Using SL.EXE

1 Read this first

The purpose of SL.EXE (Similarity Learning) is to let researchers reproduce the results reported in the paper "Learning similarity with operator-valued large-margin classifiers" or to test the algorithm therein on other data-sets. If you haven't read the paper, SL will be of no use to you whatsoever.

The program is very rudimentary and there is absolutely no error-checking. If not used with the utmost care it is likely to crash without giving any explanation (for example if you use data-sets whose vectors have different dimensionality for training and testing). Most errors seem to be benign, and sofar SL has done no damage to my file-system, but if it does so to yours, I will assume no responsibility. In downloading and starting SL you accept complete responsibility for the consequences.

To handle image files, SL comes with some libraries proprietary to STEMMER IMAGING. This company has allowed me to have these files on my webpage as a courtesy towards me and you. Further distribution on your part would constitute a violation of copyright. If you are interested in Image-Processing I recommend you to download a free 14day-version of their software package from the STEMMER website (link on my first page).

2 Getting Started

To run SL you need Windows XP or compatible. Download SL.ZIP and unpack it into a directory of your choice. This will create an SL-subdirectory of this directory. Download and unpack the example data-sets. The EXE-file in the SL-subdirectory should be ready to run.

3 The Objects of SL

are data-sets and transformations.

3.1 Data-Sets

The data-sets consist of labeled vectors. The labels themselves are irrelevant, what matters is if vectors share a label or not. For the first version of SL a data-set is identified by a folder, which has a subfolder for each label. The subfolder contain the vectors, or data-points, as individual files. There are two possibilities for vectors:

1. Image files (*.jpg, *.bmp or *.png). SL works only with gray-scale, 8-bit/pixel images. The vectors then have dimension Width×Height and are normalized automatically to unit ℓ_2-norm before being further processed.

2. Text files (*.txt). They just have one textline giving the numerical value for each component. They are not automatically normalized, and if you want to train a transformation from such a data-set, you should normalize it (to ℓ_2-norm=1) before.

If a data-set is read into SL, the first file fixes the dimension. If any other image or textfile gives rise to a vector of different dimension, it is ignored.

Look at the example-datasets to see what the data-sets should look like. Other formats for data-sets are in preparation.

3.2 Transformations

map vectors to vectors. If you train a transformation from a dataset, the learning algorithm attempts to find a transformation which maps vectors with equal labels near to each other and vectors of different labels far from each other. The transformation should be sufficiently smooth to generalize not only to other vectors but even to datasets originating from different but related tasks. For details see the paper.

4 Menu commands

file / new T: Creates a transformation. Load a Data-set by opening the folder which contains the folders for the labels as subfolders. For nonlinear kernels the Gramian is computed and inverted, a process which can take from seconds to half an hour depending on the size of the data-set. It may be convenient to go for a coffe during this time. You should save the (raw) transformation after this step (even though it is still useless).

file / open T: Loads a transformation from file.

file / save T: Stores a transformation on file. The files may have substantial size, because they contain all the training data for the case of nonlinear kernels.

transformation / start or continue training. Runs the gradient descent. Properties of the transformation are updated every 1000 steps.

transformation / stop training. Stops the gradient descent. Necessary to change the training parameters.

transformation / truncate T. Replaces T^*T by PT^*TP, where P is the orthogonal projection to the span of the first d eigenvectors of T^*T, and d is value of the control for the maximal output dimension. Good for dimensional reduction.

transformation / view spectrum and norms of T. Shows the eigenvalues, the trace-, Hilbert-Schmidt- and operator-norms of T^*T.

operate / transform dataset and test. Load a Data-set by opening the folder which contains the folders for the labels as subfolders. The vectors in the dataset have to be equidimensional to those used in the creation of T. They are transformed by T and the resulting dataset is tested for its metric properties. For an explanation of the properties see the paper.

operate / transform dataset and store. Load a Data-set by opening the folder which contains the folders for the labels as subfolders. The vectors in the dataset have to be equidimensional to those used in the creation of T. They are transformed by T and the resulting dataset is saved to disk.

5 An Experiment

to learn about the algorithm:

1. Download and unpack the exe and example directories. The program SimLearn should start.

2. Select **new T**, open the **Rotation_Alpha** folder, and press open. After a while (3-10min) the transformation will be ready, and you should save it to file (**save T**), in case of a subsequent program crash (unlikely but possible).

3. With default settings **start and continue training** for about 100 000 iterations (Age approximately 100000). This may take about 10 min, you can pass time by occasionally **viewing the spectrum** of T*T as it evolves.

4. **Stop training** and look at the spectrum to verify that the transformation is essentially 4-dimensional.

5. Save the transformation again.

6. Run **transform dataset and test** on the **Rotation_Alpha** folder, to verify that the transformation works reasonably well on the training set. The ROC-curves of input and output data are drawn in red and blue respectively.

7. Run **transform dataset and test** on the **Rotation_Digits** folder. This is a transfer experiment, because the training algorithm hasn't seen digits before. You should get an output error-rate in the order of 1%. That the results are even slightly better on the digits than on the training set is an artifact caused by the fact, that there are more than twice as many categories in the training set.

8. Truncate the transformation to rank 4, by setting the maximal output dimension to 4 and running **truncate T**. Verify that the performance is comparable.

9. Verify the deterioration of performance with truncations to 3 and 2 dimensions.

10. Stop the program and restart it (this is good to avoid memory-leaks, due to bad programming).

11. Repeat the creation of T with a linear kernel and iterate to 100000. Notice that the creation part goes much faster. Verify that the performance of the linear transformation is comparable to the gaussian kernel. This is not generic behaviour, but can be linked to the properties of the rotation invariance of the data.

Repeat the above steps with the Scale-dataset (**Scale_Alpha** and **Scale_Digits**), iterating to 100 000 and to 1000 000 iterations. This will of course take some time, but you can verify that the performance improves much more slowly now, that the essential rank of the resulting transformations is much higher, and that the performance with the linear kernel is inferior to the Gaussian case - which seems to be the generic behaviour. The experiments with the Rotation×Scale datasets are best run overnight.

If you use your own data-sets, I am very curious about the results. If they don't consist of images, don't forget to normalize the vectors to unit ℓ_2-norm before using them with SL.

Andreas Maurer