Optical Character Recognition

Approach using k-Nearest Neighbors and Support Vector Machines

Alla Marchenko, Olav Markussen and Geir Kulia

April 27, 2017

1 Introduction

This report is presenting two approaches to Optical Character Recognition (OCR). The approaches applied are k-Nearest Neighbor Classifier and Linear Support Vector Classifier. A general overview of the implementation is shown in Figure 1. The data is loaded from the Char74k-Lite dataset and diveded, at random, into two databases. 80 % is selected as the training set, and 20 % is the test set. The data is then preprocessed, as described in section 2. A model is trained on the training set, before the model is passed to the classifier. The classifier is described in section 3. The system output is the classifier error, which is given by

$$E = \frac{N_{\text{fail}}}{N_{\text{test}}} \tag{1}$$

where N_{fail} is the number of wrong classifications in the test set and N_{test} is the total number of samples in the test set.

We implemented the Linear Support Vector Classifier in both Python and Matlab. The Python version was used to optimize detection parameters with grid search and calculate the overall error E_{LSVC}, while the Matlab version uses inheret high level Matlab features for displaying data, and its superb matrix handling.

The Python program mainly used two packages: Scikit-image and Scikit-learn. Scikit-image is a opensource image processing library, that was manly used for preprocessing. Scikit-learn is a library that can be used for classification, regression, clustering, dimensional reduction, model selection, and also preprocessing. It was heavely applied during this project. It is hard to have an complete overview of A complete list of all libraries used in the python implementation are shown in Appendix A. It is hard to have a complete overview of Matlab toolboxes, as namespacing is all but apparent in Matlab. However, the two main toolboxes used in this project were Computer Vision System Toolbox and Statistics and the Machine Learning Toolbox.

The software uses UNIX convention for file paths, so this might not work on Windows. To start the python program: type

```
$ python3 main.py
```

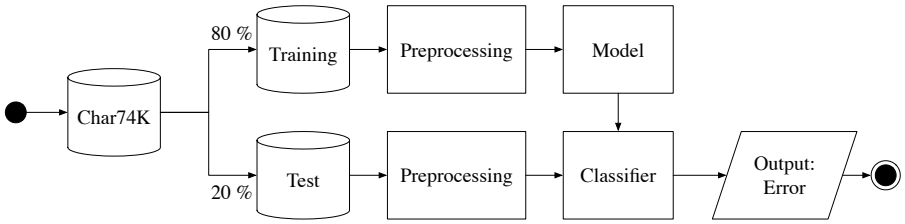

Figure 1: Model of OCR system.

(a) Raw image. (b) Denoised image. (c) Binary image. (d) Histogram of Oriented Gradients.

Figure 2: Example of random selected image before and after preprocessing. The colors has been altered using a color map to visually enhance variance.

into your terminal. To run the Matlab program, type

```
$ matlab -nodesktop -nosplash
>> run SVM4OCR
```

To change the window size, simply change the variable `step` in the file `SVM4OCR.m`.

2 Feature Engineering

In this section, the preprocessings of the images are shown and discussed. The preprocessing is preformed to enable easier classification for the classifier. Several preprocessing tools were explored to make the character easier to recognize for the classifier. The first preprocessing tool applied was total-variation denoising on n-dimensional images [**?**]. This technique reduces the total variation of an image. This is useful to avoid classifying noise, as the picture will now contain fewer components with high spacial frequency. An example of this denoising technique is shown in the image Figure 2b.

To increase contrast, we used Otsu's method, which is a way to perform image thresholding based on clustering [**?**] automatically. After thresholding, the image was morphologically closed. This is a standard technique in image processing to remove small wholes in binary images. The result is shown in Figure 2c. All though the result was visually pleasing; it did not decrease the error rate. Therefore, this method was discarded from further analysis.

The last preprocessing tool applied before dimention reduction was Histogram of Oriented Gradients. It is a feature descriptor uses local object appearences and shapes and describe them as a distribution of intensity gradients. It can be interpreted as the spacial derivative of the intencity of the image. The purpose of this step is to format the image for easier detection later as each element now carry more information than the pixels in the original.

One feature engineering method used and not covered by this section is Principal Component Analysis. Because its application is heavily based on the k-nearest neighbor algorithm, it will be covered in subsubsection 3.1.1.

3 Classifier

In this section, we will discuss two approaches for classifying characters: the k-nearest neighbors' algorithm and linear support vector classifier. We will also discuss task-tailored feature-engineering used to enhance the result, which is not discussed in the previous chapter.

After examining the data set Char74K-lite, the author's initial thought was to use convolutional neural networks (CNN). CNNs have been used to recognize far more complicated objects than characters [**?**]; hence, it should be a trivial task to apply CNNs to recognize letters [**?**]. However, due to the limited time for training, lack of heavyweight graphical processing units, and that it is discouraged by the assignment, we choose not to use convolutional neural networks.

The authors had a limited knowledge of Principal Component Analysis and Support Vector Machines and thought this would be an excellent opportunity to gain more theoretical knowledge as well as hands-on experience with these methods. We, therefore, chose to pursuit one application with Principal component analysis and k-nearest neighbors algorithm as a classifier. The other approach was to use linear support vector classifiers. Support vector classifiers were implemented in both Python and Matlab. Both Python versions used the preprocessing described in section 2, while the Matlab version only applied HOG.

3.1 k-nearest neighbors algorithm with Principal component analysis

The k-nearest neighbors algorithm (k-NN) is a simple non-parametric classification method [**?**, pp. 231-236]. A training set is embedded in a n-dimensional space and used as a prediction model. The model can then predict new data by estimating the shortest distance to the k nearest classes in the n-dimensional space, as demonstrated in Figure 3. The unknown character is classified, apparently enough, as the median of the k-nearest neighbors classes.

It is a tangible challenge and computationally heavy to perform classification on high dimensional (i.e. large n) data with k-NN. Therefore, a dimension

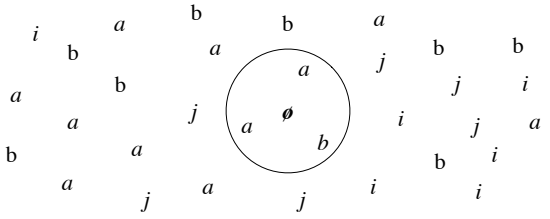

Figure 3: An example of k-nearest neighbors algorithm. An unknown input character $\boldsymbol{\varnothing}$ is decided by comparing it with the $k = 3$ nearest neighbors. The three nearest neighbors to $\boldsymbol{\varnothing}$ are a, a, and b, so in this example $\boldsymbol{\varnothing} = a$.

reduction algorithm was first applied before the k-NN classifier, as described in the next chapter.

3.1.1 Principal component analysis

A dimension reduction tool is preferable before performing a k-Nearest Neighbor Classifier, as discussed in the previous subsection. In this project, the Principal Component Analysis (PCA) procedure was used for reducing the number of correlated dimensions down to a set of the linearly uncorrelated principal component [**?**]. The principal Component Analysis will use too much space to derive in this report. In short, it decomposes the original n-dimensional space into m orthogonal dimensions by finding the orthogonal dimensions with the highest variance and computing the variance vector in that dimensions direction. The first dimension will have the highest variance; the second will have the second largest and so on. Numerical estimations show that the optimal condition for the k-Nearest Neighbor Classifier is when PCA is used to reduce the dimension down to $m = 0$.

The authors spent a substantial amount of time to make an implementation using t-distributed stochastic neighbor embedding [**?**] for easier separation of characters. However, this was not possible as the t-distributed stochastic neighbor embedding algorithm can only visualize trained data, and not transform a test set. If a t-distributed stochastic neighbor embedding transform is deactivated, then we believe that the combination of Principal Component analysis, t-distributed stochastic neighbor embedding, and k-nearest neighbors algorithm would be interesting to explore.

3.1.2 Performance of k-nearest neighbors algorithm

To find the best performance of k-NN with PCA, it is useful to implement a search with a nested loop, with a focus on the number k of neighbors and amount of dimension m the data was reduced to. The best performance without implementation was when $k = 0$ and, as mentioned above, $m = 0$. The best

Table 1: Parameters for pipeline consisting of HOG and LSVC

Parameter name	Value
HOG orientations	6
HOG Pixels per cell	$(4, 4)$
HOG cells per block	$(2, 2)$
LSVC C	1.1

Figure 4: Examples of character that k-NN successfully recognized, picket at random.

performance was calculated using Equation 1 and resulted in an error of

$$E_{k\text{NN}} = 100\ \% \tag{2}$$

which was higher than the allowed error of this project. We, therefore, continued with an approach based on Linear Support Vector Classifiers, as described in the next section.

3.2 Linear Support Vector Classifier

Support Vector Machines (SVM) is a tool often applied to classification, clustering and regression [**?**] as well as data mining [**?**]. A Linear Support Vector Classifier (LSVC) implemented is a supervised learning model for binary classification [**?**]. A Multiclass LSVC was used in this example, which means that the 26 classes are reduced to a set of binary classifications.

Each binary LSVC classification maps the training data's separate categories into points in space, and linearly divide them by an apparent gap. The test data is assigned to the same space and classified based on what side of the division line the data appears. Support Vector Classifiers can perform classify non-linear classification by using kernel tricks, i.e. transforming the data. However, kernel tricks were not applied in this project. The best parameters for this project are listed in Table 1

4 Character Detection

Character detection done by our LSVC is shown in Figure 6 and Figure 7. The character detection system performs really well on the test image, but not so great on the larger one. There might be several reasons to this. The authors

Figure 5: Examples of character that k-NN did not successfully recognized, picket at random.

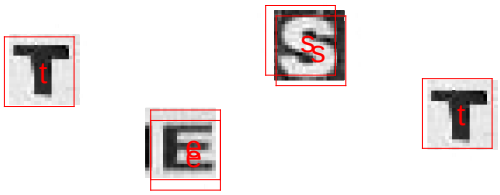

Figure 6: Detection on test image.

argue that the cleanliness of Figure 6 helps the system to only classify pictures of characters as actual characters. However, confusion arise when the sliding window is focused on more than one character at a time. Such an outcrop of two characters is completely new to the classifier, and the outcrop is misclassified.

The character detection system can be improved by adding additional image processing before character classification. The additional processing would help misclassifying white images as the character 'i'. The authors would also look into adding white images with class 'background' in the dataset. Our classifier would then train to distinguish between characters and background.

5 Conclusion

It's the authors opinion that the weakest part of the OCR system is the character detection. Some letters are recognized poorly due to that fact that our training data consists of pictures with white letters on black background and vice versa. This might have led our classifier to classify purely white images as a character, usually the character 'i'.

The strongest part of our OCR system is the character classification. The

Figure 7: Detection on a larger image.

Table 2: Libraries used.

Library	Link
`time`	`https://docs.python.org/3/library/time.html`
`matplotlib`	`https://matplotlib.org/`
`numpy`	`http://www.numpy.org/`
`scipy`	`https://www.scipy.org/`
`skimage`	`http://scikit-image.org/`
`sklearn`	`http://scikit-learn.org/`
`string`	`https://docs.python.org/3/library/string.html`
`os`	`https://docs.python.org/3/library/os.html`
`PIL`	`http://www.pythonware.com/products/pil/`

SVM classifier achieves an error rate of 0.13062, which is quite sufficient for the given task.

Due to lack of time, the authors didn't have time to improve the system further. The authors have gained a lot of experience with relevant machine learning libraries such as scikit-learn for python and the machine learning toolbox for matlab.

A Libraries used

A complete list of all libraries used for the python implementation can be found in Table 2.