 genuine signatures and 200 random forgeries on the first 10 people in the database, and tested on the remaining 8 genuine signatures and the 30 skilled forgeries. We pitted against each other a linear kernel with a fixed $C = 1024$ and a linear kernel with a C chosen for each person on the basis of best error rate and sum of square errors (SSE), in that order. The results are given in Table I, and as one can see, the results obtained with validation are slightly

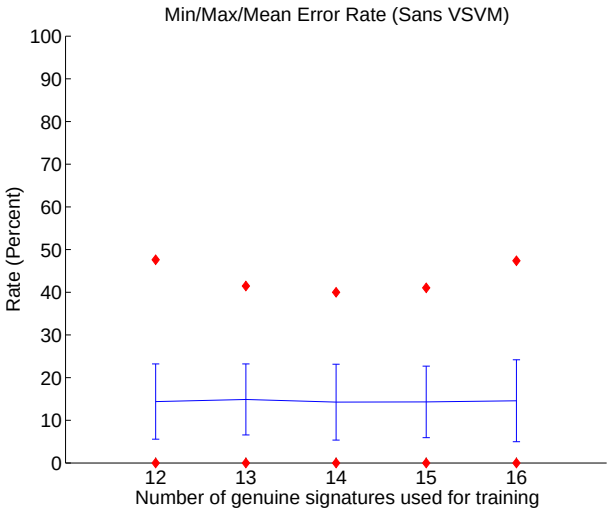

Fig. 8. Plot of min/max/mean error rate without VSVM.

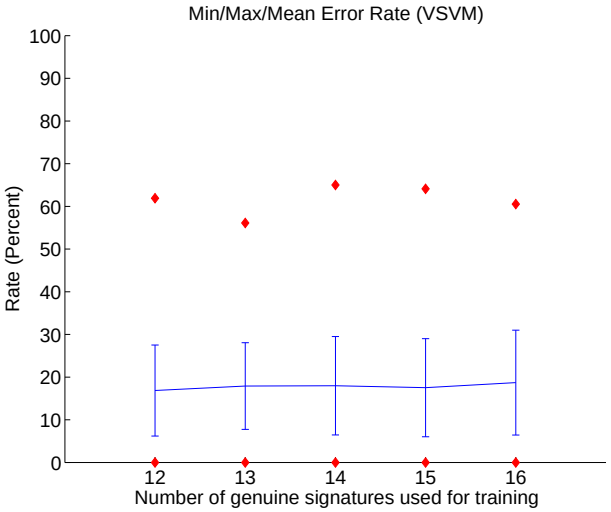

Fig. 11. Plot of min/max/mean error rate with VSVM.

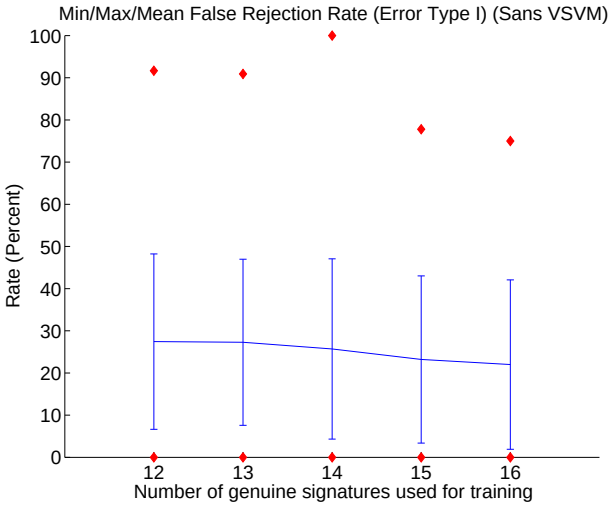

Fig. 9. Plot of min/max/mean false rejection rate (error type I) without VSVM.

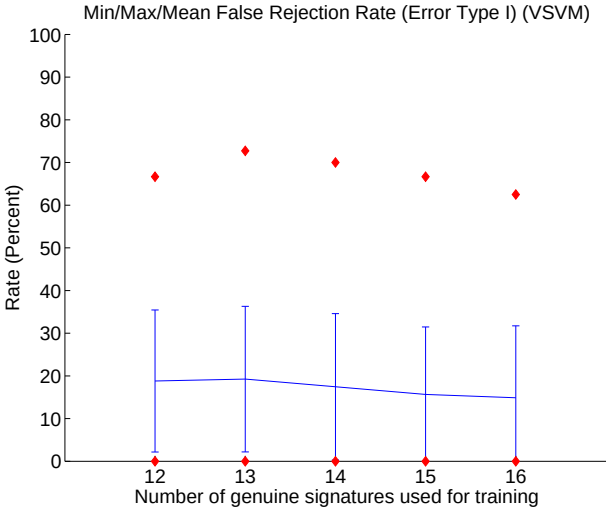

Fig. 12. Plot of min/max/mean false rejection rate (error type I) with VSVM.

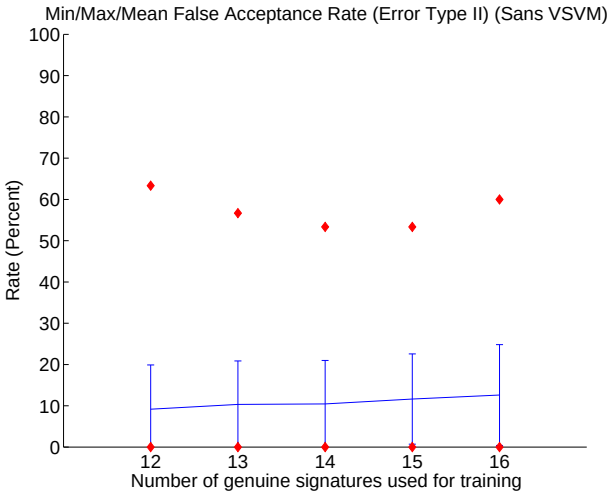

Fig. 10. Plot of min/max/mean false acceptance rate (error type II) without VSVM.

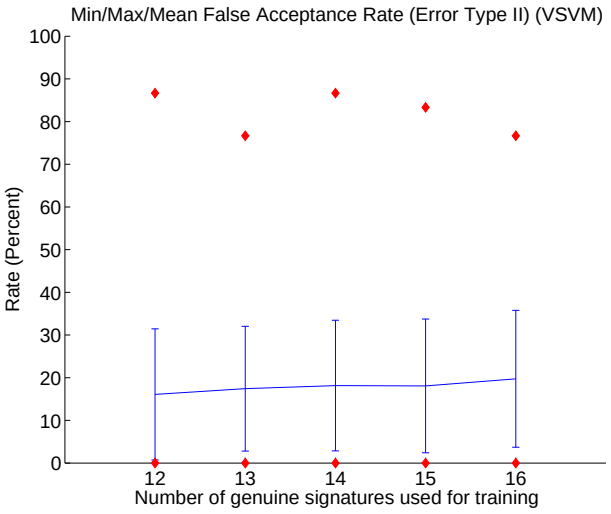

Fig. 13. Plot of min/max/mean false acceptance rate (error type II) with VSVM.

TABLE I
AVERAGE TEST RESULTS FOR 10 PERSONS USING A LINEAR KERNEL WITH 16 GENUINE SIGNATURES AND 200 RANDOM FORGERIES FOR TRAINING.

	Fixed Parameter $C = 1024$	**Per Person Validated** C
Margin Width	2.99	3.04
SSE	19.1	17.3
Error Rate	15.3%	15.0%
FAR	15%	16%
FRR	15%	13%
Min False Rejection	-0.20	-0.13
Max False Acceptance	0.41	0.37

better overall. We would probably get even better results using an RBF kernel, but in order to get any meaningful results, we would need to perform a grid search on both C and γ parameters. Our current implementation is not fast enough for this purpose. Note also that with the knowledge of the maximum distance from the decision plane to the misclassified forgeries (for example 0.41 for $C = 1024$), a threshold value can be devised in order to minimize the false acceptance rate at the expense of the false rejection rate or vice versa if desired.

V. CONCLUSION

As can be seen from the results, we were able to extract decent performance from our signature verification system using SVM classifiers. The average error rate on the database used was of about 14%, although the extrema vary from about 0% to almost 50%. Also the false acceptance rate, the most serious error of biometric errors, is lower ($\sim$11%) than the false rejection rate ($\sim$25%) indicating that the system automatically seems to "understand" the fact that the first type of error is of greater importance, although VSVM does not seem to carry over this characteristic as it seems to favor the reduction of the FRR at the expense of the FAR.

As future work, the testing procedure as well as the SVM algorithm need to be refined and improved in order to properly validate the choice of kernel and the values of its parameters. Also, since Virtual SVM showed promising results, it would be of great research interest to study the effect of virtual signatures on signature verification in greater details, since we believe this idea has great potential. Among others, different invariant transformations should be studied and the RBF kernel should be used.

ACKNOWLEDGMENT

The authors would like to thank Yon Visell for his help understanding SVM and for all the material learned in class.

REFERENCES

[1] E. J. R. Justino, F. Bortolozzi, and R. Sabourin, "A comparison of svm and hmm classifiers in the off-line signature verification," *Pattern Recogn. Lett.*, vol. 26, no. 9, pp. 1377–1385, 2005.

[2] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, "Hypermedia Image Processing Reference (HIPR2)," 2004, http://homepages.inf.ed.ac.uk/rbf/HIPR2/.

[3] B. Schölkopf, C. Burges, and V. Vapnik, "Incorporating Invariances in Support Vector Learning Machines," in *ICANN 96: Proceedings of the 1996 International Conference on Artificial Neural Networks*, vol. 1112. London, UK: Springer-Verlag, 1996, pp. 47–52.

[4] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Min. Knowl. Discov.*, vol. 2, no. 2, pp. 121–167, 1998.

[5] A. W. Moore, "Support Vector Machines, Tutorial Slides," 2001, http://www.autonlab.org/tutorials/svm.html.

[6] V. Vapnik, *Estimation of Dependences Based on Empirical Data*. New York, USA: Springer Verlag, 1982.

[7] J. C. Platt, "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines," Microsoft Research, Tech. Rep. MSR-TR-98-14, 1998.

[8] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A Practical Guide to Support Vector Classification," Department of Computer Science and Information Technology, National Taiwan University, Tech. Rep., 2003. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[9] M. A. Ferrer, J. B. Alonso, and C. M. Travieso, "Offline Geometric Parameters for Automatic Signature Verification Using Fixed-Point Arithmetic," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 6, pp. 993–997, 2005.