Version

2.0

HCRF

Hidden-state Conditional Random Field Library

User Guide

HCRF

User Guide

Main point of contact :
Louis-Philippe Morency
morency@ict.usc.edu

Contributors:
C. Mario Christoudias
Ariadna Quattoni
Hugues Salamin
Giota Stratou
Sybor Wang

Version 2.0a

January 12th, 2010

Table of Contents

Chapter

1

Introduction

HCRF is a C++ library for training and inference of Conditional Random Field (CRF), Hidden-state CRF (HCRF) and Latent-dynamic CRF (LDRCF) models.

This guide presents the functionalities and characteristics of the HCRF library. The main purpose of this library is to offer a simple and intuitive interface to efficient implementations of different Conditional Random Field models. The core part of the library was implemented in C++. The hCRF library also offers Matlab and Python interfaces. Matlab, Python and C++ interfaces are all compatible with Microsoft Windows(XP/7) and Linux.

This library implements three main models: Conditional Random Field (CRF), Hidden-state Conditional Random Fields (HCRF) and Latent-Dynamic Conditional Random Fields (LDCRF). The CRF implementation is based on Lafferty *et al.* original paper [5] as well as Sha and Pereira paper [6]. We implemented two variants of the HCRF model: Quattoni *et al.* [3] (referred as HCRF model) and Gunawardana *et al.* [7] (referred as Gaussian-HCRF or GHCRF model). Finally, we implemented the Latent-Dynamic CRF of Morency *et al.* [1] which extends the original CRF model to include hidden variables. The CRF and LDCRF models can be applied to unsegmented sequences while the HCRF (and GHCRF) should be apply to pre-segmented sequences (only one label per sequence).

The following chapter explains the installation procedure for the hCRF library. Chapter 3 explains the different parameters of the demo application TestHCRF. Chapter 4 presents the Matlab interface. Chapter 5 presents the C++ interface and the details of the implementation.

Chapter
2

Installation

The hCRF library can be easily installed on a Microsoft Windows system using the IntallShield installation package.

System Requirements

- Microsoft Visual Studio 2008 or better, or equivalent C++ compiler.
- MATLAB 7.1 or superior to use the MATLAB interface.

Package structure

The core installation file hcrf-x.xx.exe will copy on your machine the following components:

- hCRF\apps: C++ files and projects for the Matlab wrapper (matHCRF), the Python wrapper (using SWIG) and the demo application (TestHCRF)
- hCRF\bin: hCRF demo program (TestHCRF.exe) and Matlab Mex files necessary to access the Matlab interface;
- hCRF\doc: Documentation of the hCRf library;
- hCRF\Samples\C++: Sample data files to train CRF models using testHCRF.exe application ;
- hCRF\Samples\Matlab: Samples programs for the Matlab interface;
- hCRF\libs: Contains source files for the core hCRF library and the 3rd party libraries.

- hCRF\libs\3rdParty: Contains the source code and projects for the 3rd party libraries. cgDescent, liblbfgs and uncOptim are optimization algorithms . The Matlab directory contains some file for the Matlab wrapper.
- hCRF\libs\shared\hCRF: core hCRF library.

Compilation

Microsoft Visual Studio (Windows)

The main solution for Microsoft Visual Studio is hCRF\apps\hCRF\hcrf.sln. This solution contains 6 projects:

- hCRF: core part of the hCRF library. Contains the code implementing CRF, LDCRF and HCRF.
- matHCRF: Matlab wrapper.
- testHCRF: Demo applications.
- cgDescent: An optimization library implementing Conjugate Gradient Descent.
- Lib (short version for liblfgs): Optimization library implementing Limited-memory BFGS with L1 or L2 regularization.
- uncOptim: Optimization library implementing Limited-memory BFGS with only L2 regularization.

Makefiles (Linux)

To compile in Linux you will first need to install cmake and g++ compiler on your machine.

Each project contains a CMakeLists.txt. To compile, you should go to the project folder and run "cmake ./ " command, which will create the necessary makefiles. In the same folder, run "make" and that should compile the project and necessary dependencies.

Matlab Installation

To run the Matlab sample programs, you will need to add the hCRF\bin directory to your Matlab path list. To add the path manually you can use the 'addpath' command as

```
> addpath('c:\matlab\work\hcrf');
```

or by using the MATLAB GUI path interface.

Licenses

HCRF library is distributed under GNU General Public License. Licenses for the 3rd party libraries used, are as follows:

- CgDescent: Distributed under GNU General Public License. More information at: http://www.math.ufl.edu/~hager/papers/CG

- Liblbfgs: Distributed under the term of the MIT license. Please refer to COPYING file in the distribution. Refer to the libLBFGS web site for more information. http://www.chokkan.org/software/liblbfgs/

- UncOptim: Distributed under GNU General Public License. Source code from: http://faculties.sbu.ac.ir/~katanforoush/uncoptim/uncoptim.tar.gz

Chapter 3

Software Interface

The demo program gives you command prompt interface to train and test different models.

Overview

The HCRF library comes with a demo program called TestHCRF.exe. This program can be used to train and test CRF, HCRF and LDCRF models from the command prompt.

Parameters

The demo program TestHCRF.exe is called using the following syntax:

```
TestHCRF.exe [-t] [-T] [-d filename] [-l filename] [-D filename] [-L
filename] [-m filename] [-f filename] [-r filename] [-o cg|bfgs|asa]
[-a crf|ldcrf|hcrf|ghcrf]
```

Parameters	Description	Default
-t	Train the model using the training dataset	
-T	Test the model	
-tc	Resume training using the intermediate saved data	
-d	Name of the file containing the training data	dataTrain.csv
-ds	Name of the file containing sparse training data	
-l	Name of the file containing the training labels	labelsTrain.csv
-TT	Test the model on both the training and testing data	
-D	Name of the file containing the testing data	dataTest.csv
-L	Name of the file containing the testing labels	labelsTest.csv
-m	Name of the file where the model is written	model.txt
-f	Name of the file where the features are written	features.txt
-r	Name of the file where computed labels are written	results.txt
-c	Name of the file where statistics are written	stats.txt

-o	Select optimizer: 'cg', 'bfgs' or 'lbfgs'	bfgs
-a	Model: 'crf','hcrf','ghcrf' or 'ldcrf'	ldcrf
-i	Maximum number of iterations	300
-s2	Sigma2: L2 regularization factor	0.0 (no L2 reg.)
-s1	Sigma1: L1 regularization factor	0.0 (no L1 reg.)
-I	Initialization strategy: 'random', 'gaussian' or 'zero'	random
-R	Range for random initiliazation	-1 1
-w	Window size. Number of neighboring observations used in the input vector. If w=1, then the next and previous observations will be used.	0
-h	Number of hidden state	3
-P	Number of parallel thread	1
-p	Debug print level	1

Data Format

The data files and label files are encoded using the CSV format (comma separated values). Each file contains multiple matrices or vectors encoding the feature values (data files) or label values (label files).

Data Files

A data file contains multiple matrices, one for each sequence. There are two types of data files, dense and sparse matrix data

Dense matrix representation: For each matrix, the first line always contains 2 numbers: the number of rows and the number of columns. The number of rows for each matrix represents the number of features. All the matrices should have the same number of features. The number of column for a specific matrix represents the number of time samples in the sequence.

Sparse matrix representation: we can use data entries with sparse representation for the matrices as follows:

```
#rows_matrix1          #columns_matrix1          0

row_nonZeroElement1    col_nonZeroElement1       value1

…

row_nonZeroElementN    col_nonZeroElementN       valueN

#rows_matrix2          #columns_matrix2          0

row_nonZeroElement1    col_nonZeroElement1       value1
```

```
…
```

Label Files

Since HCRF models have only one label associated to each sequence while CRF and LDCRF have one label associated to each time sample in the sequence, the HCRF library supports two file format for labels.

For HCRF and GHCRF models, the label file contains one integer per line, representing the label for the specific sequence

For CRF and LDCRF models, the label file is encoded as a data file with matrix headers specifying the number of row and columns but in this case the matrices always have one row. This row should have the same length as the corresponding sequence in the data file, with one label for each time sample.

Chapter

4

MATLAB Interface

The HCRF library comes with a MATLAB interface (matHCRF). The MATLAB library allows to access to full functionality of the C++ library.

Overview

The HCRF library functionalities are all accessed through the *matHCRF* function. The *matHCRF* function is defined as:

matHCRF(Action,Parameters)

The following section describes the different action available as well as the associated parameters for each action.

Action Descriptions

Action 'createToolbox'

This action creates an instance of the hCRF toolbox, and defines the main characteristics of this particular toolbox.

matHCRF('createToolbox',modelType,optimizerType,nbHiddenStates,windowSize);

The parameters are:

Parameter	Type	Explanation	Possible inputs
modelType	String	Defines the desired CRF type to use.	'crf','hcrf',’ghcrf’, 'ldcrf'
optimizerType	String	Defines the optimizer to be used. Use 'bfgs' for limited Newton and	'cg', 'bfgs', ‘lbfgs’

		'cg' for conjugate gradient.	
nbHiddenStates	Int	Defines the number of hidden (latent) states in either HCRF or LDCRF type toolboxes.	
windowSize	Int	Defines the number of time sample used to create the feature vector.	0 : only the current time sample is used 1+ : neighbor samples are concatenated with the current sample to create the final feature vector

Action 'setData'

This action sets the data of a certain dataset into the current instance of hCRF.

matHCRF('setData',featureSequences,labelSequences,labels);

The main parameters are:

Parameter	Type	Explanation	Possible inputs
featureSequences	Array of matrices	Each matrix contains the input features for one sequence. All matrices should have the same number of rows, representing the number of features per sample.	Double
labelSequences	Array of integer vectors	Each vector contains the labels for a specific sequence. Labels are zero-based. Applies only to CRF and LDCRF models.	Integer
labels	Integer vector	Each entry represents the label of an entire sequence. Labels are zero-based. Applies only to HCRF and GHCRF models.	Integer

Action 'loadData'

This action reads from the dataset from files.

matHCRF('loadData',fileFeatureSequences,fileLabelSequences,fileLabels);

The main parameters are:

Parameter	Type	Explanation	Possible inputs
fileFeatureSequences	String	Name of the CSV file containing the precomputed features. See Chapter 3 for details on the format.	Double
fileLabelSequences	String	Name of the file containing the labels for each node of each sequence. Applies only to CRF and LDCRF models. See Chapter 3 for details about the file format.	Integer
fileLabels	String	Name of the file containing the label of each sequence. Applies only to HCRF and GHCRF model. See Chapter 3 for details about the file format.	Integer

Action 'set'

This action sets internal parameters of the toolbox currently instanced.

matHCRF('set',parameter,parameter_value);

The possible parameters are:

Parameter	Type	Explanation	Parameter_value
'maxIterations'	String,Int	Defines the maximum number of iterations at training time.	Integer
'debugLevel'	String,Int	Defines Debug level and verbosity	0,1,2
'regularizationL2'	String, Double	Defines the value of the L2 regularization constant for training.	Double
'regularizationL1'	String, Double	Defines the value of the L1 regularization constant for training.	Double
weightsInitType	String	Mode for initializing weight: 'ZERO', 'CONSTANT', 'RANDOM', 'GAUSSIAN'	String
'minRangeWeights'	String, Double	During initialization of the weight vector, this parameter defines the lower bound.	Double

'maxRangeWeights'	String, Double	During initialization of the weight vector, this parameter defines the higher bound.	Double
initWeights	String, Double vector	Used to manually initialize the model using a weight vector	Double
'nbThreads'	String, Int	Defines number of threads to use when multithreading is enabled (default value: all cores)	Integer
randomSeed	String, Double	Random seed for the weight initialization (when using the random initialization mode)	

Action 'getModel'

This action obtains the current model parameters from the currently instanced toolbox.

[model, featureDefinition]=matHCRF('getModel');

The function returns two structures, one containing the main model parameters and other containing the generated features. The table below explains the parameters that the action returns.

Parameter	Type	Explanation	parameter_value
model	Structure	Contains the information about the trained model including the weight vector.	Structure
featureDefinition	Structure	Contains the features information for the current model.	Structure

Action 'setModel'

This action sets the parameters for the model in the currently instanced toolbox.

matHCRF('setModel',model,featureDefinition);

This is particularly useful for already trained models.

Parameter	Type	Explanation	parameter_value
Model	Structure	Contains the information about the trained model including the weight vector.	Structure

featureDefinition	Structure	Contains the features information for the current model.	Structure

Action 'saveModel'

Save the current model parameters from the currently instanced toolbox.

matHCRF('saveModel',fileModel,fileFeatureDefinition);

The parameters are:

Parameter	**Type**	**Explanation**	**parameter_value**
fileModel	String	Name of the file which will contain the information about the trained model including the weight vector.	
fileFeatureDefinition	String	Name of the file which will contain the features information for the current model.	

Action 'loadModel'

This action reads from file the parameters for the model in the currently instanced toolbox.

matHCRF('loadModel',fileModel,fileFeatureDefinition);

The parameters are:

Parameter	**Type**	**Explanation**	**parameter_value**
fileModel	String	Name of the file which contains the information about the trained model including the weight vector.	
fileFeatureDefinition	String	Name of the file which contains the features information for the current model.	

Action 'train'

This action starts the training of the current instanced toolbox, using the parameters previously set.

matHCRF('train');

Action 'test'

This action initiates and run the test of the current instanced toolbox.

matHCRF('test');

The results can be later retrieved.

Action 'getResults'

Obtains the results from the last test.

matHCRF('getResults');

Action 'unloadData'

Clear the current dataset from memory.

matHCRF('unloadData');

Action 'clearToolbox'

Clear the current Toolbox object from memory.

matHCRF('clearToolbox');

Sample program

The HCRF library comes with one Matlab sample script called runSampleExperiment.m. This sample script load data from file, train three models (CRF, LDCRF and HCRF) and plot the resulting ROC curve. This is a simple example showing how to use the Matlab interface. This sample program does not do any validation of the training parameters. The dataset used in this sample example come from an eye gesture user study[4]. The input features are 2D eye gaze estimates. The positive labels (1) represent eye gaze aversion gestures.

Chapter 5

C++ Interface

HCRF offers a C++ interface for training and testing all three models: CRF, HCRF and LDCRF.

C++ Classes Overview

Main classes

The following classes are the main classes needed to interact with hCRF:

Name	Inherit from	Description
Toolbox		Main interface for the hCRF library.
ToolboxCRF	Toolbox	Implementation of the Toolbox abstract class for CRF.
ToolboxLDCRF	Toolbox	Implementation of the Toolbox abstract class for LDCRF.
ToolboxHCRF	Toolbox	Implementation of the Toolbox abstract class for HCRF.
ToolboxGHCRF	ToolboxHCRF	Simple modification of ToolboxHCRF to include two new types of features. Implements the Gaussian-HCRF model.
DataSequence		Input features and labels for a specific sequence.
DataSet	List<DataSequence>	List of sequences and labels.

Detailed Interface

The following subsections list and describe the member functions of the most important classes.

Toolbox

Toolbox classes are utility classes designed to give a user friendly interface to train and test different models. There are three implementations of the abstract class Toolbox, one for each model: ToolboxCRF, ToolboxHCRF and ToolboxLDCRF. All three classes offer the same interface, except for the constructor function.

Constructors

- **ToolboxCRF():** This constructor expects two arguments: the type of optimizer and the window size. There are two optimizers currently implanted: conjugate gradient descent and BFGS. The parameter window size can be used to concatenate neighborhood feature vectors. The defaults value is 0, no concatenation.

- **ToolboxHCRF():** This constructor expects three arguments: the number of hidden states, the type of optimizer and the window size. The number of hidden state is the total number of possible values the hidden state variable can take. These hidden states are shared along all class labels.

- **ToolboxGHCRF():** This constructor expects three arguments: the number of hidden states, the type of optimizer and the window size. The number of hidden state is the total number of possible values the hidden state variable can take. These hidden states are shared along all class labels.

- **ToolboxLDCRF():** This constructor expects three arguments: the number of hidden states per labels, the type of optimizer and the window size. The total number of hidden states is equal to the number of labels time the number of hidden states per labels.

Training and testing

- **train():** Train the model using the Dataset passed as argument. The toolbox should have been initialized properly (i.e., set number of hidden states and regularization factor).

- **test():** Apply the current model to each sequences in the Dataset. Optionally, save the results on files.

Load and save

- **load():** Load a model and feature descriptions in memory.

- **save():** Save on file the model and feature descriptions. Suppose that the model was trained before.

Utility functions

- **getMaxNbIteration():** Return the maximum number of iteration for the optimizer. Default: 200.
- **getRegularization():** Return the current value for the regularization term. Default: 0, no regularization.
- **getDebugLevel():** Return the level of debug messages. 2: Message every iteration. 1: Message only at the end of training. 0: No debug messages. Default: 1.
- **setDebugLevel():** Set the level of debug messages.
- **setRangeWeight():** Set the maximum and minimum range for initializations of the weight vector. Each element of the weight vector will be randomly picked between these two values. Default: [-1 1].
- **setMinRangeWeight():** Set minimum value for the weight vector. Default: -1.
- **setMaxRangeWeight():** Set maximum value for the weight vector. Default: 1.
- **getMinRangeWeight():** Returns minimum value for the weight vector.
- **getMaxRangeWeight():**Returns maximum value for the weight vector.

Internal structures

- **getModel():** Returns a pointer on the internal Model object.
- **getFeatureGenerator():** Returns a pointer on the internal FeatureGenerator object.
- **getOptimizer():** Returns a pointer on the internal Optimizer object.

DataSequence

This class is a container structure for sequences with pre-computed features. These features will be directly passed to the training algorithm (through the RawFeature or WindowRawFeature class). This class can be expanded to include other types of features (e.g., words). This class also contains three other type of information: the labels, the adjency matrix and the results from the test() function. The labels can be encoded as a single value (for HCRF models) or as a vector of labels (for CRF and LDCRF). The adjency matrix represents the connectivity between each node of the sequence. For a chain, this matrix should be diagonal. If no adjency matrix is set, then the Model class will create one automatically based on the current settings. Finally, the class contains the estimated labels and probabilities that are computed by the test() function.

DataSet

This class is an augmented list of data sequences where we added utilities functions to read data from file and parse the sequences to find the number of labels (sequence labels for HCRF and state labels for CRF and LDCRF) and number of pre-computed features.

FeatureGenerator

This class creates a vector of all non-zero features for a given data sequence and a given data position in the sequence. This task is performed by calling the main function of this class: getFeatures. The main body of this function goes through all the active FeatureType and records the non-zero features from each FeatureType.

Main function

- **`getFeatures()`:** `Based on a DataSequence, a model and a position in the sequence, this function returns the list of non-zeros features. Optionally, the index of the previous node can be passed so that edge features are also returned. The parameter seqLabel is used by HCRF models to specify the current sequence label.`

FeatureType utility functions

- **`AddFeature()`:** `Add a FeatureType object to the list of active FeatureType. This list will later be used by the function getFeatures() to compute the non-zeros features.`

- **`initFeatures()`:** `Initialize all the FeatureType objects that were added on the list. This function also makes sure that each feature has a unique ID.`

- **`clearFeatureList()`:** `Remove all the FeatureType objects from the active list.`

All features utility functions

- **`getNumberOfFeatures()`:** `Returns the number of features (zero and non-zero) for a specific type (edge and/or state features).`

- **`getAllFeatures()`:** `Returns a vector with all possible features for this model. The value of each features are irrelevant. This function can be used at debugging to know the meaning of each feature.`

FeatureType

This abstract class presents a uniform interface for creating features. Currently, four implementations of this class are available: RawFeatures, WindowRawFeatures, EdgeFeatures, LabelEdgeFeatures.

The RawFeatures represents "state" feature functions that depend only on the observation vector and the current position in the sequence. These features can be used with all three models (CRF, HCRF and LDCRF). A feature is created for each pair of observation feature and label (for CRF) or hidden state (HCRF, GHCRF and LDCRF). No processing of the original feature values is done. This class returns the value stored in the DataSequence. The RawFeature class should be used as a template to implement your own class for feature generation.

The WindowRawFeature is an extension of RawFeature where observation feature vector is extended to include neighbor observations. If the window size is 1, then this class will concatenate the observations from the node before and after the current node.

The EdgeFeatures represents "edge" feature function between two adjacent labels (for CRF) or hidden states (for HCRF, GHCRF and LDCRF). These edge features depends on the adjacency matrix for the specific DataSequence. Given the indices for the current and previous nodes, the EdgeFeatures will return non-zero feature only if there is a link between the two nodes.

The LabelEdgeFeatures represents the feature functions between a hidden state and the sequence label. This FeatureType applies only to HCRF models. While this feature has only binary values like the EdgeFeature, it is considered as a "state" feature during training since the sequence label is a known observation during training.

- **`Init()`**: `Initialize the number of feature for this specific FeatureType based on the type of model (CRF, HCRF, LDCRF) and based on the number of Raw features in the Dataset. This function could also be used to initialize some internal structures such as a dictionary of all the words for DataSequence consisting of sentences.`

FeatureVector

This class represents a vector of Features. This class was created for efficiency so that memory allocation does not happen too often. This class replaces the original implementation using std::vector.

Feature

This class structure represents a non-zero feature value in the FeatureVector. The two most important fields are the globalId and the value.

- **globalId**: The global index represents a unique index specific to that feature. During the call of FeatureGenerator::InitFeatures(), the library ensures that no feature has the same index number. This global Index represents the position of this feature in the weight vector.
- **Value**: The value field member represents the value of this feature.
- **id**: Local ID of this feature for the specific FeatureType. The global Id of a feature is equal to the id + the offset from the FeatureType.
- **nodeIndex**: index of the node representing the current position.
- **NodeState**: State of the node at the current position. For a CRF, this state represents the label. For HCRF and LDCRF, this state represents the hidden state.
- **nodeIndex**: index of the node representing the previous position. Applies only to "edge" features. -1 otherwise.
- **prevNodeState**: State of the node at the previous position.
- **sequenceLabel**: Label for this sequence. Applies to HCRF models only.

Custom Features and Toolbox

In this section we describe how to create your own features. This tutorial refers to two classes: MyFeatures and MyToolbox.

The class MyFeatures (\include\MyFeatures.h and \src\MyFeatures.cpp) is an example of custom feature generation. This class is based on RawFeatures but instead of returning the original precomputed features for each state, this class returns the squared value of the precomputed features. This is a small modification but the goal here is to show how to create your own features.

The class MyToolbox (\include\MyToolbox.h and \src\MyToolbox.cpp) is an example of a CRF toolbox that uses the new feature generator MyFeatures. This class inherits from ToolboxCRF. The same technique would apply for HCRF and LDCRF.

The most important function in MyFeatures is the getFeatures() function. This function is called automatically by the FeatureGenerator every time the inference engine needs to know the non-zero features for a specific node/sample (i.e., sample *j*) in the DataSequence. The main purpose of FeatureType::getFeatures() is to return add to the variable ListFeatures all the non-zero features for the sample j in the DataSequence X.

There is two type of call for the function getFeatures: state features and transition (edges) features. A state feature is a feature that depends only on one state node. RawFeatures, WindowRawFeatures and MyFeatures are state feature generators. A transition/edge feature depends on two state nodes. EdgeFeatures and LabelEdgeFeatures (HCRF and LDCRF only) are transition feature generators. If the FeatureGenerator wants only the state features, the previousNodeIndex will be equal to -1. If the FeatureGenerator wants only the edge/transition features, then the previousNodeIndex will be larger or equal to 0.

Chapter

6

Troubleshooting

In this chapter, we describe solutions to common problems/mistakes happening when installing and using the HCRF library.

"Can't find the matHCRF function"

Problem

When trying to access the Matlab interface (matHCRF), Matlab can't find the matHCRF function.

Solution

Be sure that you added \hCRF\bin in Matlab PATH variable. To add it, use the command *addpath('*c:\path_to_bin*')*.

"Error when running executables compiled with OpenMP"

Problem

If projects compiled with OpenMP do not work, there is a chance that the target machine does not have some necessary openMP dll's installed.

Solution

Installing "Microsoft Visual C++ 2008 SP1 Redistributable Package" should fix this error.

Acknowledgement

The authors would to thank Juan Pablo Narino for his help debugging and documenting the first release.

References

[1] L.-P. Morency, A. Quattoni and Trevor Darrell, Latent-Dynamic Discriminative Models for Continuous Gesture Recognition, Proceedings IEEE Conference on Computer Vision and Pattern Recognition, June 2007

[2] Wang, S. Quattoni, A., Morency, L.-P., Demirdjian, D., and Trevor Darrell, Hidden Conditional Random Fields for Gesture Recognition, Proceedings IEEE Conference on Computer Vision and Pattern Recognition, June 2006

[3] A. Quattoni, M. Collins, T. Darrell, Conditional Random Fields for Object Recognition, In Neural Information Processing Systems, 2004

[4] L.-P. Morency, C. M. Christoudias and T. Darrell, Recognizing Gaze Aversion Gestures in Embodied Conversational Discourse, Proceedings International Conference on Multimodal Interfaces, October 2006

[5] Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: *Proc. 18th International Conf. on Machine Learning*, Morgan Kaufmann, San Francisco, CA (2001) 282–289

[6] Sha, F., Pereira, F.: Shallow parsing with conditional random fields. *Technical Report* MS-CIS-02-35, University of Pennsylvania (2003)

[7] Gunawardana, A., Mahajan, M., Acero, A, and Platt J. C. (2005). Hidden conditional random fields for phone classification . In International Conference on Speech Communication and Technology.