StrokeRecognitionLib

User Documentation

Preface

The StrokeRecognitionLib is a .NET library for the analysis of digital pen data. The library was developed for the examination of assessment tests with multiple choice questions and handwritten text answers.

This document describes how to implement and use the library in C#.

Its main features are:

1. Classification of patterns, that are typically used in multiple choice tests (crosses, circles, hatched regions)
2. Handwriting recognition of arbitrary text
3. Classification of handwritten text from a given dictionary

Introduction

To digitize drawings or handwriting, digital pens can capture their position on a pen display, pen tablet or sheet of digital paper. The pen captures the x and y coordinates (these must be non negative), the pressure on the tip and the time. A digitized *stroke* is represented by a set of *points*. Each point has an x and y coordinate and a timestamp.

Point 1:	X Y Time Pressure

A new stroke begins each time, when the pen's tip is pressed and ends, when the tip is released.

Stroke 1:	Point 1 Point 2 … Point n
Stroke 2:	Point n+1 …

This library was developed for the examination of assessment tests. The answers are drawn or written into separate rectangular *regions*. There are two types of regions: regions for text ("TextRegion") and regions for multiple choice (MC) answers ("PatternRegion").

The set of strokes of a TextRegion represents handwritten text. The strokes of a PatternRegion represent a pattern, like a cross or a circle, or a sequence of those patterns.

To collect all the strokes that relate to an answer, one typically collects all strokes that cross the corresponding region.

Region 1:	TextRegion Stroke 1 Stroke 2 Stroke 4 …
Region 2:	PatternRegion Stroke 2 Stroke 8

While each point uniquely belongs to one stroke, a stroke can cross multiple regions.

Library Installation

The StrokeRecognitionLib package includes the libsvm.dll, libSVMWrapper.dll (a .NET wrapper for the libsvm) and the supported vector machine model file (SVM_Model). Copy libsvm.dll and SVM_Model to your working directory and finally add a reference to StrokeRecognitionLib.dll in your project.

Library Usage

The analysis is performed region by region. The regions are either a "**PatternRegion**" for multiple choice (MC) answers or a "**TextRegion**". TextRegions can be used to recognize arbitrary handwritten text or to classify from a given dictionary (like "a","b","c", or "yes" and "no"). The general workflow to analyze a new region is as follows: one calls the right Region object's constructor for texts or patterns, adds the related strokes to this region and finally calls the "Recognize" method.

A region contains one or more **Stroke** objects. A **Stroke** is defined as the set of data that is captured between a pen down and pen up event. This data contains information such as an Id, a bounding box, a start time, a stop time and a list of **Point** objects. Each **Point** consists of x-coordinate, y-coordinate and a timestamp.

In the following sections the different kinds of regions are explained in detail.

PatternRegion

A **PatternRegion** contains one or more strokes. These strokes represent single patterns, like a circle or a cross, or could be a sequence of different patterns. The **PatternRegion** object sequences and recognizes all strokes in one region chronologically as well as upon given patterns.

Region 4:	PatternRegion Sequence 1 : Cross Sequence 2 : Crossed Out Sequence 3 : Circle
Region 8:	PatternRegion Sequence 1 : Slash

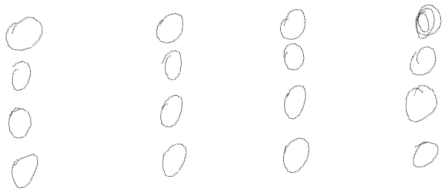
Patterns

The **PatternRegion** object is designed to recognize Multiple Choice (MC) answers. Typically a cross marks the selected answer (answers) out of the choices from a list. Sometimes the correct answer should be encircled. To deselect a given answer, this must be crossed out. As for some reasons the recording of the digital pen may fail, the list of patterns is extended by two more patterns. These are "Slash" and "Backslash" and might be interpreted as partial cross. Individual respondents may draw the same patterns differently.

- "Cross"

- "Crossed Out"

- "Circle"



- "Slash"
- "Backslash"

The StrokeRecognitionLib recognizes these patterns with an accuracy of over 92%. Note that the bounding rectangle of all patterns should have an aspect ratio of at most 1:2. Otherwise recognition falls of in quality.

Sequencing

All strokes in one **PatternRegion** must be sequenced with regard to the pattern a subset represents. The first chronological sequencing is based upon the strokes Ids. Strokes with consecutive Ids are grouped together. Suppose two regions build a group of possible answers to a MC-question and only one answer should be selected. The respondent first draws a cross in one region, then ticks the second region and finally crosses out the first.

Region 4:	PatternRegion Sequence 1 : Cross (Stroke 1, Stroke 2) Sequence 2 : Crossed Out (Stroke 5)
Region 5:	PatternRegion Sequence 1 : Cross (Stroke 3, Stroke 4)

A further sequencing might be necessary if two different patterns are drawn one after another in one region. For example the respondent marks a region with a cross and cross it out directly. This further sequencing is based upon the pattern a subset of consecutive strokes in one region forms out.

Region 8:	PatternRegion Sequence 1 : Cross (Stroke 1, Stroke 2) Sequence 2 : Crossed Out (Stroke 3)

PatternRegion Usage

The **PatternRegion** object sequences and recognizes all strokes drawn within its boundaries with regard to defined patterns. These patterns are “Cross”,”Cross Out”, “Circle”, “Slash” and “Backslash”. First of all you have to build a new **PatternRegion** object calling

PatternRegion region = new PatternRegion(Left, Top, Right, Bottom, shapeExtension).

The boundaries are given by a top left and a bottom right corner in pixel coordinates. If a shapeExtension value is defined, the regions boundaries are internally extended by this value. For each stroke, that crosses this (extended) region, build a new **Stroke** object

Stroke stroke = new Stroke(Id, startTime, stopTime, Left, Top, Right, Bottom).

Each stroke is defined by an Id, a startTime, a stopTime and a bounding rect. If no time is available for this stroke, set these values to 0. Note that in this case the time for all strokes have to be set to 0. Now **Point** objects are added to the stroke for each point belonging to this stroke.

stroke.addPoint (new Point(X,Y,Time))

Note that the coordinates of each **Point** object must be nonnegative, as the origin is in the top left corner!

Alternatively, if a list of Point objects is available, directly call

Stroke stroke = new Stroke(Id,startTime, stopTime, Left,Top,Right,Bottom, points).

Note that the points must be in chronological ascending order as the recognition algorithm is rather based on the pen movement than the coordinates. If for any reason the recording of the digital pen fails during a pen down and pen up event, points with a valid timestamp but no coordinates will be recorded. These failed coordinates **must** also be passed to the stroke in chronological correct order

stroke.AddPoint (new Point(Time)).

Failed coordinates are marked by negative x and y coordinates. If the points are not added in chronological ascending order, call

Stroke.sortPoints()

before adding this stroke to the region. This is only effective if all points are defined with a timestamp. Now this Stroke is added to the region calling

bool added = region.AddStroke(stroke)

A stroke is only added to the region if the strokes gravity center is contained within the (extended) regions boundaries and if the number of valid coordinates is at least 4. A minimum length in pixel could be passed to this method. In this case only strokes longer than the defined minimum length are added to this region.

Note: The strokes must be added to the region in chronological ascending order. This means the strokes Ids must be in ascending order for one region.

After all strokes are added to the region, the sequencing and classifying results are available in a list of **SequencingResult** objects. Simply call

List<SequencingResult> result = region.Recognize(bool listEmptyRegions).

The number of objects returned in result corresponds to the number of sequences the sequencing algorithm detects. A sequence is a list of strokes that makes up one pattern.

Note: Strokes that are drawn one after another in one region (these are strokes with consecutively numbered Ids) will be divided into at most two sequences.

If for example a cross is drawn in one region, after that this region is crossed out and finally a circle is drawn, only two sequences will be recognized. These will be "Crossed Out" and "Circle". A **SequencingResult** object describes one sequence. For each **SequencingResult** object in the list you obtain the SequenceNumber and for each pattern the predicted likelihood. The pattern with the highest probability therefor is the predicted one. If strokes are defined with a start time and a stop time, for each sequence a start and a stop time is available.

Sequence Nr	StartTime	StopTime	Pattern (% Probability)		
			Cross	Crossed Out	Circle
1	17.01.2013 09:00:42	17.01.2013 09:00:42	99,63%	0,1%	0,1%
2	17.01.2013 09:00:44	17.01.2013 09:00:45	0%	99,99%	0%
3	17.01.2013 09:00:47	17.01.2013 09:00:49	1,4%	0,16%	98,26%

For each possible combination of strokes, the classification is performed by a supported vector machine (SVM). Therefor the LIBSVM software is used. A trained SVM Model is available with this software. If not available during execution the recognition algorithm will throw an exception. For more information about how LIBSVM works, take a look at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

TextRegion

The **TextRegion** object is designed for recognizing digital ink. Two different kinds of recognition are considered, these are "DictionaryText" and "FreeText". When recognizing "DictionaryText" a dictionary is passed to the recognition algorithm and the recognition is limited to this dictionary. When recognizing "FreeText" no dictionary is available. All strokes that cross a region are passed to the Microsoft Ink Recognizer. If no recognizer is installed, the recognition algorithm throws an exception. See the Microsoft Ink documentation for further information.
Construct a new **TextRegion** object passing the regions boundaries and, if set, a shapeExtension, to the constructor by calling

TextRegion region = new TextRegion(Left,Top,Right,Bottom,shapeExtension).

For each stroke crossing this region build a **Stroke** object

Stroke stroke = new Stroke(Id,StartTime,stopTime,Left,Top,Right,Bottom).

A **Stroke** object is defined by an Id, a start and a stop time and a bounding rect. Now add all points belonging to this **Stroke** object. A **Point** object is defined by its x- and y-coordinate and a timestamp. One point can be added to the stroke calling

stroke.addPoint(new Point(X,Y,Time)).

Note that the points must be added to the stroke in chronological ascending order. If the points are not ordered chronologically and a timestamp is available for each point, call

Stroke.sortPoints().

Add each **Stroke** object to the region

bool added = region.AddStroke(stroke).

Only strokes, whose gravity center lies within the regions boundaries are added to this region. If a shape extension value is defined, the regions boundaries are extended by this value before. Strokes must be added to the region in ascending order. If a dictionary is available and the recognition should be limited to this dictionary, call

TextResult result = region.Recognize(LCID,dictionary,forceRecognition).

The LCID is the language code identifier for which you are retrieving the default recognizer. For a table of all LCID take a look at the table of all “language identifier constants and strings” on the Microsoft website. If the defined recognizer is not available the recognition algorithm tries to instantiate an English recognizer. Further parameters are the dictionary given as an array of strings and a Boolean indicating if recognition should be enforced. Set **forceRecognition** true, if all allowed words or characters for this region are known. If for example the region should contain single letters lsuch as “A, B, C” set the dictionary to

string[] dictionary = new string[]{A,B,C}

and **forceRecognition** to true. If on the other hand a free combination of known letters is allowed, pass in the dictionary and set **forceRecognition** to false. This might be the case if a date or an arbitrary number should be written. The dictionary then is {0,1,2,3,4,5,6,7,8,9}.

If the recognition should not be limited to a dictionary call the **Recognize** method within passing in a dictionary. The result is also obtained in a **TextResult** object

TextResult result = region.Recognize(LCID).

The language code identifier (LCID) again sets the default recognizer. The **TextResult** object gives information about the used dictionary (empty array if “FreeText” recognition was considered), whether recognition was enforced (false, if no dictionary was used), the start and stop time of the written text, the top string the Microsoft Ink Recognizer returns ,the corresponding confidence level and finally a list of all alternates and corresponding confidence levels.

Create SVM only one -Model

The classifying and sequencing of strokes in a **PatternRegion** object is based upon a supported vector machine (SVM). A trained SVM-Model is available with this library. This section describes how to create training data and train a new SVM-Model if necessary.

Create LibSVM training data

To create a valid training data set, data for each pattern must be available. These patterns are cross, crossed out, circle, slash and backslash. The format of training data is

```
<label> <index1>:<value> <index2>:<value> <index3>:value>…
.
.
.
```

Each pattern has a unique label that is a natural number. The labels used in this library are

Pattern	Label
Cross	1
Crossed Out	2
Circle	3
Slash	4
Backslash	5

Do not change the labeling, as this labeling is used for predicting! The values in the training data set are the extracted features and must not be changed or calculated by the user. Data is added region by region to the same training data file. To create new training data in the correct format, build a complete **PatternRegion** object and add crossing strokes to this object as described in the "PatternRegion Usage" section. Training data in the correct format is added to the training data file calling

region.createLibSVMProblem(List<Strokes> strokes, string filename, int label).

All strokes contained in this region, a filename and the correct label are passed to this method. The label passed into this method is dependent on the pattern the strokes in this region forms out. A file called filename.dat is created in the working directory. To obtain a complete training data file first loop through all regions containing a cross and call

region.createLibSVMProblem(region.strokes, "TrainData",1)

on each region.

Then loop through all regions that are crossed out and pass the same filename, but the correct label that is "2". Do this for all patterns in the correct order that is predetermined by the labeling. Note that training data **must** be added in this order (cross, crossed out, circle, …). Otherwise you will not obtain the correct probability estimates.

Train SVM-Model

When a complete training data file (TrainData.dat) is available, a new SVM-Model can be trained. Before training a new model cut the existing SVM_Model file from the working directory and save a copy.

Call

region.trainSVMModel(string filename);

on a **PatternRegion** object and pass the correct filename. A Model file called "**SVM_Model**" and a scale range called "**scaleRange.txt**" are created in your working directory. The scale range is used to scale data to the correct range. Copy the values in the .txt file into the static variable **private static List<double> scaleRange** in PatternRegion.cs. Make a copy of the old scale range if the old Model file should be used again. Next time the library is used, the new SVM_Model is used to classify and sequence strokes.