# Pattern Recognition, Spring 2019
# Project 2: Math Symbol Segmentation and Classification

## Prof. Zanibbi

## Due Date: 11:59pm Friday April 23, 2019

**Teams.** This project will again be completed in teams of two students. Make sure that both student names appear in the write-up and source code files.

**Grading.** 100 points total (15% of final grade)

**40 points** Write-up (6 pages max.)
**60 points** System implementation and results

- 10 LgEval Metrics
  - Summary file
  - Confusion histogram .html file (single symbols, n=1)
- 10 Preprocessing and features
- 10 Classifier
- 30 Segmenter

**Bonus: +5 / +3 / +2 points** For best performing segmenter
**Bonus: +5 / +3 / +2 points** For best performing segmenter and classifier

* Systems ranked by F-measure for symbol (object) detection, and the F-measure for correctly detected and classified symbols. A separate test set will be used for the bonus.

---

**Submission.** Through the MyCourses dropbox for Project 2, have one student from your team submit:

1. **proj2.pdf** for the write-up,
2. **system.zip:** with README, python source files, and parameter files *(including for the bonus!)*
3. **outputs.zip:** for the test set you create, should contain both of:
   (a) LgEval metric summary file
   (b) Confusion histogram .html file (for individual ground truth symbols, $n = 1$)
4. **bonus.zip**: containing O-R format .lg output files (one per .inkml file).
   - **File Names:** test file *input1.inkml* should produce output file *input1.lg*.

---

## Task

You will construct a program that reads stroke data from a CROHME .inkml file, segments strokes into symbols, classifies the symbols, and then produces a label graph (.lg) file in **Object-Relationship Format** as output. You only need to use one classifier - the classifier should be an improved version of your more 'sophisticated' classifier from Project 1 (*not* the kd-tree).

**Note!!** You do not need to define relationships between symbols in the output .lg files for this project.

**Note the 2nd!!** You must implement any features and your segmentation algorithm on your own - do not use calls to libraries providing pattern recognition operations other than for classification/regression (e.g., to select or 'score' candidate symbols (segments)).

**Data Split:** Create a *randomized* 2/3 (train) to 1/3 (test) split of the CROHME training data set. This time however, you need to split whole .inkml files into a 2/3 : 1/3 split, trying to keep the symbol distributions (i.e., prior probabilities) as close as possible. **Hint:** you can use a metric, such as the Kullback-Leibler divergence to direct your splitting algorithm.

**Bonus:** As in Project 1, you should also re-train your segmenter and classifier using the complete training data set, and then run your system on the provided bonus test set. Provide .lg files in O-R format.

## Viewing .inkml data, and File Formats

You will be using two libraries developed for the CROHME competition for this project. CROHMELib is supports .inkml file visualization and conversion to different formats. LgEval is provided for evaluation using label graphs (.lg files), which are needed to use the LgEval `evaluate` (produces metrics and error visualizations) and `confHist` (error analysis) programs.

**Viewing InkML files:** .inkml files may be loaded and viewed online at:

`http://saskatoon.cs.rit.edu/inkml_viewer`

The same program can be run on your local machine using the code provided in the CROHMELib library. This is the same code as was provided for Project 1.

**Converting .inkml to .lg:** In the updated training set, there is a directory `lg_norel` that contains the symbol segmentation and classification ground truth for each .inkml file, but no relationships. Use this version of ground truth for training and testing if you can. The stroke data still needs to be read from the .inkml files (e.g., using BeautifulSoup).

## Implementation and Results

You will implement your segmenter and classifier in Python, and create a program that both segments and classifies symbols. The program should accept an .inkml file as input, and produce a label graph (.lg) file defining symbols and symbol classes as output. **All learned parameters must be stored in a file that can be loaded at run-time** (i.e. systems will not be retrained).

Your code must contain a README explaining how to run your program using provided parameter files. **You will need to create a simple 'baseline' segmenter** to help you quickly structure your implementation and test your output files and evaluation framework, and for comparison in your report.

**Results:** You will fit your system parameters using the training set, and then evaluate performance of your system using the LgEval library on 1) the training set, and 2) the test set. Then, for the test set, use the LgEval library to obtain a summary metric file, and confusion histograms for individual symbols (n=1), and submit these with your project.

## Write-Up

Your write-up should be at most six pages long, and contain the following sections:

1. **Design.** Indicate which properties of the data or assumptions informed your choice of pre-processing, features, segmenter, and classification method. **Do not define the segmentation task, as this is defined in this document.**

2. **Preprocessing.** Summarize computations performed before measuring features (e.g. normalization, whitening, scaling, resampling stroke points, etc.).

3. **Symbol Classifier.** Provide a description of any *new* features used in your classifier, and indicate which type of classifier you used, and what changes you made to the classifier relative to Project 1. Where appropriate, provide a mathematical definition for the feature (do *not* copy and paste definitions directly from .pdf files or other sources).

4. **Segmentation.** Provide a clear description of any features used by your segmenter, and use pseudo code to define your algorithm. Also describe your segmentation algorithm, making sure to identify 1) the space of possible segments (symbol locations) considered, 2) the detector(s) used in the algorithm, and 3) how search is performed, and the final segmentation selected. Make sure that key parameters for training the segmenter are discussed (including their values).

5. **Results and Discussion.** First, describe how you produced your test/train split of the provided training data. Then, present your segmentation and classification results, for your baseline segmenter, as well as your more sophisticated segmentation algorithm. For both your training and test sets, provide recall, precision and F-measure metrics for 1) symbols ('objects' in LgEval), and 2) symbols with correct class labels. The confusion histogram for symbols should be summarized. Make use of figures (e.g. plots, bar graphs) to compare metrics.

   Also, provide the training and execution times for your classifier and segmenter.

   In your discussion of the results, provide an explanation for patterns of success and error observed in your results, and suggest improvements to your system.

6. **References.** Provide references that informed the design of your features, classifier or analysis. These should be referenced using an appropriate citation format (e.g. ACM, IEEE) in your report.