Pattern Recognition, Spring 2019
Project 1: Handwritten Math Symbol Classification

Prof. Zanibbi

Due Date: Sunday, March 31st, 2019

This project will be completed in teams of **two** students. For this first part of the project, you will create Python programs for classifying individual handwritten mathematical symbols using the CROHME isolated symbol competition data set.

Submission: One student from each team should submit the following files. **Make sure to set up a group in MyCourses (for 'P1: Project Groups'), and that both student names appear in the write-up and source code files!**

1. `project1.pdf`, the project write-up,
2. `code_params.zip`, containing your python code, README, and *all* trained parameter files needed to run your trained system,
3. `best_results.html`, HTML output from `evalSymbIsole.py` for your best classification result obtained for the 30% 'test set' taken from the provided CROHME training data
4. **Optional (bonus):** `bonus_output.csv`, containing the .csv file with your results for the provided test set without class labels

Grading

Project 1 is graded out of 100 points (15% of final grade for the course).

40 points Project write-up
60 points System implementation and results
- 5 - README explaining how to run your classifiers
- 20 - Preprocessing and feature design
- 20 - Your classifiers *(including trained parameter files) - kd-tree + another*
- 15 - Results on the test set cut out from the provided training data

Bonus: for results using the provided bonus test set (without labels):
- +5 / +3 / +2 : top 3 recognition rates
- +5 / +3 / +2 : top 3 recognition rates for the *baseline* kd-tree classifier

Program and Experiment Requirements

Classifiers: You will construct two clasifiers:

1. A kd-tree implementation of an (approximate) 1-nearest-neighbor classifier (e.g., from the scikit-learn library),
2. A second classifier of any type, that **uses the same features as your nearest neighbor classifier (i.e., they must be continuous features).**

You are welcome to write python code for your classifier, or use/modify code from open source python libraries such as:

- scikit-learn: `http://scikit-learn.org/stable/` (various methods)
- libsvm: `http://www.csie.ntu.edu.tw/~cjlin/libsvm` (well-known SVM library)
- lasagna: `https://lasagne.readthedocs.io/en/latest` (neural net library; python framework for Theano)
- pybrain: `http://pybrain.org` (neural networks, deep learning, etc.)

- orange: `http://orange.biolab.si/screenshots` (contains Quinlan's C4.5 implementation)

You must acknowledge the origin of any code in your source code and write-up.

Trained Parameters: You must provide all files needed to run your trained classifiers *without* running training again. **Use .csv files to store parameters wherever possible; if you wish to instead pickle your classifiers, make sure to use compact, binary pickle files!** For the bonus, your (non-kd-tree) classifier may be trained using the *entire* training set, so you need to include these additional parameters for the bonus in your code_params.zip file.

So you will probably provide two sets of training parameters: one for the results on your sampled training fold, and one for the bonus using the complete training data set.

Data Sets and Evaluation Tools: You will need to partition the provided data into training and testing sets, with symbols for *each class* split 70% for training and 30% for testing. These should be stored in separate .csv 'ground truth' files (in the format used by the `evalSymbols.py` evaluation program). **The split should have nearly the same prior class probabilities in the training and test sets.**

You will want to split 'junk' and 'real' symbols separately. You will produce 4 .csv files, a train/test split for the valid symbols, and a separate train/test split for the junk samples. You can combine files into a new .csv to obtain inputs for valid + junk sample experiments (e.g., using the `cat` command in Linux).

Classifier Output: Both of your classifiers should take a .csv file containing file names (and optionally classes), and then run over each of the named files. Your classifiers should then store the resulting classes from each classifier in a separate .csv file in the expected format for CROHME, e.g.:

```
kdtree-output-<input-file-name>.csv
```

for the kd-tree results. **To save time debugging, it is worth taking the time to make sure that your output file names are easy to understand!!**

Experimental Results: Your classifiers should be trained on the testing folds that you create. For **both** the kd-tree and your classifier, produce the following results:

1. Valid symbols
   - For 70% training split *(resubstitution)*
   - For 30% test split
2. Valid + junk symbols
   - For 70% training split *(resubstitution)*
   - For 30% test split

Write-Up

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

1. **Design:** Explain your choice of pre-processing, features and (non-baseline) classifier type. **Do not define the classification task; this is provided in this document.**
2. **Preprocessing and Features:** First, summarize computations performed before measuring features (e.g. normalization, whitening, scaling, resampling stroke points, etc.). Then, provide a clear description of the features used by your two classifiers. For features more complex than counting, provide your own mathematical definition for the feature (do *not* copy and paste definitions directly from .pdf files or other sources). This section should also describe any dimensionality reduction (e.g. PCA or LDA) applied to features.

3. **Classifier:** Describe the two classification algorithms used in your project. Make sure that key parameters of the classifier for training and execution are discussed (including their values) along with any modifications that you made.
4. **Results and Discussion:** Report on the training and testing recognition rates obtained, along with the errors made by each classifier (see your `evalSymbolIsole.py` HTML output file). Present the recognition results for the kd-tree classifier and your own algorithm, as described above. **Confusion matrices should be summarized** - compare primarily the most frequent confusions made by the two classifiers, as there are over 100 classes in the CROHME data set. Provide possible explanations for patterns of success and error observed in your results, and suggest improvements for your classifier.
5. **References:** provide references that informed the design of your features, classifier or analysis. These should be cited using an appropriate reference format (e.g. ACM or IEEE).