Algorithmical and Statistical Modelling, Fall 2012

Miniproject 1: Modelling the distribution of digits

Summary. In this miniproject, the objective is to model the distribution of the pixel values of the digits 0, ..., 9, using as a source for the model data from the widely used MNIST digit database.

Project coach. This miniproject will be supervised and coached by Mantas Lukosevicius (who did the same MP many years ago, when he was still a grad student... now he is just finishied his PhD studies in the Machine Learning group). Email: m.lukosevicius at jacobs-university.de. Please consult him in all cases of doubt, anxiety, or euphoria.

Data. We use the MNIST database (http://yann.lecun.com/exdb/mnist/), a set of 70,000 pixel images of handwritten digits which is widely used as a benchmark dataset. You can download Matlab .mat files containing these data from http://minds.jacobs-university.de/sites/default/files/uploads/teaching/share/MNISTdata.zip (12.7 MB). The original MNIST file format is proprietary; I obtained these Matlab versions by running the transformation `converter.m` which I fetched from http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html. After unzipping, you will have 10 files named `digit`*i*`.mat` and `test`*i*`.mat`, where *i* runs from 0 to 9. Loading any of these into Matlab (with `load digit`*i*`.mat`) will give you a workspace variable `D` which is a matrix of size N^i x 784. Each row in this matrix contains the grayscale pixel values of a 28 x 28 = 784 pixel image of a handwritten digit. The file `digit1.mat` contains only „1" digits, etc. N^i, the number of samples per digit, varies a bit from file to file; it is in the order of 60,000 for the `digit` and in the order of 1,000 for the `test` samples. In order to visualize these data, you can use the Matlab helper function http://minds.jacobs-university.de/sites/default/files/uploads/teaching/share/drawDigit.m which I wrote for your (and my) convenience.

Challenge and task. From a theoretical (and application...!) perspective, the digits from the MNIST database are a sample from a 784-dimensional RV. Here I adopt a view that we are dealing with 10 different RVs, one for each digit; that is, I want to keep the model of the „0" digits separate from the „1" model, etc. Let X^i, where $i = 0, ..., 9$, denote the 784-dimensional RV that has 28x28 pixel images of the i-th digit as values. Your task: for each digit i, use the data from `digit`*i*`.mat` to obtain an estimate of the distribution P_{X^i} of X^i. There are at least two challenges involved:

1. The *representation* challenge: how to write down / implement $P(X^i)$, concretely? Being a 784-dimensional RV, a histogram or an analytical pdf is out of the question. The first thing to do is to take a decision regarding the observation space – the space hosting the pixel images. There are (at least) two options. First, one can adhere to the integer format of the MNIST pixel vectors. Then, while the observation space is certainly large – namely, it has 256^{784} elements –, it is still finite, so P_{X^i} is a discrete distribution with nonzero values for individual pixel images. Second, one can interpret the pixel vectors as real-valued vectors. Then one would be dealing with a continuous distribution in the hypercube $[0, 255]^{784}$. You may decide to go either way, discrete or continuous. At any rate, you are requested to deliver a Matlab function `dist.m`, which takes an index i as the first argument and a 784-dimensional image-coding vectors π as a second argument and returns ...
    a. ... in the discrete case, where π is an integer vector: a log probability `dist.m`$(i, \pi) = \log(P_{X^i}(\pi))$. Note that your function `dist.m` must satisfy

$$\sum_{\text{all possible pixel image vectors } \pi} \exp(\texttt{dist.m}(i,\pi)) = 1$$

    b. ... in the continuous case where π is a real vector: a log density `dist.m`$(i, \pi) = \log(p(\pi))$, where p is a pdf on $[0, 255]^{784}$. Here it must hold that

$$\int_{[0,255]^{784}} \exp(\texttt{dist.m}(i,\pi))\, d\mathbf{x} = 1$$

Note. Your function `dist.m`(i, π) may read in (by load commands) parameters or auxiliary data structures, depending on i.

2. The *learning* challenge: how to optimize the parameters of your model such that on the one hand, it extracts as much information from the `digit`*i*`.mat`'s as possible, while on the other hand it doesn't overfit? To get a first understanding of this core issue of machine learning, please read the second (introductory) chapter in the Machine Learning lecture notes which you find at http://minds.jacobs-university.de/sites/default/files/uploads/teaching/lectureNotes/LN_ML_Fall11.pdf, especially the parts on overfitting („bias-variance-dilemma") and cross-validation.

Approach. Is completely up to you -- . You may use whatever you pick up from the lecture, or anything else you happen to glean from elsewhere (MOG's or Parzen Windows would be reasonable candidates, but not the only ones).

Evaluation. The quality of your model `dist.m` (where $i = 0, ..., 9$) is measured in two ways, using the `test`*i*`.mat` samples from the MNIST data:

1. Compute the average test log likelihood (LL)

$$avLLtest = \frac{1}{10} \sum_{i=0,...,9} \frac{1}{N^i_{test}} \sum_{\pi \in \texttt{test}i} \texttt{dist.m}(i,\pi)$$

   The larger this value, the better the model. (Note that values obtained for discrete vs. Continuous models are comparable!)

2. Use your `dist.m` to *classify* the test patterns. Concretely, for each of the test patterns π (from all of the ten test sets), compute a decision d for one class index by $d = \operatorname{argmax}_i \texttt{dist.m}(i,\pi)$.

   Then report the percentage of misclassifications across all test patterns, similar to the results reported in http://yann.lecun.com/exdb/mnist/.

Important warning. *Never* use the test data for optimizing your model!! You should use the test data only **once**: namely, when you evaluate the test performance for inclusion in the report.

Deliverables. Please send (by email both to m.lukosevicius and h.jaeger) the following items:

1. a Matlab function `dist.m`, as described above, and any files that may be called or loaded when evaluating this function (if you go for a non-parametric solution which needs the raw data `digit`*i*`.mat`, don't supply these along with your solution – we only want the algorithm, not the raw data back...);
2. any Matlab functions or scripts that you used for the parameter optimization (very likely you will use cross-validation);
3. the Matlab script by which you computed your average test LL and the classification error rate;
4. a report (order of 5++ pages) where you explain what approach you took, and where you report the average test LL and the misclassification rate.

Teaming. You may (but don't have to) work in teams of 2 (one set of deliverables per team). I suggest that students with a prior exposure to machine learning team up with novices.

Grading. The project will be graded with percentage points, based (i) on the formal qualities of the report and (b) the depth of understanding, sweat, ingenuity, and results – in short, scientific quality. Both components count equally. It is expected that the delivered code is reasonably documented (inline and in a README for instance) to a degree that Mantas can easily understand the code's relation to the report. Point subtractions for poorly structured, poorly documented code. You can find a model example of an almost perfect report (except for the Lithuanian syndrome: lack of definite articles...) at http://minds.jacobs-university.de/sites/default/files/uploads/teaching/share/MP3_ML_Fall06_VSakenas.pdf.

Bonus points. All delivered projects will be order-ranked according to the average test LL and classification error rate achieved. The two rank numbers per delivery thus obtained will be averaged,

yielding a total ordering. The authors of the first, second and third running entries will be awarded with 5, 3, 2 extra points which will be fully added to the average point score achieved over the semester in the final grade calculation at the end of the semester. If you teamed, the extra points will be split between the two teamers. Very valuable (namely, undiluted by averaging) points.

Two-stage submission, timelines. There will be a two-stage submission process. In the first stage, which is **optional**, only submit a draft report. It will quickly be checked and returned, giving hopefully valuable feedback on report writing. Deadline for this optional report checkin is Wednesday Oct 17, midnight. The ultimate deadline then is two weeks later (Wed Oct 31 midnight); then all deliverables are due (see above). Per day of delay there is a 10% subtraction of points (except bonus points).