TDT4173 - Assignment 5

Eirik Tvedt Jensen, Markus Andersson, Ole Steinar L. Skrede

1 Overview

When running the code the path/name of the image need to be written to the image field. The size of the image, width and height, also need to be changed to fit the chosen image. Then run the program and it will train on the example data and slide through the chosen image and draw blue squares around recognized letters in the image.

The Scikit-learn library has been used for this assignment. To run our code you also need numpy and scipy.

2 Pre-processing techniques

Noise removal by using the Scipy ndimage package.
Replaces each pixel value by the median of neighboring pixels.

Standardization of features by using the scikit-learn preprocessing package.
Removes mean value by subtracting it from each feature. Variance scaling by dividing features by the standard deviation. Features with a large variance might dominate the objective function and make the estimator unable to learn from other features.

3 Models

Decision tree:
Decision tree model is a non-supervised learning method for classification. It creates simple rules/questions inferred from the training data about the target for traversing down the tree. Reaching a leaf node will give a prediction of the value/classification.

Advantages of decision trees are that they are relatively simple to understand. It does not require a lot of data to create and the cost of using the tree is low, making the runtime fast. It also handles numerical and categorical data, and is able to handle multi-output problems, which is not relevant for this assignment. White box model, which means the logic behind the decisions are simple to follow. Decision trees also performs well, even if some assumptions when generating the tree are somewhat violated from the true model of the data.

Disadvantages of decision trees are that decision tree learners may create overly complex trees, meaning it overfits the data. To combat this we used a max depth of the tree to make it simpler. Decision trees may give unstable outcomes from small variations and/or noise in the data, and could generate different trees than what is optimal. The problem of learning an

optimal decision tree is NP-complete, and therefore it relies on the heuristics in the algorithms that creates the tree and cannot guarantee returning the globally optimal decision tree. The trees may sometimes become biased if there are classes that dominate, and requires data to be balanced before fitting.

The use of decision tree classifier was interesting because of the fast runtime, and because it was easy to understand as the first classifier to try out. But with low precision and poor results it was not the best choice as a classifier.

Support vector:
Support vector machines constructs a hyperplane in the dimension space which may be used for classification. Finding the largest distance to the nearest training-data point of class, lowers the generalization error of the classifier.

Advantages of the support vector classifiers are being effective in high dimensional spaces. They are still effective in cases with number of dimensions greater than samples and its memory usage is efficient.

Disadvantages are if the number of features is greater than the number of samples, the performance might be poor. And probability estimates is not directly derived and has to be calculated using expensive calculations such as five-fold cross-validation.

When using support vector classification the results were greatly improved compared to decision trees, and the precision of the results was high for most characters. But the runtime was also increased significantly, and the probability calculation later in the assignment increased the runtime even more.

4 Evaluation

Half of the dataset was used for training, and half of the set was used for testing. We used the built-in functionality to test our models, and then the built-in functionality to print statistics from the testing. The statistics included measures for precision and recall for each label. Below we show the results for each model. The labels are the leftmost column (a=0, b=1, …).

We clearly see that SVM outperforms the decision tree in both precision and recall measurements across all labels. The decision tree performed generally poor, only performing somewhat decent on a couple of the labels. However, the decision tree model is much faster to both train and predict. We tried different max depths for the decision tree model, with a max-depth of 3 it performed terrible. With this shallow depth the system was only able to recognize a couple of the characters at all, and the ones it recognized had precision scores in the range [0.11, 0.33]. Since $2*2*2 = 8 < 26$, it was no surprise that there were a lot of labels the system did not learn.

Decision tree, max depth = 20

	precision	recall	f1-score	support
0	0.54	0.49	0.51	358
1	0.12	0.13	0.13	76
2	0.40	0.45	0.43	139
3	0.21	0.23	0.22	119
4	0.56	0.52	0.54	336
5	0.19	0.12	0.15	58
6	0.16	0.13	0.14	90
7	0.25	0.36	0.29	122
8	0.44	0.46	0.45	214
9	0.07	0.13	0.09	55
10	0.33	0.24	0.28	63
11	0.43	0.39	0.41	137
12	0.27	0.21	0.24	96
13	0.48	0.47	0.48	249
14	0.49	0.48	0.48	265
15	0.24	0.24	0.24	98
16	0.04	0.05	0.04	44
17	0.35	0.45	0.39	258
18	0.54	0.44	0.49	233
19	0.48	0.48	0.48	211
20	0.17	0.19	0.18	67
21	0.37	0.32	0.35	59
22	0.25	0.23	0.24	52
23	0.15	0.14	0.15	57
24	0.26	0.18	0.21	55
25	0.12	0.11	0.12	45
avg / total	0.39	0.38	0.39	3556

Support Vector Machine

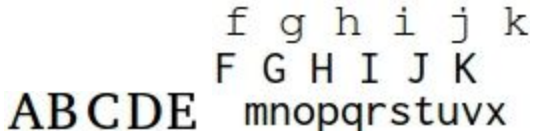
	precision	recall	f1-score	support
0	0.67	0.92	0.78	358
1	0.83	0.32	0.46	76
2	0.91	0.82	0.86	139
3	0.79	0.55	0.65	119
4	0.60	0.90	0.72	336
5	1.00	0.31	0.47	58
6	0.97	0.37	0.53	90
7	0.87	0.74	0.80	122
8	0.72	0.88	0.79	214
9	0.73	0.20	0.31	55
10	1.00	0.46	0.63	63
11	0.97	0.74	0.84	137
12	0.88	0.71	0.79	96
13	0.86	0.85	0.85	249
14	0.65	0.92	0.76	265
15	0.93	0.72	0.82	98
16	0.67	0.05	0.09	44
17	0.64	0.88	0.74	258
18	0.91	0.87	0.89	233
19	0.79	0.82	0.80	211
20	0.96	0.40	0.57	67
21	0.98	0.80	0.88	59
22	1.00	0.52	0.68	52
23	0.90	0.49	0.64	57
24	0.97	0.62	0.76	55
25	0.94	0.33	0.49	45
avg / total	0.79	0.76	0.74	3556

Here is the result of predictions on five random cases in our test set:

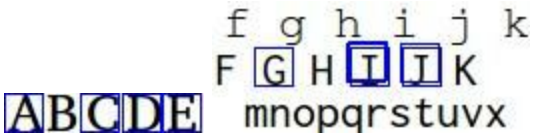
Target	**Prediction**
17 (R)	17 (R)
0 (A)	0 (A)
13 (N)	13 (N)
21 (V)	21 (V)
13 (N)	4 (E)

5 Sliding window test

Below are the images we tested our system on. We used the support vector machine model here, because it had the best precision and recall, and also the decision tree model does not give us probabilities.
This is some of the images we did the testing on.



And below are the same images marked with boxes where our system found characters.



In the “ABCDE” image, our detector performed pretty good, and was able to correctly detect and recognize 4 out of 5 characters. This should be an easy image for the detector, which explains the good result. Out of the 5 characters, our system scored lowest on recall (0.32) for the character ‘B’, while consistently scoring better on the other 4 characters, so it is not surprising that our system was not able to recognize the ‘B’. For this image our threshold was 0.5.

In the images to the right, we had to lower our threshold to (0.4) for the system to recognize more than the ‘G’. Because of the serifs on the ‘I’ and ‘J’, the system believes that these are ‘T’’s. But the ‘G’ is correctly recognized. We had expected our system to recognize a bit more in this image. The reason that the system did not recognize any of the “mnopqrstuvx” characters might be that they are placed too close to each other, so the sliding window always gets part of other characters within it’s frame.

It seems that our system has trouble with detecting lower case letters, but it performs decent on capital case letters.

6 Weak and strong components of our OCR

The noise removal did minimal for the performance and increased the precision with about 1% point. It would probably be more effective if the data included handwritten examples and not only computer generated, as that would probably increase the risk of unrecognizable/unusable letters. On the other hand the scaling/standardizing of the data significantly increased the precision with the support vector classifier.

Character detection was overall weak in our system on self made images. It does well in testing, but our pictures differ too much from the provided examples for it to recognize the letters, even with the threshold at 50%.

7 What went good/bad

Although we got a some good precision and recall on most of the labels, we feel that our system was a bit slow. We could have used more of the test set to train our system. Instead of a 50/50 split, we could have used 75% of the cases for training. Also we could have tried to scale all pixel values to 0 or 1, which might make the problem easier for our system. Another idea is to then count the number of white and black pixels in each image, and invert the colors if we suspect that the background is black, with the hope that all cases would end up with a white background and black characters.

8 References

Standardization:
http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing

Noise removal:
https://en.wikipedia.org/wiki/Median_filter

SVM:
http://scikit-learn.org/stable/modules/svm.html

Decision tree:
http://scikit-learn.org/stable/modules/tree.html