TDT4173: Machine Learning and Case-Based Reasoning

Assignment 5

March 31, 2016

- **Delivery deadline: April 21, 2016** by 22:00.
- You can work **on your own** or **in groups of up to three people**. If you are working in a group, make sure to deliver together and not separately.
- Deliver your solution on *itslearning* before the deadline.
- Please upload your report as a PDF file, and package your code into an archive (e.g. zip, rar, tar).
- The programming tasks may be completed in the programming language of your choice.
- Your code is part of your delivery, so please make sure that your code is well-documented and as readable as possible. The first section of the report must explain how to run the system/code.

Learning Objectives: Gain experience with (a) how to set up a pipeline for a difficult machine learning problem, and (b) how to read documentation of a machine learning library and use it to solve a problem.

1 Optical Character Recognition

Optical character recognition (OCR) is a fairly old area in computer science that concerns the electronic, or mechanical, process of transforming images of characters into digital text. In this assignment you will get the opportunity to implement such a system using machine learning. An explicit listing of the deliverables can be found at the end of the assignment text. What follows is a brief explanation of the dataset and the three main components of your OCR system.

Dataset: Chars74K Lite

We will be using a modified version of the Chars74K dataset by de Campos et al. from 2009[1, 2]. The complete dataset includes a range of symbols in both English and Kannada, however, the modified version we will be using only encompasses majuscule and minuscule letters from the English alphabet, i.e. *A-Za-z*. Please consider visiting the dataset website if you want to know more about the Chars74K dataset.

Below is a summary of the modified dataset as it is presented on *itslearning*:

- 26 classes (*a*-*z*).
- 7112 images of characters stored as 8-bit grayscale images with JPEG compression.
- Each image is of size 20×20px, which means that each image, or feature vector, contains 400 features/pixels.
- Classes are not explicitly labelled anywhere other than by the containing directory, e.g., all images of the ***a*** or ***A*** symbol reside in the directory named "*a*".
- The dataset does not differentiate between upper- and lower-case letters, e.g., both ***f*** and ***F*** belong to the same class.

Please refer back to this overview if you are unsure that you have loaded the data correctly. A good set of sanity checks you can do is to (i) count up the number of images, (ii) print out one of the images to see that the intensity values lie in the 8-bit, or normalised[1], range, and (iii) visualise a couple of images using a plotting library. 16 images randomly sampled from the modified Chars74K dataset can be seen in Figure 1.

Figure 1: 16 random images sampled from the modified version of the Chars74K dataset. Each image is a 20×20px 8-bit grayscale image.

Feature Engineering

As is the case with most machine learning applications, the first thing we have to do is to engineer *good* features. That is to say, if we have data that lie on a high dimensional manifold (many features), what we are interested in is to engineer features that, when varied, will allow us to discriminate between classes. For example, when trying to recognise whether or not an image contains a person, we are not interested in the rotation or scale of that person. In this case, a *good* feature vector will contain features that do *not* vary (much) with respect to the distance or angle of a person; however, the features should vary a lot depending on whether or not a person is in the image.

Given Occam's razor we can assume that there exists a low-dimensional space that allows us to discriminate between our data perfectly. The features that make up this space

[1]It is common to transform intensity values in a set of images to values between 0 and 1 before any processing takes place. Assuming we have an 8-bit image, with intensity values between 0 and 255, all we have to do is change the data type to `float` and divide by 255.

are the ones we are most interested in, however, they are typically very difficult to find. In the case of the dataset in this assignment, we have 400 grayscale intensity values, or features, per image, however, it is safe to assume that we do not actually need 400 real-valued features in order to differentiate between English letters. While we will probably not find the *ideal* set of features, we may be able to improve and/or reduce them.

Features are often engineered by domain experts, however, automatic procedures, such as unsupervised learning, do exist and are heavily used. Below are a series of topics related to feature engineering, that may prove useful for the assignment:

- Manually cropping, or otherwise removing parts of the image that are uninteresting.
- Feature scaling, such as ensuring that features (i) reside in the 0 to 1 range, (ii) have zero mean, and (iii) have unit variance.
- Dimensionality reduction, such as projecting our high dimensional data to the top n principal components, using principal component analysis (PCA)[3, 4].
- Image processing techniques, such as noise removal or edge detection filters.
- Deep learning, such as convolutional neural networks that allow us to learn hierarchical features.
- Feature descriptors, such as the scale-invariant feature transform (SIFT)[5, 6], histogram of oriented gradients (HOG)[7], and local binary patterns (LBP)[8]. These are algorithms that create feature vectors with *good* properties.

Classification

The main goal of the assignment will be to train a model that assigns a class (a-z) according to a 20×20px distribution of pixel intensities. Seeing as the classes are discrete, it follows that we can do a process called classification. A popular approach to create a classifier is to model the mapping from the inputs $\mathbf{x}$ to the output class y. This can be done by fitting a model of the form $\Pr(y|\mathbf{x})$ and is called a *discriminative* classifier. The relationships defining the possible mappings between the inputs and output are determined by the parameters of the model. The symbol θ is usually used as a catch-all term for these parameters. Intuitively, this means that calculating the probability of a new datum is done by calculating the posterior $\Pr(y|\mathbf{x}, \theta)$.

In this assignment there are 26 classes, meaning that we have a multiclass classification problem. The classes are the 26 letters of the English alphabet. Instead of using alphabetical letters (a-z) it is helpful to let our labels be numerical ($0-25$), e.g., $\boldsymbol{a}$ corresponds to 0, while $\boldsymbol{z}$ corresponds to 25. A number of machine learning models can be used for this assignment, such as logistic regression, support vector machines, k-nearest neighbour, decision trees, naive Bayes, artificial neural networks, and ensemble approaches. Several of these models do not care about the spatial structure in the images in our dataset. If you are using a model such as this, e.g. logistic regression, it is often assumed that all images are reshaped,

flattened, or unrolled to regular 1-d feature vectors[2]. Please investigate how this can be achieved in the programming language you have elected to use.

Detection

Object detection or object localisation, is a difficult problem where we attempt to find objects within a scene. In this assignment we will be doing a special case of object detection called character detection. The concept of finding the exact position of a character in an image seems like a daunting task, however, with the help of a character classifier (see above) and a technique called a *sliding window detector* we may be able to overcome it. The idea is to divide the incoming image, e.g., a document of handwritten or printed text, into a slew of overlapping patches at different locations, and then classify each patch based on what kind of character it may contain. With each patch being associated with a probability estimate, per class, produced by a classifier, we can use a threshold to localise patches that are likely to contain a letter.

In practice, a sliding window detector can be implemented by looping over an image[3] with a certain stride, and for each (x, y) pixel coordinate extract a window of nearby pixels, using a particular window size. In our case it is natural to let the window size be 20×20px, however, in order to detect characters at multiple scales it is important to have different window sizes. Keep in mind that with a classifier with a fixed-sized input constraint, the window of pixels must be resized to 20×20px before applying the classifier.

As previously mentioned, character detection is a difficult task and there are many issues that must be considered when creating your OCR system. Common issues include overlapping patches that are difficult to prune away using a simple threshold, multiple character classes with high probability for a single patch, scale and orientation issues, and how to interpret background patches that should not be assigned to any class.

Machine Learning Libraries

Implementing most of the machine learning techniques mentioned above from scratch is outside the scope of this assignment. Rather, you should use a general-purpose machine learning library. In other words, take advantage of what is already out there instead of reinventing the wheel. Getting more experience with reading documentation is a core part of this assignment.

Below is a selection of four common general-purpose machine learning libraries using four different programming languages. Click on the bolded name of the library to visit the library webpage.

- **JuliaStats** A family of statistics and machine learning packages in *Julia*.

[2]Using the raw input images, a flattened image from the assignment dataset will have 400 elements.

[3]By looping over an image we mean setting up a nested loop that iterate over each pixel in scanline.

- **Machine Learning Toolbox** A suit of machine learning tools in the *MATLAB* environment. Can be combined with the Computer Vision System Toolbox to build systems using, for example, feature descriptors.
- **scikit-learn**[9] The most popular general-purpose library for machine learning in *Python*. Has tools for just about anything related to machine learning. Does not support artificial neural networks very well.
- **Weka**[10] A popular collection of machine learning algorithms for data mining tasks in *Java*. Contains tools for data pre-processing, classification, regression, clustering, association rules, and visualisation.

In addition to the libraries listed above, there are an abundance of specialised libraries for all sorts of machine learning techniques. Some notable examples for deep learning are Theano, mxnet, Torch, Caffe, and Tensorflow[11, 12, 13, 14, 15, 16]. We do not recommend using deep learning unless you are either already well-versed in one of these libraries or are heavily invested in the field.

2 Deliverables

Your deliverables must include the following:

- Source code for all aspects of your OCR system.
- Report of approximately 4 pages (not including references).
- Any additional material, e.g. detection images.

A detailed overview of what you must do for each part can be seen in the next two sections. Read both parts before starting on the assignment.

Programming

The programming part consists of loading and pre-processing the dataset provided on *itslearning*, creating a classifier for discriminating between characters in the dataset, and creating a system for character detection. Keep in mind that you should try to use already existing libraries for most of the tasks; it is not recommended that you try to implement different kinds of models on your own.

1. Load the dataset and split it into a training and test set. The latter is only to be used for evaluating your model.
2. Generate numerical labels for the dataset. This is easily done during loading because each directory of images is associated to only one class. For example, when loading in all n_b images of the letter $\boldsymbol{b}$, create an array of size n_b containing all ones (1 is the class associated with the letter $\boldsymbol{b}$).

3. Try out at least **two** different ways of *pre-processing* your data (feature engineering). What techniques to use is completely up to you, however, you must be able to both justify and explain the techniques in the report.

4. Try out at least **two** different *models* for classifying the dataset according to the 26 labels. You can use any model, as long as it can be used for classification, you are able to explain it in the report, and can justify your model selection. While there are no performance requirements, you are expected to be able to experiment with different models and select the best ones. Have a look at the paper by de Campos et al. to see what the baseline performance might look like on the Chars74K dataset[2].

5. Create a simple sliding window detector using the method described above. The detector should yield probabilities for each patch using the best classification model you have trained. You may want to experiment with simple extensions to your detector, e.g. different scales, but this is not required.

6. Add functionality for drawing coloured boxes around the patches with the highest probability. This can be done by keeping track of the (x, y) pixel coordinate of the current window (when looping over the image) and also the pixel coordinate you get by adding the window size, i.e. $(x + size_x, y + size_y)$, where $size_x$ and $size_y$ are the horizontal and vertical window size, respectively.

The use of data augmentation[4] is encouraged, however, using pre-trained model parameters is **not** allowed.

Report

The report must include the following:

1. A brief overview of the whole system. This includes a short explanation on how to run it. What libraries are your system using?

2. A description of the **two** *pre-processing* techniques you have tried out. Why did you select these techniques? What worked/did not work? Are there any techniques that you wanted to try out but were not able to?

3. A description of the **two** *models* you elected to try. Why did you select these models? After having looked at the dataset, did you have any initial idea about what kind of model that might work well? Are there any additional models you would have liked to try?

4. A critical evaluation of your two models. How are you measuring their performance? How did they do? Which model gave the best results? Include at least five predictions in the report (both good and bad).

[4]Data augmentation is a set of techniques used to artifically increase the size of the dataset. This includes, for example, flipping images, adding small amounts of noise, and affine transformations.

5. Find an image online or take an image of a piece of paper with written or printed letters on it and use your character detector on it. Give an evaluation of your detection system. How did it do? Describe any improvements you made to your detector. Discuss how you can improve your system further. Are there any particular writing styles or typefaces your system is good/bad at detecting? Include a few screenshots of how your detector fared on the images you have gathered.

6. What is the weakest and strongest component of your OCR system (feature engineering, character classification, and character detection)? Explain your answer.

7. What went good/bad with the project? Any lessons learned?

References

[1] *The Chars74K dataset - Character Recognition in Natural Images* 2012 URL: `http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/` (visited on 2016-03) (cit. on p. 1)

[2] Teo de Campos, Bodla Rakesh Babu, and Manik Varma "Character Recognition in Natural Images." in: *VISAPP (2)* 2009, pp. 273–280 (cit. on pp. 1, 6)

[3] Karl Pearson *On Lines and Planes of Closest Fit to System of Points in Space. Philiosophical Magazine, 2, 559-572* 1901 (cit. on p. 3)

[4] Harold Hotelling "Analysis of a complex of statistical variables into principal components." in: *Journal of educational psychology* 24.6 (1933), p. 417 (cit. on p. 3)

[5] David G Lowe "Object recognition from local scale-invariant features" in: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on* vol. 2 Ieee 1999, pp. 1150–1157 (cit. on p. 3)

[6] David G Lowe "Distinctive image features from scale-invariant keypoints" in: *International journal of computer vision* 60.2 (2004), pp. 91–110 (cit. on p. 3)

[7] Navneet Dalal and Bill Triggs "Histograms of oriented gradients for human detection" in: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* vol. 1 IEEE 2005, pp. 886–893 (cit. on p. 3)

[8] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns" in: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24.7 (2002), pp. 971–987 (cit. on p. 3)

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay "Scikit-learn: Machine Learning in Python" in: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 5)

[10] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten "The WEKA data mining software: an update" in: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10–18 (cit. on p. 5)

[11] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio "Theano: new features and speed improvements" in: *arXiv preprint arXiv:1211.5590* (2012) (cit. on p. 5)

[12] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio "Theano: a CPU and GPU math expression compiler" in: *Proceedings of the Python for scientific computing conference (SciPy)* vol. 4 Austin, TX 2010, p. 3 (cit. on p. 5)

[13] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems" in: *arXiv preprint arXiv:1512.01274* (2015) (cit. on p. 5)

[14] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet "Torch7: A matlab-like environment for machine learning" in: *BigLearn, NIPS Workshop* EPFL-CONF-192376 2011 (cit. on p. 5)

[15] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell "Caffe: Convolutional Architecture for Fast Feature Embedding" in: *arXiv preprint arXiv:1408.5093* (2014) (cit. on p. 5)

[16] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems* Software available from tensorflow.org 2015 URL: `http://tensorflow.org/` (cit. on p. 5)