Journal of Machine Learning Research 15 (2014) 3483-3487 Submitted 2/14; Revised 5/14; Published 10/14

The Gesture Recognition Toolkit

Nicholas Gillian NGILLIAN@MEDIA.MIT.EDU
Joseph A. Paradiso JOEP@MEDIA.MIT.EDU
Responsive Environments Group, Media Lab
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

Editor: Isabelle Guyon, Vassilis Athitsos, Sergio Escalera

Abstract

The Gesture Recognition Toolkit is a cross-platform open-source C++ library designed to make real-time machine learning and gesture recognition more accessible for non-specialists. Emphasis is placed on ease of use, with a consistent, minimalist design that promotes accessibility while supporting flexibility and customization for advanced users. The toolkit features a broad range of classification and regression algorithms and has extensive support for building real-time systems. This includes algorithms for signal processing, feature extraction and automatic gesture spotting.

Keywords: gesture recognition, machine learning, C++, open source, classification, regression, clustering, gesture spotting, feature extraction, signal processing

1. Introduction

Gesture recognition is a powerful tool for human-computer interaction. It is increasingly redefining how we interact with our smartphones, wearable devices, televisions and gaming consoles. In addition to the increasing prevalence of gesture-based interactions in consumer devices, a diverse range of individuals are gaining access to affordable sensor technology and rapid-prototyping tools that facilitate non-specialists to build custom gesture-based applications. Commercial sensors such as the Microsoft Kinect or easy-to-use hardware platforms like Arduino (Mellis et al., 2007), combined with prototyping environments, such as Processing or Openframeworks,[1] are empowering professional developers, students, researchers, hobbyists, creative coders, interaction designers, musicians and artists to create novel-interactive systems that are playful, poignant and expressive.

Nevertheless, while a diverse range of individuals now have access to powerful sensors and rapid-prototyping tools, performing *real-time* gesture recognition can pose a challenge, even to accomplished developers and engineers (Patel et al., 2010). This is despite the large number of sophisticated machine-learning applications currently available, such as MATLAB, R and WEKA (Hall et al., 2009). Many of these applications are primarily designed for offline analysis of prerecorded data sets by domain experts, and require substantial effort to recognize real-time signals. There are accessible machine-learning libraries in Java (Abeel et al., 2009) and Python (Pedregosa et al., 2011) that can be used to prototype real-time

1. More information on Processing and Openframeworks can be found on their respective websites: `http://processing.org` and `http://www.openframeworks.cc`.

systems. However, many users need to build their systems in C++ due to the computational overhead of the sensor data and interactive visualizations and therefore benefit from C++ tools for real-time machine learning. While there are a number of powerful C++ libraries that can be adapted for gesture recognition (King, 2009; Sonnenburg et al., 2010; Gashler, 2011), these tools still require the user to develop the supporting infrastructure needed to build real-time systems and can have steep learning curves for non-specialists. This leaves C++ users with a sizable gulf of execution, specifically the gap between their goals and the actions needed to attain those goals with the system (Hutchins et al., 1985). This gap can significantly impede the process of building novel gesture-based interfaces for technologists, researchers, artists and beyond.

2. Gesture Recognition Toolkit

To address this issue, we have created the Gesture Recognition Toolkit (GRT), a cross platform open source C++ machine-learning library for real-time gesture recognition. The toolkit was developed with the following core design principles:

Accessibility: The GRT is a general-purpose tool for facilitating non-specialists to create their own machine-learning based systems. Emphasis is placed on ease of use, with a clear and consistent coding convention applied throughout the toolkit. The GRT provides a minimal code footprint for the user, reducing the need for arduous and error-prone boilerplate code to perform common functionality, such as passing data between algorithms or to preprocess data sets. This consistent, minimalist design significantly lowers the entry barrier for a new user because the same subset of core functions apply throughout the toolkit.

Flexibility: To support flexibility while maintaining consistency, the GRT uses an object-oriented modular architecture. This architecture is built around a set of core **modules** and a central **gesture-recognition pipeline**. The input to both the modules and pipeline consists of an N-dimensional double-precision vector, making the toolkit flexible to the type of input signal. The algorithms in each module can be used as stand-alone classes; alternatively a gesture-recognition pipeline can be used to chain modules together to create a more sophisticated gesture-recognition system. The GRT includes modules for preprocessing, feature extraction, clustering, classification, regression and post processing.

Choice: To date, there is no single machine-learning algorithm that can be used to recognize all gestures. It is therefore crucial for a user to be able to choose from, and quickly experiment with, a number of algorithms to see which might work best for their particular task. The GRT features a broad range of machine-learning algorithms such as AdaBoost, Decision Trees, Dynamic Time Warping, Hidden Markov Models, K-Nearest Neighbor, Linear and Logistic Regression, Naïve Bayes, Multilayer Perceptrons, Random Forests, Support Vector Machines and more.[2] In addition to supporting a broad range of algorithms, the toolkit's architecture facilities a user to seamlessly switch between different algorithms with minimal modifications to the user's code.

Supporting Infrastructure: Building sophisticated machine-learning based systems requires more than just a state-of-the-art classifier. In many real-world scenarios, the input

2. For Support Vector Machines, we provide an easy-to-use wrapper for LibSVM (Chang and Lin, 2011). All other algorithms are custom implementations unless otherwise stated in the source documentation.

to a classification algorithm must first be preprocessed and have salient features extracted. Preprocessing and feature extraction are important because they can significantly improve the predictive performance of a classifier, and also provide faster and more cost-effective predictors (Guyon and Elisseeff, 2003). The GRT therefore supports a wide range of pre/post processing, feature extraction and feature selection algorithms, including popular preprocessing filters (e.g., Moving Average Filter), embedded feature extraction algorithms (e.g., AdaBoost), dimensionality reduction techniques (e.g., Principal Component Analysis), and unsupervised quantizers (e.g., K-Means Quantizer, Self-Organizing Map Quantizer). Accurate labeling of data sets is also critical for building robust machine-learning based systems. The toolkit therefore contains extensive support for recording, labeling and managing supervised and unsupervised data sets for classification, regression and time series analysis.[3]

Customizability: In addition to using the wide range of existing GRT algorithms, more advanced users commonly want to test or deploy their own algorithms when building novel recognition systems, such as using a custom feature-extraction algorithm. The GRT is therefore designed to facilitate users to easily incorporate their own algorithms within the toolkit's framework by inheriting from one of the GRT base classes. The toolkit leverages advanced object-orientated concepts, such as polymorphism and abstract base-class pointers, facilitating custom algorithms to be used alongside any of the existing GRT algorithms.

Real-time Support: The GRT supports common techniques for performing offline analysis on pre-recorded data sets, such as partitioning data into validation and test data sets, running cross validation and computing accuracy metrics. In addition to these offline techniques, the toolkit is designed to enable a user to seamlessly move from the offline analysis phase to the real-time recognition phase. One significant challenge involved in moving from offline analysis to real-time gesture recognition is automatically segmenting valid gestures from a continuous stream of data (Junker et al., 2008). This is a nontrivial task because the input data might consist of generic movements that are not valid gestures in the model. To support real-time gesture recognition, the GRT features algorithms that automatically perform gesture spotting. These algorithms, such as the Adaptive Naïve Bayes Classifier (Gillian et al., 2011a) and N-Dimensional Dynamic Time Warping (Gillian et al., 2011b), learn rejection thresholds from the training data, which are then used to automatically recognize valid gestures from a continuous stream of real-time data.

3. Code Example

The code example below demonstrates the core design principles of the GRT. This example shows how to setup a custom gesture-recognition system consisting of a moving-average filter preprocessing module, a fast Fourier transform and custom feature extraction modules, an AdaBoost classifier and a timeout-filter post processing module. The example also illustrates how to load a training data set from a CSV file, train a classification model, and use this model to predict the class label of a new data sample.

```
   //Setup a custom recognition pipeline
1: GestureRecognitionPipeline pipeline;
2: pipeline.addPreProcessingModule( MovingAverageFilter( 5 ) );
```

3. A detailed description of the GRT data structures can be found at `http://www.nickgillian.com/wiki/pmwiki.php/GRT/Reference`.

```
3: pipeline.addFeatureExtractionModule( FFT( 512 ) );
4: pipeline.addFeatureExtractionModule( MyCustomFeatureAlgorithm() );
5: pipeline.setClassifier( Adaboost( DecisionStump() ) );
6: pipeline.addPostProcessingModule( ClassLabelTimeoutFilter( 1000 ) );

   //Load a labeled data set from a CSV file and train a classification model
7: ClassificationData trainingData;
8: trainingData.load( "TrainingData.csv" );
9: bool success = pipeline.train( trainingData );

    //The following lines would be called each time the user gets a new sample
10: vector< double > sample = getDataFromSenor(); //Custom user function
11: pipeline.predict( sample );
12: unsigned int predictedClassLabel = pipeline.getPredictedClassLabel();
13: double maxLikelihood = pipeline.getMaximumLikelihood();
```

Lines 1 through 6 show how a `GestureRecognitionPipeline` can be used to link several modules together to build a more complex recognition system. Note that the customization of the recognition system is achieved with a minimal code footprint, as the pipeline will automatically connect the output of one module to the next module's input; propagating signals through the entire pipeline at both the training, testing and real-time prediction phases. These six lines also illustrate the flexibility of the toolkit's modular design, and demonstrate how a user can easily experiment with different algorithms from existing modules, or insert a custom algorithm into the pipeline as illustrated on line 4. Line 10 demonstrates how real-time sensor data from a variety of devices can be incorporated; input can consist of something as simple as the three-dimensional data from an accelerometer connected to an Arduino, to more complex inputs, such as the high-dimensional skeleton data from a Kinect.

This example also demonstrates one of the key designs of the GRT that make it more accessible: clean and consistent coding through abstraction. For instance, lines 9 and 11 show respectively how a user can train a model and then predict the class label of a new sample using that model. These key functions are the same, regardless of which algorithms are used. This abstraction significantly reduces the learning curve for new users, because the same key functions are consistent across all the GRT algorithms.

4. Conclusion

The gesture recognition toolkit is open source under the MIT license and has been publicly available since 2012, receiving over ten-thousand hits on the toolkit's website.[4] It has been downloaded several thousand times and has built up a community of over 300 users on the toolkit's forum. To support a diverse range of users, we have established a number of online resources, including detailed examples for each module and a wide range of tutorials and references.[5] Future work includes an interactive graphical user interface, in which a user can record and label training data; configure; train and test a gesture-recognition model; perform real-time prediction and then export their model and pipeline configuration so it can be loaded directly into the user's program, using the C++ API.

4. The toolkit's website can be found at: `http://www.nickgillian.com/software/grt`.
5. Online tutorials, references and examples can be found at: `http://www.nickgillian.com/wiki`.

References

T. Abeel, Y. Van de Peer, and Y. Saeys. Java-ML: A machine learning library. *Journal of Machine Learning Research*, 10:931–934, 2009.

C.C. Chang and C.J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

M. Gashler. Waffles: A machine learning toolkit. *Journal of Machine Learning Research*, 12:2383–2387, 2011.

N. Gillian, R. B. Knapp, and S. O'Modhrain. An adaptive classification algorithm for semiotic musical gestures. In *Proceedings of the 8th Sound and Music Computing Conference*, 2011a.

N. Gillian, R. B. Knapp, and S. O'Modhrain. Recognition of multivariate temporal musical gestures using n-dimensional dynamic time warping. In *Proceedings of the 2011 International Conference on New Interfaces for Musical Expression*, 2011b.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

E. L. Hutchins, J. D. Hollan, and D. A. Norman. Direct manipulation interfaces. *Human–Computer Interaction*, 1(4):311–338, 1985.

H. Junker, O. Amft, P. Lukowicz, and G. Tröster. Gesture spotting with body-worn inertial sensors to detect user activities. *Pattern Recognition*, 41(6):2010–2024, 2008.

D.E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

D. Mellis, M. Banzi, D. Cuartielles, and T. Igoe. Arduino: An open electronics prototyping platform. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2007.

K. Patel, N. Bancroft, S. M. Drucker, J. Fogarty, A. J. Ko, and J. Landay. Gestalt: integrated support for implementation and analysis in machine learning. In *Proceedings of the 23rd Annual ACM symposium on User Interface Software and Technology*, 2010.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

S. Sonnenburg, G. Rätsch, S. Henschel, C. Widmer, J. Behr, A. Zien, F. Bona, A. Binder, C. Gehl, and V. Franc. The SHOGUN machine learning toolbox. *Journal of Machine Learning Research*, 11:1799–1802, 2010.