Summer Internship Report

School of Computing and Electrical Engineering

IIT Mandi

Deep Chakraborty
B.Tech 3rd year, Dept. of ECE,
Manipal Institute of Technology
KA - 576104
`deepc94@gmail.com`

July 18, 2015

Objective

The objective of this internship was to learn and understand the working of Support Vector Machines for Pattern Classification, use SVMTorch[1]; a library for Large-Scale Regression and Classification problems on sample datasets and then extending its functionality to incorporate user defined Kernel Gram matrices for training and testing on real world datasets such as Emotion Recognition in Speech using Class Independent GMM based Intermediate Matching Kernel.

Acknowledgements

First and foremost, I would like to express immense gratitude towards my research advisor Dr. Dileep A. D. for considering me eligible for this project and for his constant motivation, support and unfathomable patience during difficult stretches. This project wouldn't have been a success if it wasn't for his guidance. I truly couldn't have imagined a better mentor than him.

Besides my advisor, I would like to thank Nikhil Choubey, my fellow intern for hours of doubt clearing sessions in programming and other concepts and stimulating discussions, and the sleepless nights we were working together before deadlines. He has been an amazing companion to work with. Last but not the least, I would like to thank the entire IIT Mandi administration and non-teaching staff for making my stay here an unforgettable experience.

Contents

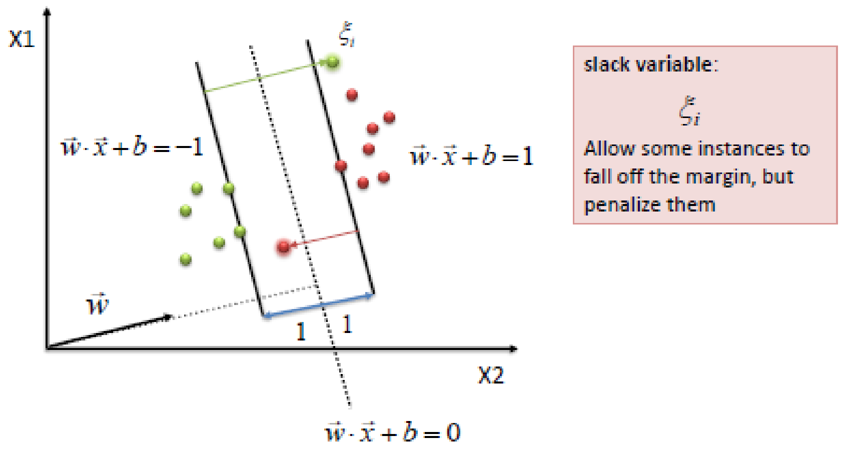
Chapter 1

Introduction

1.1 Support Vector Machines

Support vector machines[2] (SVM) are a group of supervised learning methods that can be applied to classification or regression. The idea behind SVMs is to make use of a (nonlinear) mapping function Φ that transforms data in input space to data in feature space in such a way as to render a problem linearly separable. The SVM then automatically discovers the optimal separating hyperplane (which, when mapped back into input space via Φ^{-1}, can be a complex decision surface). The function Φ is realized by using a Mercer kernel such that $K(\mathbf{x}, \mathbf{z}) =< \Phi(\mathbf{x}), \Phi(\mathbf{z}) >$. This is also known as the kernel trick.

Suppose the training data set consists of m examples, $\{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$ where $\mathbf{x}_i$ is ith training example and y_i is the corresponding class label. Figure 1.1 illustrates the construction of an optimal separating hyperplane for linearly separable classes in the two-dimensional input space of x. A hyperplane is specified as $\mathbf{w}^T\mathbf{x} + b = 0$, where $\mathbf{w}$ is the parameter vector and b is the bias. The examples with class label $y_i = +1$ are the data points lying on the positive side of the hyperplane and the examples with class label $y_i = -1$ are the data points lying on the negative side of the hyperplane.



X1
ξ_i
$\vec{w}\cdot\vec{x}+b=-1$
$\vec{w}\cdot\vec{x}+b=1$
$\vec{w}$
1
1
X2
$\vec{w}\cdot\vec{x}+b=0$
slack variable:
ξ_i
Allow some instances to fall off the margin, but penalize them

Figure 1.1: Illustration of constructing the optimal separating hyperplane for linearly separable data.

1.2 Software

1.2.1 SVMTorch II

SVMTorch II is an implementation of Vapnik's Support Vector Machine that works both for classification and regression problems, and that has been specifically tailored for large-scale problems (such as more than 20000 examples, even for input dimensions higher than 100).[1] It was written by Ronan Collobert and Samy Bengio at IDIAP, Switzerland. It's salient features are:

- The code has been written in C++. It's clean and optimized for a chunk of size 2.
- The optimality check is similar to SMO. (Fast)
- Shrinking heuristic in optimality check
- Supports sparse and dense input file formats.
- Better than Nodelib, SVMlight[3] and LIBSVM[4] (MATLAB)

1.2.2 MATLAB

MATLAB is the high-level language and interactive environment used by millions of engineers and scientists worldwide. It lets you explore and visualize ideas and collaborate across disciplines including signal and image processing, communications, control systems, and computational finance. In our case, it has been extensively used for preparing the datasets to run the SVM on, plotting the decision regions generated by the SVM, preparing Kernel Gram matrices, etc.

1.2.3 Sublime Text

Sublime Text is a sophisticated text editor for code, markup and prose. It's salient features are code colorization, code Folding, auto-completion, etc. and its cross platform

1.2.4 LaTeX

LaTeXis a high-quality typesetting system; it includes features designed for the production of technical and scientific documentation. LaTeXis the de facto standard for the communication and publication of scientific documents. LaTeXis available as free software.

1.2.5 GNU Compiler Collection (GCC)

The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran, Java, Ada, and Go, as well as libraries for these languages (libstdc++, libgcj,...). GCC was originally written as the compiler for the GNU operating system. The GNU system was developed to be 100% free software.

Chapter 2

Getting familiar with SVMTorch

The objective of this study was to experiment with different parameters for support vector machines and kernels such as Linear kernel, Polynomial kernel and Gaussian kernel on the given dataset using the SVMTorch[1] library. MATLAB has been used intermittently for preparing the datasets and plotting the results. As a rule of the thumb, 75% of the given dataset was used as the training set whereas the rest 25% was used as the test set to determine the classification accuracy. The procedure followed is as follows:

1. Train the SVM using the Training set using SVMTorch using different parameters each time
2. Test the generated model on the Test set to yield the training accuracy and number of support vectors
3. Find the discriminant values for a set of 1000×1000 points that make up the plotting space
4. Find which class these points belong to and assign them a specific colour
5. Observe the generated decision region and make conclusion about the effect of varying the parameters on it

The SVMTorch commands used were:

- For Training:
  ```
  ./SVMTorch [-multi -t [0 1 2] -c #] <training_file> <target_model_file>
  ```
- For testing:
  ```
  ./SVMTest [-multi] <model_file> <test_file>
  ```
- For Plotting:
  ```
  ./SVMTest -no -oa <ascii_model_file> <model_file> <grid_test_file>
  ```

2.1 Given Datasets

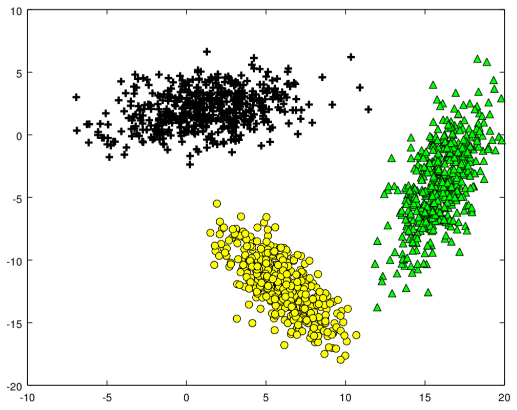


Figure 2.1: Linearly separable data

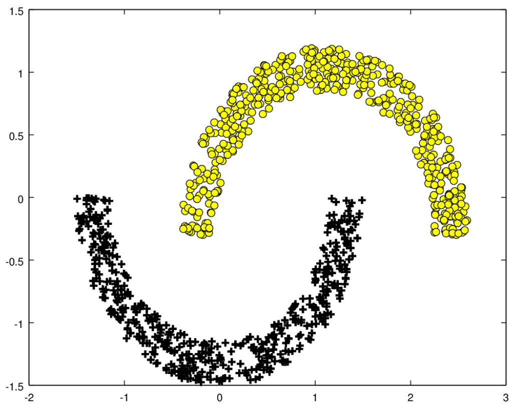


Figure 2.2: Interlock data

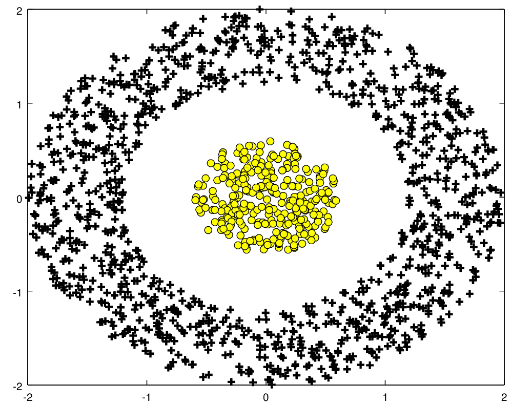


Figure 2.3: Ring data

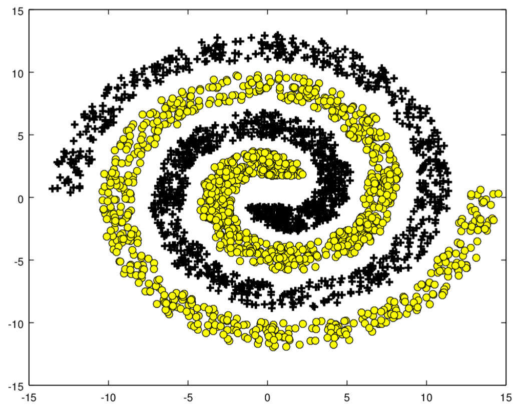


Figure 2.4: Spiral data

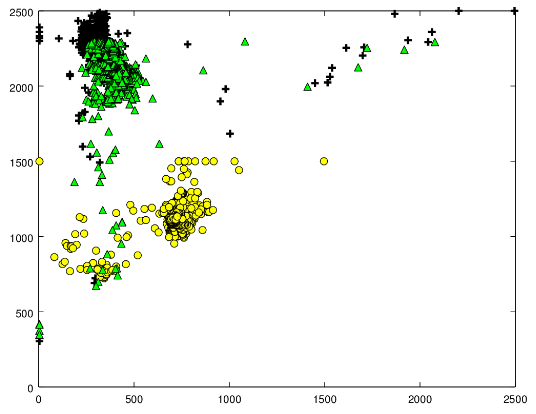


Figure 2.5: Real World Data

2.2 Using Gaussian Kernel

$$K(\mathbf{a}, \mathbf{b}) = \exp\left(-\frac{||\mathbf{a} - \mathbf{b}||}{\sigma^2}\right)$$

2.2.1 Linearly Seperable Data with 3 classes

1125 examples in the training set
375 examples in the test set

Table 2.1:

std	# support vectors	# false positives	# false negatives	Total misclassified	Correctly classified
0.1	1112	0, 0, 0	91, 82, 94	98 (26.13%)	277 (73.87%)
0.3	925	0, 0, 0	16, 16, 16	14 (3.733%)	361(96.267%)
1.0	393	0, 0, 0	0, 0, 1	0 (0%)	375(100%)
3.0	60	0, 0, 0	0, 0, 0	0 (0%)	375(100%)
10.0	7	0, 0, 0	0, 0, 0	0 (0%)	375(100%)
20.0	5	0, 0, 0	0, 0, 0	0 (0%)	375(100%)
60.0	12	0, 0, 0	0, 0, 0	0 (0%)	375(100%)
100.0	21	0, 1, 0	0, 0, 0	0 (0%)	375(100%)

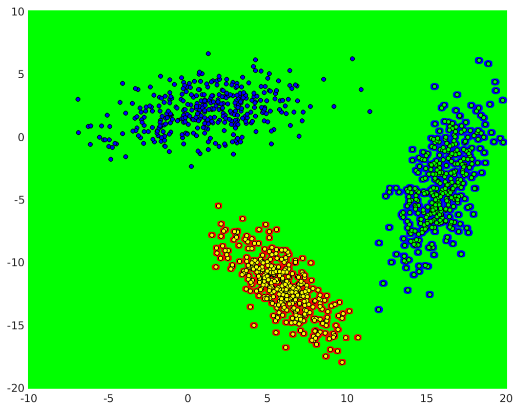


Figure 2.6: std=0.1

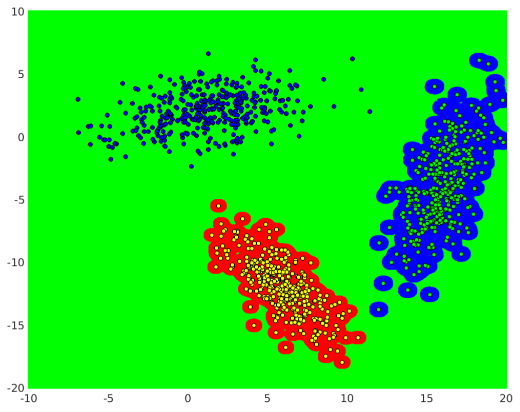


Figure 2.7: std=0.3

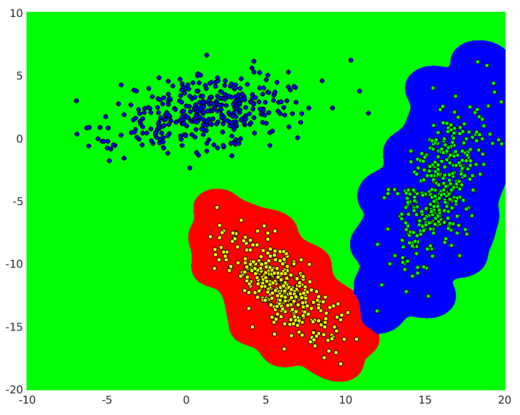


Figure 2.8: std=1.0

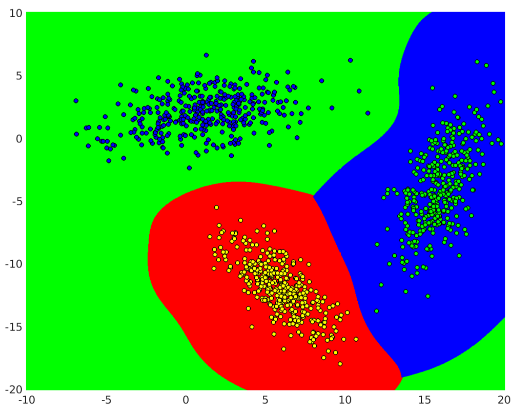


Figure 2.9: std=3.0

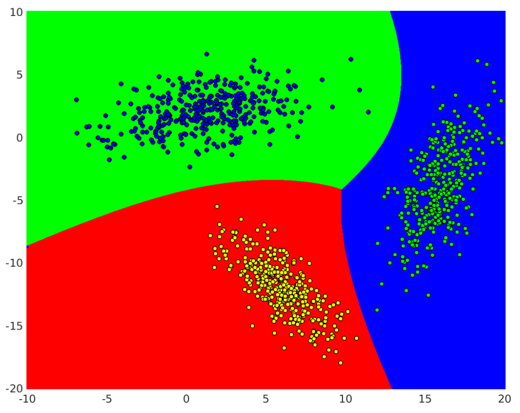


Figure 2.10: std=10.0

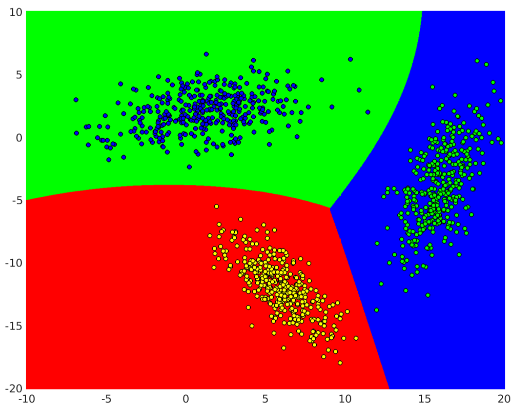


Figure 2.11: std=20.0

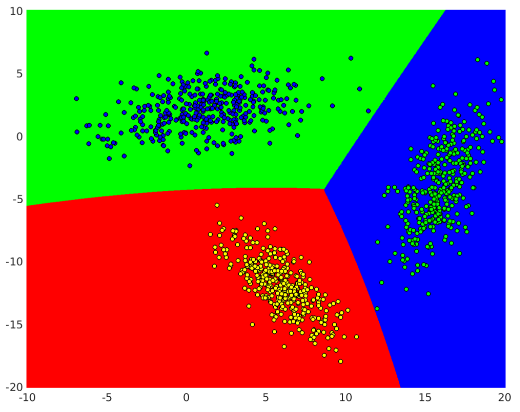


Figure 2.12: std=60.0

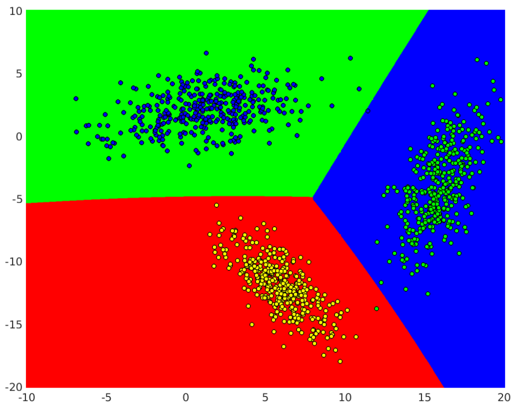


Figure 2.13: std=100.0

2.2.2 Interlocking Data

750 examples in the training set
250 examples in the test set

Table 2.2:

std	# support vectors	# false positives	# false negatives	Total misclassified	Correctly classified
0.03	708	0	4	4(1.6%)	246(98.4%)
0.1	370	0	0	0(0%)	250(100%)
0.3	75	0	0	0(0%)	250(100%)
1.0	11	0	0	0(0%)	250(100%)
3.0	14	0	0	0(0%)	250(100%)
10.0	112	8	4	12(4.8%)	238(95.2%)

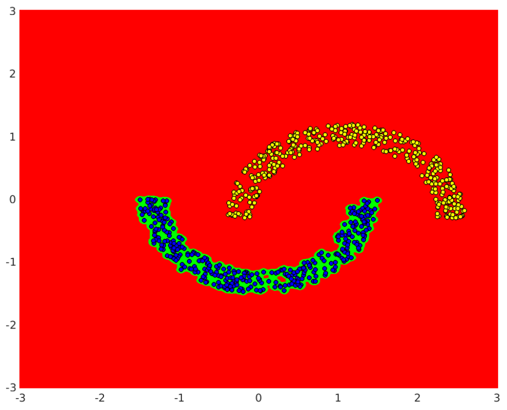


Figure 2.14: std=0.03

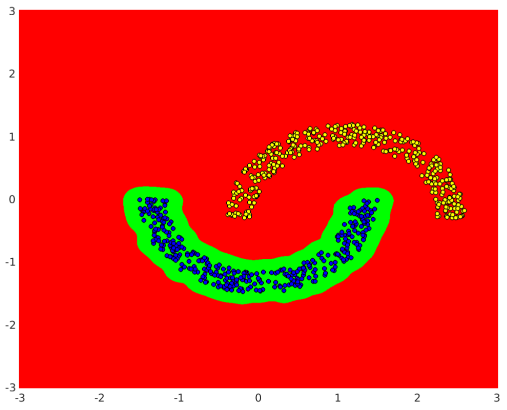


Figure 2.15: std=0.1

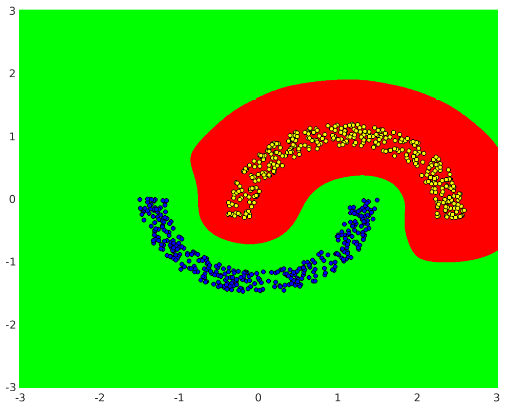


Figure 2.16: std=0.3

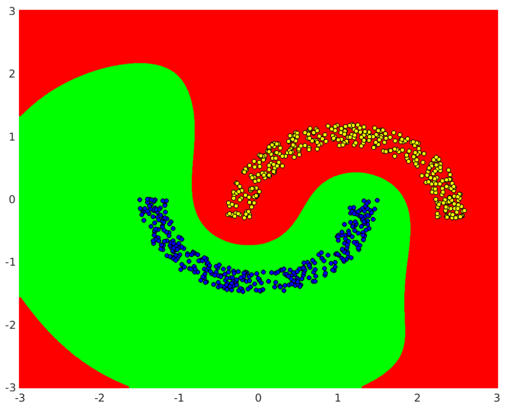


Figure 2.17: std=1.0

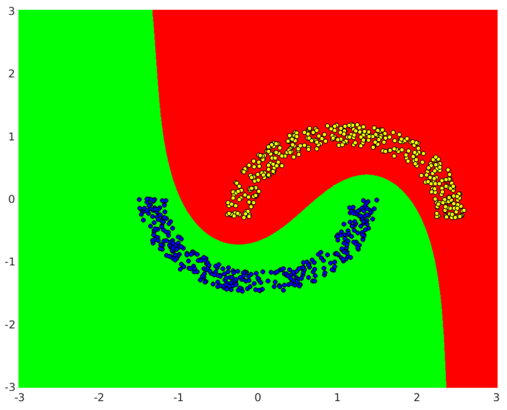


Figure 2.18: std=3.0

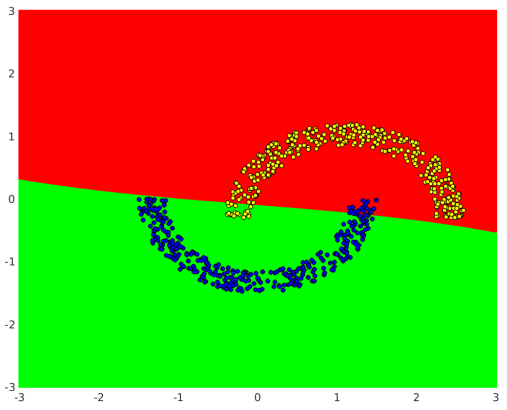


Figure 2.19: std=10.0

2.2.3 Ring Data

1125 examples is training set
375 examples in test set

Table 2.3:

std	# support vectors	# false positives	# false negatives	Total misclassified	Correctly classified
0.1	754	1	0	1(0.2667%)	374(99.73%)
0.3	203	0	0	0(0%)	375(100%)
1.0	19	0	0	0(0%)	375(100%)
3.0	11	0	0	0(0%)	375(100%)
10.0	129	0	0	0(0%)	375(100%)

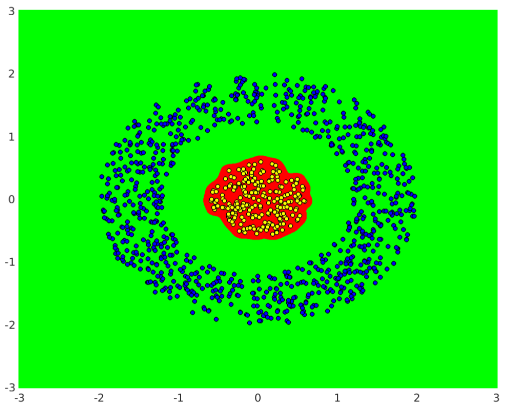


Figure 2.20: std=0.1

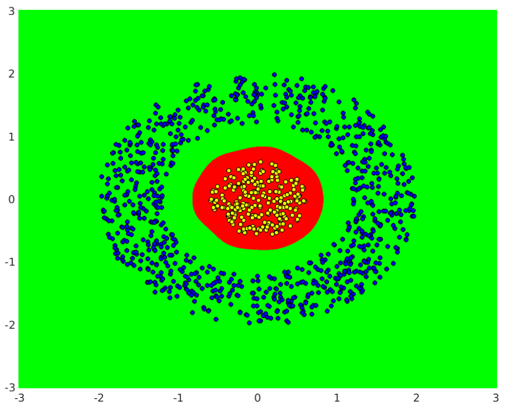


Figure 2.21: std=0.3

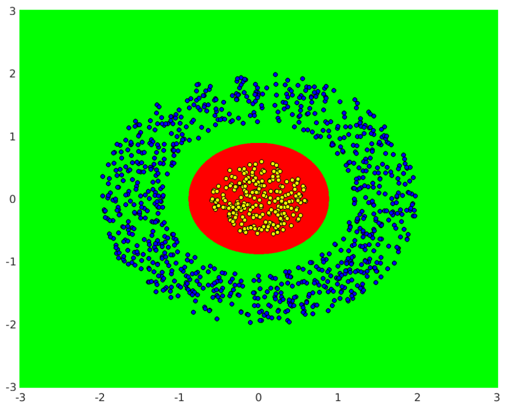


Figure 2.22: std=1.0

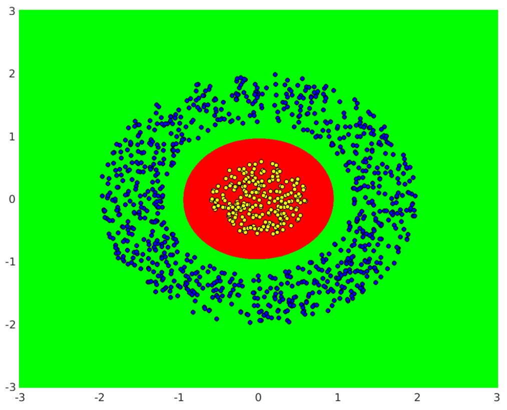


Figure 2.23: std=3.0

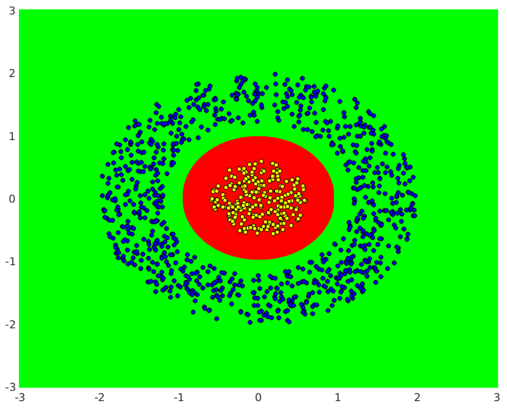


Figure 2.24: std=10.0

2.2.4 Spiral Data

1955 examples is training set
652 examples in test set

Table 2.4:

std	# support vectors	# false positives	# false negatives	Total misclassified	Correctly classified
0.1	1931	97	1	98(15.03%)	554(84.97%)
0.3	1601	0	1	1(0.1534%)	651(99.85%)
1.0	415	0	1	1(0.1534%)	651(99.85%)
3.0	90	0	1	1(0.1534%)	651(99.85%)
10.0	1260	60	48	108(16.56%)	544(83.44%)

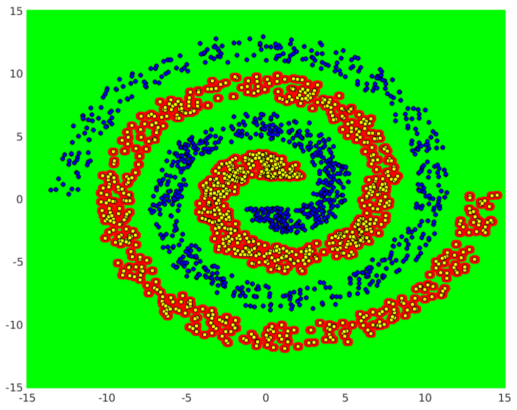


Figure 2.25: std=0.1

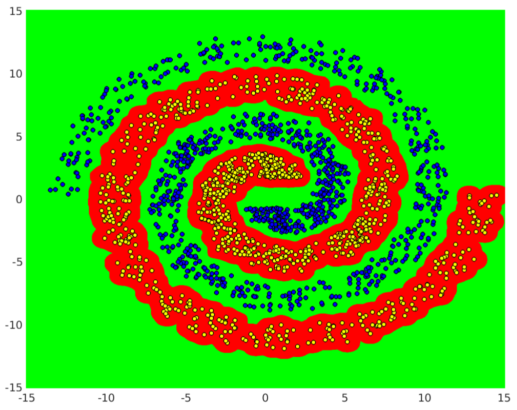


Figure 2.26: std=0.3

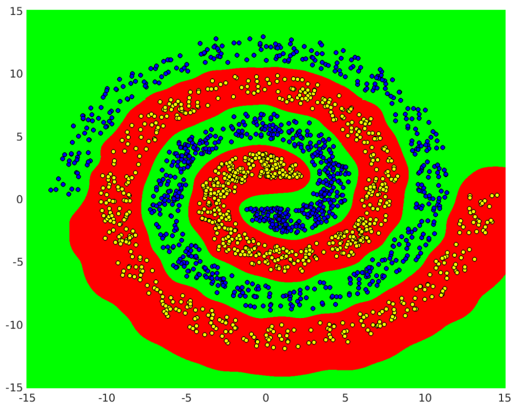


Figure 2.27: std=1.0

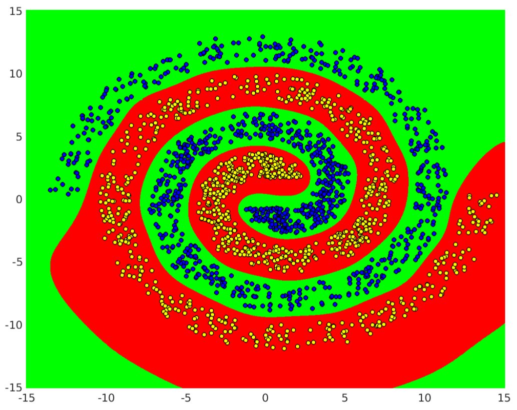


Figure 2.28: std=3.0

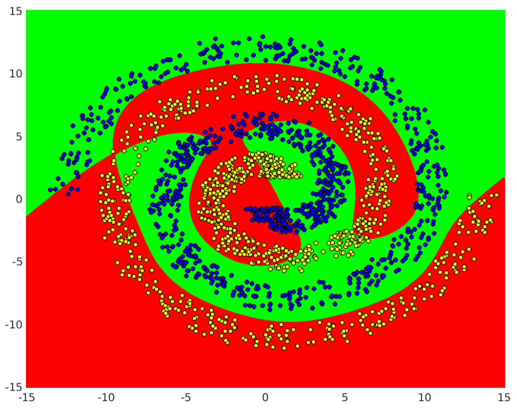


Figure 2.29: std=10.0

2.2.5 Real World Data with 3 classes

5132 examples is training set
1711 examples in test set

Table 2.5:

std	# support vectors	# false positives	# false negatives	Total misclassified	Correctly classified
150	452, 92, 460	40, 2, 23	22, 2, 43	66(3.857%)	1645(96.143%)

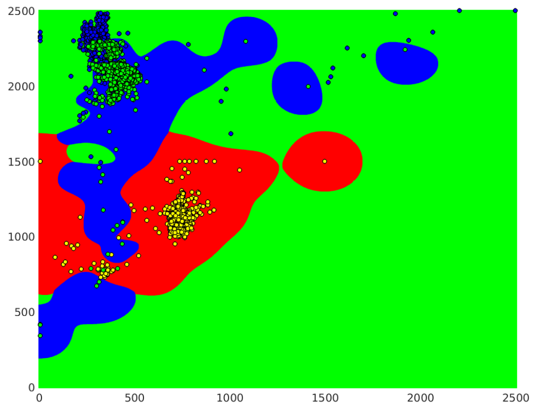


Figure 2.30: best case std=150

2.2.6 Conclusion

It is observed that selection of a very small value of std. (standard deviation) or σ for the gaussian kernel (≤ 0.1) leads to high variance or overfitting of the training set and fails to generalize well to new test examples. On the other hand, selection of a very high value of sigma (≥ 10.0) leads to high bias or underfitting of the training set. The selection of an optimum sigma, and in effect an optimum decision boundary results in minimum number of support vectors and a decision region with smooth edges.

2.3 Using Polynomial Kernel

$$K(\mathbf{a}, \mathbf{b}) = (S\mathbf{a}^T\mathbf{b} + R)^D$$

2.3.1 Linearly Separable Data with 3 classes

Table 2.6:

parameters	# support vectors	# false positives	# false negatives	Total misclassified	Correctly classified
S=1, R=1, D=2	5, 4, 4	0, 0, 0	0, 0, 0	0(0%)	375(100
S=1, R=1, D=3	5, 6, 4	0, 0, 0	0, 0, 0	0(0%)	375(100%)
S=1, R=1, D=4	7, 6, 5	0, 0, 0	0, 0, 0	0(0%)	375(100%)

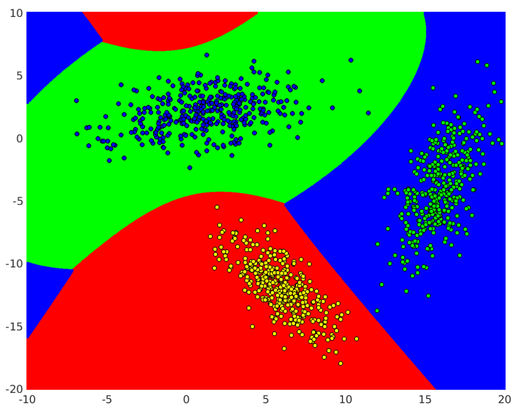


Figure 2.31: S=1, R=1, D=2

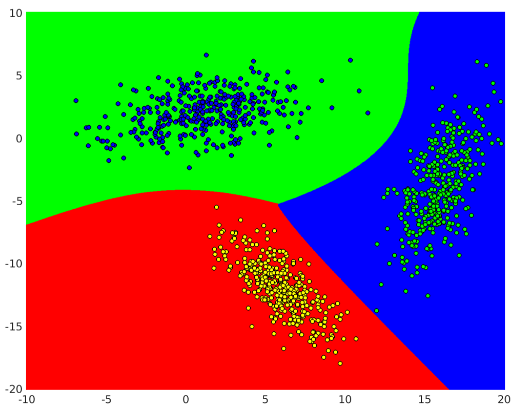


Figure 2.32: S=1, R=1, D=3

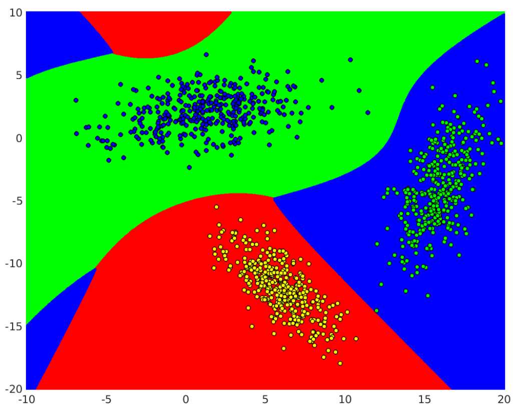


Figure 2.33: S=1, R=1, D=4

2.3.2 Interlock Data

Table 2.7:

parameters	# support vectors	# false positives	# false negatives	Total misclassified	Correctly classified
S=1, R=1, D=2	93	8	5	13(5.2%)	237(94.8%)
S=1, R=1, D=3	12	0	0	0(0%)	250(100%)
S=1, R=1, D=4	8	0	0	0(0%)	250(100%)

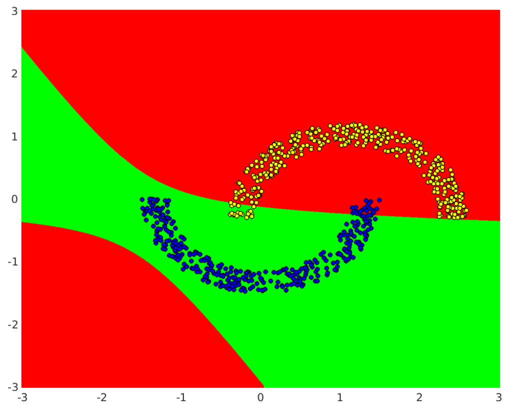


Figure 2.34: S=1, R=1, D=2

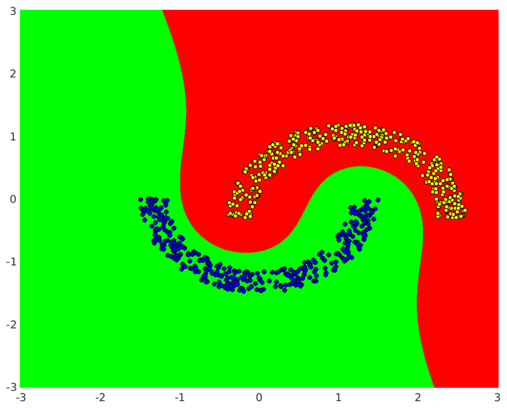


Figure 2.35: S=1, R=1, D=3

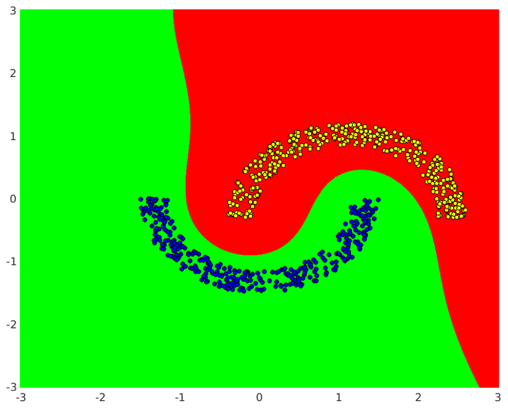


Figure 2.36: S=1, R=1, D=4

2.3.3 Ring Data

Table 2.8:

parameters	# support vectors	# false positives	# false negatives	Total misclassified	Correctly classified
S=1, R=1, D=2	8	0	0	0(0%)	375(100%)

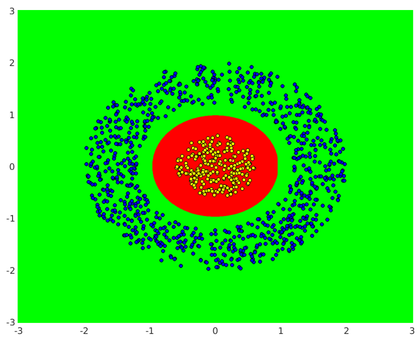


Figure 2.37: S=1, R=1, D=2

2.3.4 Spiral Data

Table 2.9:

parameters	# support vectors	# false positives	# false negatives	Total misclassified	Correctly classified
S=0.001, R=30, D=2	1617	152	144	296(45.4%)	356(54.6%)
S=0.001, R=30, D=3	1496	967	99	195(29.91%)	457(70.09%)
S=0.01, R=1, D = 5	1183	95	86	181(27.76%)	471(72.24%)

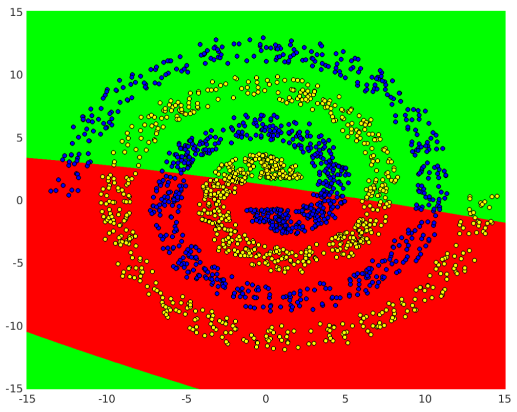


Figure 2.38: S=0.001, R=30, D=2

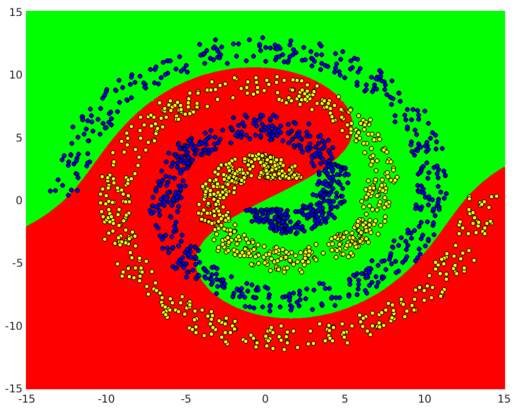


Figure 2.39: S=0.001, R=30, D=3

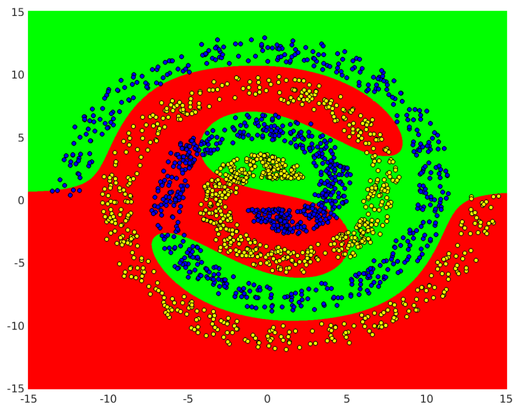


Figure 2.40: S=0.01, R=1, D = 5

2.3.5 Real World Data with 3 classes

Convergence could not be achieved for Real World data using Polynomial Kernel.

2.4 Using Linear Kernel

$$K(\mathbf{a}, \mathbf{b}) = (S\mathbf{a}^T\mathbf{b})$$

2.4.1 Linearly Separable Data with 3 classes

Table 2.10:

parameters	# support vectors	# false positives	# false negatives	Total misclassified	Correctly classified
N/A	3	0, 1, 0	0, 1, 0	0(0%)	375(100%)

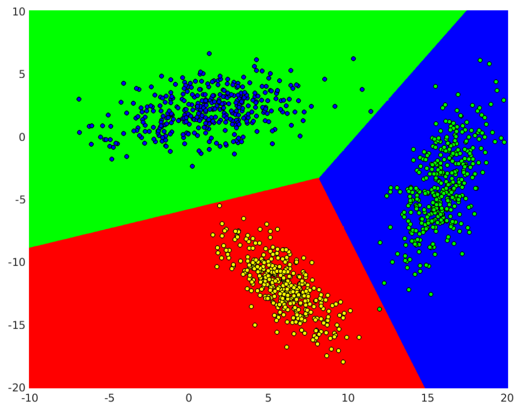


Figure 2.41: linear kernel

2.4.2 Interlocking Data

Table 2.11:

parameters	# support vectors	# false positives	# false negatives	Total misclassified	Correctly classified
N/A	95	8	4	12(4.8%)	238(95.2%)

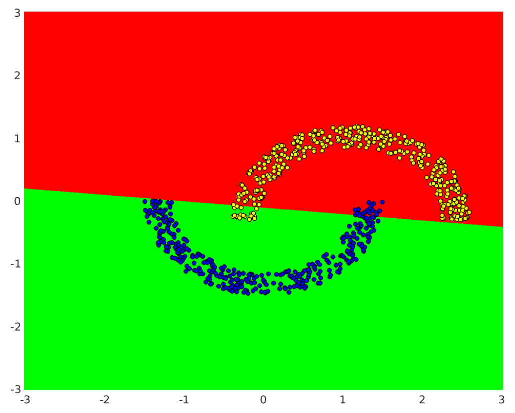


Figure 2.42: linear kernel

Chapter 3

SVMTorch for User Defined Kernels

The primary objective of this part is to enable the user to exploit the capabilities of SVMTorch for user-defined kernels by allowing them to load their own pre-calculated Kernel Gram matrices and then using SVMTorch for training the support vector machine and also testing it on the Test Kernel Gram matrices.

3.1 Parts of SVMTorch Source Code

The SVMTorch source code has the following important modules:

- `SVMTorch.cc:` The source file for running the SVM Training on the training set
- `SVMTest.cc:` The source file for testing the SVM model on the test set
- `StandardSVM.cc:` The source file responsible for the actual optimization algorithms behind the SVM training and also saving and loading the model file containing different parameters such as the support vectors, Lagrangian coefficient, etc.
- `IOTorch.cc:` The code responsible for the File IO operations such as loading the training/test set, etc.
- `Kernel.cc:` The code that contains all the standard kernels and returns the kernel value evaluated on a pair of examples when called
- `UserKernel.cc:` The part where the user must define his own kernel for use with SVMTorch

3.2 Modifications made to the Source Code

This section entails the modifications made to the different parts of the source code in detail to achieve the desired functionality.

3.2.1 Changes to `UserKernel.cc`

The equations governing training and testing of SVMs are independent of individual examples from the training set and are only dependent on the kernel value between a pair of examples at any given point. Therefore, instead of evaluating the kernel function defined in this file on a pair of examples loaded from the training set, it simply picks the corresponding kernel value of the desired pair of examples from the pre-calculated Kernel Gram matrix and returns it to the point where it is called in the training algorithm. Similarly during testing of the SVM, it simply returns those columns from the Test Kernel Gram matrix that correspond to the indices of the support vectors.

3.2.2 Changes to `StandardSVM.cc`

The following changes were made here:

- **the `StandardSVM::save()`** function:The content that is stored in the model file is now different for when user-kernel mode is opted for; in that during the use of any standard kernel, the model file stores the examples from the training set which are the support vectors itself. But when user-kernel mode is selected, it instead stores the indices of the support vectors rather than the support vectors itself. This is because during testing, `UserKernel.cc` must return the those columns from any given row (corresponding to a particular test example) of the Test Kernel Gram matrix which are the indices of the support vectors, as only these entries are responsible for the prediction of the class label. Additionally, the model file is now also stored in an ASCII format for the user to be able to read and debug the different parameters/coefficients of the generated model and make appropriate parameter/model selections. The user simply needs to look into `<model_file_name>_ascii.txt` to observe the generated model.

- **the `StandardSVM::load()` function:** Similarly, the way that the model file is loaded is also different for Standard kernels and user defined kernel mode, wherein it expects to load the support vector examples from the model file in the former case and only the indices of the support vectors in the latter.

- **the `StandardSVM::use()` function:** This function calls `Kernel::evaluate()` between a test example and support vector in Standard kernel case whereas in the User kernel case, it calls this function with a row from the Test Kernel Gram matrix (corresponding to a test example) and the index of the support vector.

3.3 Usage

The usage of SVMTorch is almost similar to the original version except for the change in the input file format in the case of User Kernel mode (`./SVMTorch -t 4 ...`).

In the case of training, the input file (Kernel Gram Matrix) should be in the following format:

Consider K_{ij} is the kernel function evaluated on $\{x_i, x_j\}$ pair of training examples or $\{z_i, x_j\}$ pair of test vs. training example. m is the number of training examples and n is the number of test examples. In standard classification, t_i is $+1$ or -1. In multiclass mode, the classes are numbered $0, 1, 2, ...$

- **Standard:** for `SVMTorch`

$$\begin{array}{cccc}
m & (m+1) & & \\
K_{11} & ... & K_{1m} & t_1 \\
 & . & & \\
 & . & & \\
 & . & & \\
K_{m1} & ... & K_{mm} & t_m
\end{array}$$

- **Standard:** for `SVMTest`

$$\begin{array}{cccc} n & (m+1) & & \\ K_{11} & ... & K_{1m} & t_1 \\ & . & & \\ & . & & \\ & . & & \\ K_{n1} & ... & K_{nm} & t_n \end{array}$$

- **Standard without target:** for `SVMTest` with `-no` option.

$$\begin{array}{ccc} n & m & \\ K_{11} & ... & K_{1m} \\ & . & \\ & . & \\ & . & \\ K_{n1} & ... & K_{nm} \end{array}$$

3.4 Results of SVMTorch using Class Independent GMM based Intermediate Matching Kernel

The following are the consolidated results of the new user-defined kernel functionality of SVM being tested on emotion recognition in speech dataset using CIGMMIMK based SVM.[5]

Table 3.1: CIGMMIMK based SVM consolidated results

Number of components	Classification accuracy (in %)					Average accuracy (in %)
	Fold1	Fold2	Fold3	Fold4	Fold5	
32	85.0000	86.0000	71.0000	86.0000	77.6596	81.1319
64	84.0000	86.0000	77.0000	87.0000	77.6596	82.3319
128	87.0000	84.0000	75.0000	92.0000	77.6596	83.1319
256	84.0000	89.0000	79.0000	92.0000	84.0426	85.6085
512	86.0000	87.0000	84.0000	93.0000	82.9787	86.5957
1024	87.0000	85.0000	83.0000	94.0000	81.9149	86.1830

The highlighted row gives the best performance of the SVM.

3.5 Conclusion & Future Scope

SVMTorch has now been equipped with the true potential of being used with user defined Kernel Gram Matrices which can easily be pre-computed using software such as MATLAB. The real advantage however is in the significant reduction of time taken for training the SVM even with a very large set of training examples, which was previously inefficient using SVM libraries such as LIBSVM in MATLAB. This improvement has been possible due to the library being written in C++ which is inherently faster and also due to an optimality check which is similar to SMO algorithm and the shrinking heuristic. However the scope for improvement in the software remains in the following areas:

- The code can further be extended to support sparse input file formats for the user-kernel mode.

- The original code can be further cleaned up replacing irrelevant variable names with relevant ones.
- A large number of comments can be added in several places explaining the functioning of the code, which is currently absent; also the remaining French comments be converted to English.
- Improvement to the method of reading files possible for increased speed.

Bibliography

[1] R. Collobert and S. Bengio, "SVMTorch: Support vector machines for large-scale regression problems," *Journal of Machine Learning Research*, vol. 1, pp. 143–160, 2001. Software available at `http://bengio.abracadoudou.com/SVMTorch.html`.

[2] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, pp. 121–167, June 1998.

[3] T. Joachims, "Making large-scale SVM learning practical," in *Advances in Kernel Methods - Support Vector Learning* (B. Schölkopf, C. Burges, and A. Smola, eds.), ch. 11, pp. 169–184, Cambridge, MA: MIT Press, 1999.

[4] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[5] A. D. Dileep and C. C. Sekhar, "GMM based intermediate matching kernel for classification of varying length patterns of long duration speech using support vector machines," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, pp. 1421–32, November 2013.