Number recognition for use in a Sudoku solver

Abstract

The goal of this work is to implement number recognition for a Soduko solver. First we will describe the generation of synthetic data, in this case labelled images of numbers. Then we compare different machine learning techniques for supervised learning using the generated data. Once the training is complete, images of real Sudokus are captured using a webcam and preprocessed to be predicted by the learned models. We evaluate the different approaches by analyzing their performance on different data sets.

1. Introduction

Number and letter recognition is key in many real-world applications. Digitalizing print media, automated sorting of letters, or recognition of signs in robotics or driver assistance systems require well performing algorithms for character recognition. In the following we analyze the performance of different techniques for number recognition, namely Neural Networks (NN), Support Vector Machines (SVM) and Logistic Regression (LR), by using them in an automated Sudoku solver.

2. Overview

Figure 2 shows the basic data flow and the participating data manipulators. First of all we generate data by sampling position, rotation, size and color of a number and place it based on these values on a white 20x20 pixel image. Examples of 9 such generated images are shown in Figure 1.The corresponding labels are defined by the name of the folder the images are stored in. These images are preprocessed (applying greyscale) by an octave script and written to an input matrix and a target vector. All pixels of one image are encoded in one row of the input matrix and the corresponding label is stored in the row of the target vector. The

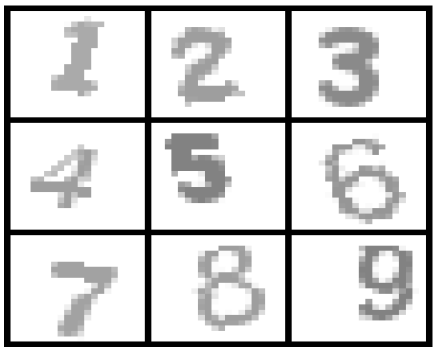

Figure 1. Examples of generated numbers

data is dumped to a "cache.mat" file which allows us to apply many different tools and programming languages to the dataset. In our case we use it in python and octave. The input matrix $X^{n\text{x}400}$ and the target

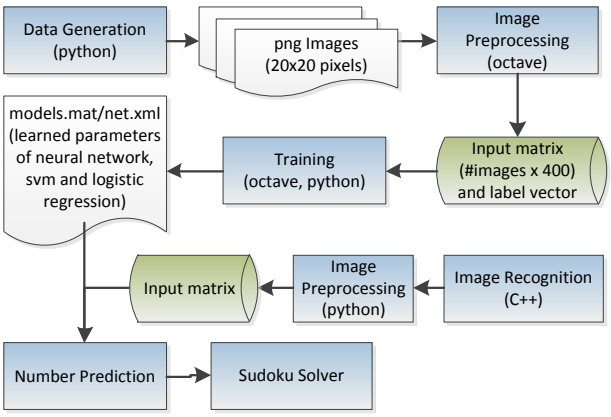

Figure 2. Overview

vector y are the inputs for training NN, SVM and LR, as described in Section 3. After training, the learned models/parameters are saved in ".mat" files that can be loaded by the number prediction. The other input for number prediction is an input matrix of the same form as X with a row entry for each Sudoku field that was detected to contain a number. This implies that we do not train our models to detect empty fields. We could detect empty fields with high accuracy by performing some preprocessing steps on the images (filling borders, adjusting thresholds etc.) thus applying a

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

machine learning technique does not seem appropriate for this task.

3. Training

3.1. Neural Network

Applying a NN seems promising as it was already successfully applied to the quite similar problem of detecting handwritten digits.[1] For the NN we choose a simple architecture with 25 hidden sigmoidal units (+ bias unit), as shown in figure 3. The size of the input layer is 400 (given by the 20 x 20 pixel images) and the size of the output layer is defined by the number of classes, in this case the numbers 1 to 9. This simple architecture turned out to perform pretty well compared to more complex ones with a higher number of hidden layers. Training is performed with error backpropagation, thus minimizing the error function

$$E(w) = \sum_{n=1}^{N} E_n(w)$$
$$\text{with} \quad E_n(w) = \frac{1}{2}\sum_{k}(y_{nk} - t_{nk})$$
$$\text{and} \qquad y_k = \sum_{i} w_{ki}x_i$$

subject to w. For NNs we test two implementations.

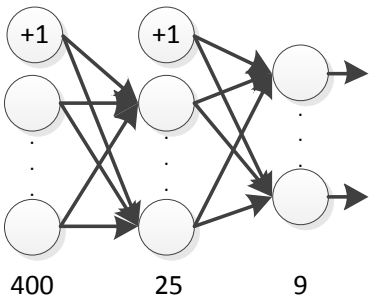

Figure 3. Neural Network with one hidden layer

On the one hand we use a partly self-developed octave code for two-layer NNs which was written as part of the stanford machine learning course[2], on the other hand we include PyBrain (Schaul et al., 2010) for comparison and easier evaluation of more sophisticated network architectures. To find a good value for the regularization parameter λ we perform a grid search with k-fold cross-validation which gives us a good performance when choosing $\lambda = 1$.

[1] http://www.ml-class.org
[2] http://www.ml-class.org

3.2. Support Vector Machines

The next technique we apply to the discussed problem are Support Vector Machines (Bishop, 2006) which try to solve the following optimization problem:

$$\min_{w,b} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i$$
$$\text{subject to} \quad y_i(w^T\phi(x_i) + b) - 1 + \xi_i \geq 0$$
$$\text{and} \quad \xi_i \geq 0.$$

Using a radial basis function (RBF) as kernel we perform a grid search over C and γ to find those parameters that provide highest accuracy on the test set. We use the open source library "libsvm" (Chang & Lin, 2011) wrapped by some octave scripts for setting up data sets and performing grid search with k-fold cross-validation. Grid search with cross validation resulted in accuracies illustrated in Figure 4. We can see that there is a certain area of (C, γ) pairs which provide an accuracy close to 100%. To prevent overfitting we decide to choose C and γ by cross checking good pairs against another validation set. Further refinement results in the following values $C = 20$ and $\gamma = 0.1$.

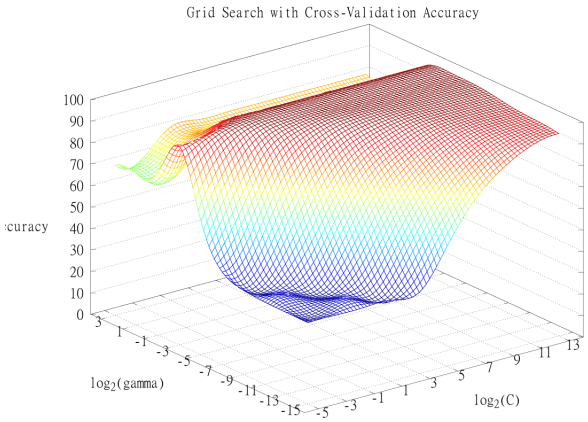

Figure 4.

3.3. Logistic Regression

Finally we apply Logistic Regression to the datasets. Again we use a (vectorized) octave implementation for learning and prediction. Therefore we minimize the (regularized) error function

$$E(w) = \frac{1}{n}(-y^T ln(\Phi w) + (1 - y^T)ln(1 - \Phi w)) + \frac{\lambda}{2n}w^T w$$

with respect to w. Minimization is performed by an algorithm[3] that was provided during the stanford machine learning course. Grid search and k-fold cross-validation recommend a value for the regularization parameter λ.

4. Image Recognition and Preprocessing

Image recognition is written in C by extensive use of the open source library OpenCV[4]. Images captured by the webcam are continously analyzed to find a Sudoku square. Images are transformed to greyscale, then an adaptive threshold algorithm is used to preprocess the image for contour detection by OpenCV. OpenCV returns a list of contours which are further checked for certain criteria like having four corners or being of certain size. Since the contours are not detected precise enough lines are fitted through six checkpoints on each edge in order to achieve subpixel accuracy. The content enclosed by the corners is perspectively warped to fit a predefined square size. Finally, the thresholded Sudoku field is saved to disk. A python script is then used to split the whole field into 9x9 subfields with each subfield being flood filled around the corners for noise reduction. The format of the subfields now equals to the one provided by the generated images which allows us to apply the trained NN/SVM/LR models and, finally, apply a Sudoku solver.

5. Evaluation

To evaluate the performance of different approaches we define the following data sets:

- D1: 4500 generated images
- D2: $\sim$ 1000 gathered images
- D3: D1 $\cup$ D2
- DVal: images of some newly captured Sudoku

As expected, on data sets D1, D2 and D3 all approaches performed pretty well as they were used for training. DVal is the data set that contains completely new images that were never used for training by any of the models. Thus, these values are the most interesting ones for evaluation. SVM and NN have comparable performance. Yet, the SVM is cheaper to train which is why applying an SVM seems more appropriate than using a NN. LR is outperformed by the other approaches but predicts with acceptable performance and requires less effort to train.

Table 1. Classification accuracies for Neural Network, SVM and Logistic Regression on different data sets. All values are rounded averages!

Data	NN	pyNN	SVM	LR	Winner
D1	86.0	84.0	98.0	83.80	SVM
D2	93.41	95.94	94.41	91.36	pyNN
D3	90.27	91.53		95.01	88.42
DVal	87.36	86.18	91.76	83.33	SVM

6. Future Work

Several ideas arose during this project. A first step could be to implement online learning to empower the models to evolute over time and increase their accuracy with more Sudokus being scanned. When detecting new Sudokus it turned out that empty field detection was not as reliable as expected, especially when light conditions were bad. Improving this part of the system or maybe doing it from a machine learning point of view (adding a new label for empty fields) might be a good idea. Another improvement of the systems accuracy could be achieved by including information of the Sudoku solver into field prediction. The solver could inform about impossible field combinations forcing the number prediction script to maybe choose a less probable number combination from the model point of view, yet, being a possible one.

References

Bishop, Christopher M. *(Pattern Recognition and Machine Learning) By Bishop, Christopher M. (Author) Hardcover on (08 , 2006)*. Springer, 8 2006.

Chang, C.C. and Lin, C.J. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

Schaul, T., Bayer, J., Wierstra, D., Sun, Y., Felder, M., Sehnke, F., Rückstieß, T., and Schmidhuber, J. Pybrain. *The Journal of Machine Learning Research*, 11:743–746, 2010.

[3] http://sprinkler.googlecode.com/svn/trunk/regression/fmincg.m

[4] http://opencv.willowgarage.com/wiki/