

Handwriting Recognition with Transductive Confidence Machines

Y.L.F.H. van Pinxteren

23rd June 2009

Bachelor's Thesis

Author:

Youri van Pinxteren

(0537403)

yourivanpinxteren@student.ru.nl

Department of Artificial Intelligence

Radboud Universiteit Nijmegen

Supervisors:

Dr. I.G. Sprinkhuizen-Kuyper

Dr. L.G. Vuurpijl

Handwriting Recognition with Transductive Confidence Machines

Youri van Pinxteren * Ida Sprinkhuizen-Kuyper * † Louis Vuurpijl * †

23rd June 2009

Abstract

The recently introduced transductive confidence machines (TCMs) framework allows to extend classifiers such that their performance can be set by the user prior to classification. In this paper we apply the TCM framework, with a plugged in k-nearest neighbor classifier, to the domain of (on-line) handwriting recognition. First, we modify the original TCM algorithm to make it much more efficient. Then, we use this modified algorithm to classify the NicIcon database of iconic gestures. Results show that the modified TCM algorithm is a promising way to classify handwriting.

1 Introduction

Handwriting recognition is the ability of a computer to read human handwriting. It therefore must be able to receive and interpret handwritten input. This input can be obtained by the computer in two different ways: on-line and off-line. Off-line handwriting data is acquired by scanning a handwritten document. Off-line data is a bitmap which consists of pixels with a grey value or color [2]. On-line handwriting means that the computer recognizes the writing while the user writes. The user must do this on an electric tablet or digitizer that can capture the writing as it is written.

Both methods have their advantages and disadvantages, but in comparison to off-line handwriting, on-line handwriting has many advantages because it can save a lot of information about the writing. In general, velocity and acceleration of the pen tip as a function of time is saved. From that it is possible to know the order of writing (which is useful for segmentation) [6], the pen pressure and the location of the pen when it is not on the paper. This enables classifiers to use a lot of features. Therefore, the results on on-line handwriting recognition are, in general, better than the results on off-line handwriting recognition.

*Department of Artificial Intelligence, Radboud University Nijmegen

†Donders Institute for Brain, Cognition and Behavior; Centre for Cognition

The main disadvantage of on-line handwriting recognition is that the writer is required to use special equipment [10], as until some years ago on-line equipment was not as comfortable and natural to use as pen and paper [9]. This changed a lot, since the development of pen-computing applications has been growing steadily the last years, due to, among other factors, the introduction of devices such as personal digital assistants (PDAs) or Tablet PCs [8]. The main characteristic of such devices is that they use the stylus as input tool, being a natural substitute for keyboards and mouse. Therefore, on-line handwriting recognition will become of more interest in the following years.

Till now, pen-computing applications are mainly used in situations where classification errors do not have far-reaching consequences. In many real-world situations classification errors actually do have far-reaching consequence, while such pen-computing applications could be very useful. The reason why they are of limited use in such situations, is that the classifiers of the handwriting recognition in pen-computing applications are not aware of their limits in knowledge. In other words, they just classify all new instances, so that we can never be sure if a particular classification is correct. They do not distinguish between certain classifications and guesses.

As said, in many real-world situations, like the military or crisis-management, errors in classification (of handwriting) could have far-reaching consequences. Therefore, if we want to use handwriting recognition in such situations, it is necessary to know how certain a particular classification is. This can be done by reliable classification approaches, like Transductive Confidence Machines (TCMs) [7] and Receiver Operating Characteristics (ROCs) [12]. Such approaches can guarantee a desired classification performance. The user can set the performance prior to classification. The key idea is that these approaches identify the instances where there is uncertainty in the true label and assign multiple or no labels to it. Virtually any classifier, like k-Nearest Neighbor or Support Vector Machines (SVM), can be plugged into these approaches [13, 11].

TCMs are recently introduced and there is no research done in combination with handwriting recognition, while ROCs already turned out to be a good method to classify handwriting. Further, TCMs have some advantages over ROCs: the scores are statistically underpinned, any preset performance can be guaranteed, they can be applied without modification to multi-classification problems and they are better on small datasets [11].

In [1] Kaptein tried other reliable classification approaches on texts, but these classifiers only say if a classification is certain or uncertain. They do not say how certain or uncertain a classification is. Also, these texts were not handwritten. In [13] Vanderlooy et al. applied six TCM implementations on ten well-known benchmark databases. These were all databases with a binary label space (only two possible labels), which is obviously not the case in handwriting databases.

Because of this multi-class label space and the wide variability of handwritings, handwriting recognition is considered a difficult task. Therefore, a classifier usually needs a lot of training data to get good results. This also counts for TCMs. It is reasonable that this could

be a problem, because the TCM algorithm already needs a lot of time to compute the results.

In this paper, we want to recognize (on-line) handwriting in a reliable way. We do this by applying the TCM algorithm on the NicIcon database of handwritten icons [5]. Through this we do not only investigate if TCMs work within a new domain (handwriting recognition), but also if they work for a typically difficult and multi-class problem. Further we test if TCMs work on a huge dataset, but we can already say that this is not the case. Therefore, it is necessary to make the TCM algorithm much more efficient. We do this first, so we can apply the modified algorithm on the NicIcon database.

The remainder of the paper is organized as follows. Section 2 describes the methods we used. We first explain the TCM framework in Subsection 2.1, also in combination with the k-NN classifier. In Subsection 2.2 we give some detail about the NicIcon database. In Subsection 2.3 we explain some modifications we had to apply to the TCM framework, in order to make it much more efficient. Section 3 provides the results of our experiments. Finally, Section 4 concludes that the TCM-kNN approach is a promising way to classify handwriting datasets.

2 Methods

2.1 Transductive Confidence Machines

Most classifiers assign a single label to an instance, but TCMs are allowed to assign multiple (or no) labels to each instance. Therefore, every instance has a so called prediction set. If there is uncertainty in the true label of the instance, a prediction set could contain multiple labels. To construct such a prediction set, TCMs operate in a transductive manner [13]. This means that TCMs reason from observed, specific (training) instances to specific (test) instances. Every possible label $y \in \mathcal{Y}$ is tried as a label for the unlabeled (test) instance x_{n+1} . In each try the example $z_{n+1} = (x_{n+1}, y)$ is formed and added to the training data $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$:

$$S^+ = \{(x_1, y_1), \dots, (x_n, y_n), (x_{n+1}, y)\} = \{z_1, \dots, z_{n+1}\} . \quad (1)$$

For every example in the extended set z_1, \dots, z_{n+1} , a nonconformity measure is calculated. This measure tells us how nonconforming an example is in comparison to all other examples, so a relative high nonconformity score means that an example is probably labeled with the wrong label.

Virtually any classifier can be plugged into the TCM framework, because for every classifier a non-conformity measure can be calculated [13]. Therefore, nonconformity scores can be calculated in different ways. We used the classifier k-Nearest Neighbor to calculate the nonconformity scores. In [7], Saunders et al. already formulated how a nonconformity score can be calculated for this classifier. This can be done as follows. Given example $z_i = (x_i, y_i)$, define an ascending ordered sequence $D_i^{y_i}$ with distances from x_i to its k nearest ‘positive’

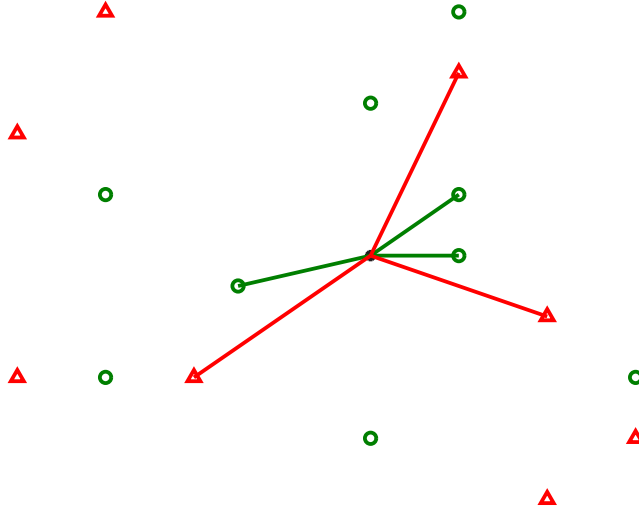


Figure 1: Calculation of the nonconformity score of the (black) star instance. The (green) circles are its nearest neighbors of class y_i , while the (red) triangles are its nearest neighbors of other classes. Here $k = 3$, so the three smallest distances are being summed. In this example, the non-conformity score of the (black) star instance is pretty low, so that instance is quite conforming.

neighbors with label y_i . Similarly, let $D_i^{-y_i}$ contain ordered distances from instance x_i to its k nearest ‘negative’ neighbors with a label different from y_i . Then, the nonconformity score is defined as:

$$\alpha_i = \frac{\sum_{j=1}^k D_{ij}^{y_i}}{\sum_{j=1}^k D_{ij}^{-y_i}}, \quad (2)$$

with j as the j -th element in the distance sequence. This means that an example is nonconforming when it is far from its nearest neighbors with the same label and close to its nearest neighbors with a different label. See Figure 1 for clarification. We used different numbers of nearest neighbors (k) to find the best results for the used data set.

To know how nonconforming an example is in the extended set, the nonconformity score must be compared to all other α_i in the extended set S^+ . The result of that is the p value of label y assigned to unlabeled instance x_{n+1} and is defined as follows:

$$p_y = \frac{|\{i = 1, \dots, n + 1 : \alpha_i \geq \alpha_{n+1}\}|}{n + 1}. \quad (3)$$

Simply said, it calculates the fraction of examples that are more nonconforming than that particular example. If the p value is low this means that the example is very nonconforming, while a high p value means that the example is very conforming. So the p value indicates how

likely it is that the tried label is actually the true label, it is the probability that the tried label is correct.

A TCM outputs the set of labels with p values above a predefined significance level ϵ . It checks for every unlabeled instance x_{n+1} , which p values are bigger than the predefined significance level ϵ . The associated labels are then added to the prediction set:

$$\Gamma^\epsilon(z_1, \dots, z_n, x_{n+1}) = \{y \in \mathcal{Y} \mid p_y > \epsilon\} \quad , \quad (4)$$

with $\epsilon \in [0, 1]$. If $\epsilon = 0$, all prediction sets will contain all possible labels and the number of errors (the correct prediction is not in the prediction set) will then be zero. If $\epsilon = 1$, almost all prediction sets will be empty and almost every prediction will then be an error.

Further, the number of errors Err_n^ϵ will always be equal (or less) than $1 - \epsilon$. This is called the calibration property:

$$\limsup_{n \rightarrow \infty} \frac{Err_n^\epsilon}{n} = \epsilon \quad . \quad (5)$$

In the on-line learning setting, when the true label is provided after prediction for feedback, TCMs have been proven to satisfy this property [14, p. 20-22 & p. 193]. We, however, used the off-line learning setting because this feedback is very expensive, because the classifier is in the on-line learning setting retrained after each prediction since new information is available. In the off-line learning setting, the classifier is learned on training data and subsequently used to classify instances one by one. In [13], Vanderlooy et al. have found strong empirical evidence that the calibration property also holds in the off-line learning setting. Because of this property, the user can set the error rate prior to classification. Because we use a non-randomized TCM for our experiments, the equality sign in (5) must be replaced by the \leq sign. For clarification, see [13, p. 4].

To give an idea of the complexity of TCMs, consider the pseudo code of algorithm 1. We can see that we first compute all distances needed. If we have a huge data set it could be a problem to store all these distances in the memory of the computer, while this is actually needed for the remainder of the algorithm. With the distances between the training instances, we can compute all ‘basic’ nonconformity scores of the training instances. Those are thus the nonconformity scores, based on (only) the training data S . After this, we are going to try every possible label on all test instances. Every time, we create the extended sequence S^+ , by adding the tried test instance to the training data S . Then we can recalculate the nonconformity scores of all training instances, so these are then based on S^+ . A nonconformity score changes when the distance between the training instance and the added test instance is smaller than the k -th smallest neighbor of the training instance (when the tried label has the same label as the training instance we look at the ‘positive’ neighbors of the training instance, otherwise to the ‘negative’ neighbors). After all nonconformity scores of the training instances are recalculated, we compute the nonconformity score of the tried test instance. That is why we needed the distances between the test and training instances. When we know that

nonconformity score, we can compute the p value of the tried label by comparing it to the recalculated nonconformity scores of the training instances. We can see here why we need to recalculate all nonconformity scores, every time S^+ is formed. If we have computed all p values, the prediction set for each test instance can be created by selecting all labels with a p value higher than the predefined significance level.

```

1 Calculate distances between training instances;
2 Calculate ‘basic’ nonconformity scores of training instances;
3 Calculate distances between test and training instances;
4 foreach Test instance do
5   | foreach Possible label do
6   |   | Add tried test instance to all training instances;
7   |   | foreach Training instance do
8   |   |   | Recalculate nonconformity score;
9   |   |   | end
10  |   | Calculate nonconformity score of tried test instance;
11  |   | Calculate  $p$  value of tried test instance;
12  |   | end
13 end
14 Create prediction sets;

```

Algorithm 1: The TCM algorithm

It looks like the complexity of algorithm 1 is not that big, but it is. This is due to the complexity of the inner loop. To recalculate the nonconformity scores of all training instances, we have to search for the distance between the particular test and training instances. This has a complexity of $O(\text{nr. of training instances})$. The total algorithm then has a complexity of $O(\text{nr. of test instances} \times \text{nr. of labels} \times \text{nr. of training instances} \times \text{nr. of training instances})$, not counting the calculation of all distances. This is very complex, especially if we have a huge data set (with a lot of test and training instances). To make the algorithm more efficient, naturally we want to decrease the complexity of the inner loop. We will see in Subsection 2.3 how we did this.

2.2 NicIcon

We applied the TCM framework on the NicIcon database [5]. This is a collection of hand-written sketches containing iconic gestures. These data were recently collected within our department of Artificial Intelligence. The database consists of a set of 14 icons important in the domain of crisis management. They were designed in a way that they are easy to learn by the users and distinguishable for the computer.

There were 32 volunteers that participated in the experiment. They all had to draw 770 iconic gestures. In spite of some skipped gestures, this resulted in a huge data set with a total















Description	Icon	n	Description	Icon	n
Accident		1736	Gas		1745
Bomb		1750	Injury		1775
Car		1720	Paramedics		1780
Casualty		1750	Person		1755
Electricity		1735	Police		1740
Fire		1740	Roadblock		1750
Fire brigade		1725	Flood		1740

Figure 2: Iconic gestures with their distribution

of 24441 usable iconic gestures. Figure 2 shows the different icons with their distribution.

There were made two sets of all these icons. First, a writer dependent set (WD), where the total set was randomly divided into a training set of 60% and a test set of 40%. Secondly, a writer independent (WI) set, where the train set contained all icons of 60% of the writers and the test set all icons of the other 40% of the writers. It can be expected that the WI set is more difficult to classify.

Niels et al. [5] used three different classifiers to distinguish between the different icons, namely a multilayered perceptron, a linear multiclass SVM and Dynamic Time Warping (DTW) [4]. The DTW classifier gave the best results of these three and we used it in our experiment. The DTW classifier calculates the similarity between two online trajectories of coordinates. This is done by point-to-point comparison of two trajectories. A ‘matching path’ is then created, that represents the combinations of points on the two curves that are matched together. The DTW score is then the summed and averaged Euclidean distance between all couples of matching points. These scores are used in our experiment as distance measures in the k-Nearest Neighbor classifier.

The DTW classifier had a performance of 98.06% on the WD set and a performance of 94.70% on the WI set. This is already a pretty high performance for both sets, but we will see in Section 3 that these performances can be improved considerably using our new technique.

2.3 Modifications

We already said that TCMs try every possible label for an unlabeled instance. It could be expected that this is very expensive if there are many possible labels. This is the case in handwriting recognition, where a label stands for a letter or, in this case, an icon. In the NicIcon database there are 14 possible labels (see Section 2.2). This is not exceptional in handwriting, but it is also a very large database (24441 instances, 60% training and 40% test instances). This is, for a TCM, very much. In Section 2.1, we said that storing all distances for the algorithm in memory, may be a problem when you have a huge data set. In

our case, we already needed approximately 3 GB memory to store the two distance matrices ($14667 \times 14667 \times 8 + 14667 \times 9774 \times 8$ bytes). This is without the id-strings of the training instances for the distances between test and training instances. These id-strings are needed to search for the correct training instance, so the total amount of memory needed is even much larger: 3 GB + $14667 \times 9774 \times 60$ bytes (header) + $14667 \times 9774 \times 115$ bytes (data) \approx 28 GB. We obviously did not have a computer to our disposal with this kind of memory available, but even if we had, the algorithm will still be taking way too much time to complete. This is why we tried to modify the algorithm.

To calculate the p value of an unlabeled instance for a tried label, we have to calculate the non-conformity score of that instance and all the training instances. This is because we have to compare these non-conformity scores with each other. To calculate a non-conformity score, we need the distances between the particular instance and its k nearest neighbors of the correct (or assigned, in case of a test instance) class and of the set of incorrect (or unassigned) classes (see Eq. 2).

First, we calculate all ‘basic’ non-conformity scores of the training samples, so we only need the k smallest distances of the correct class and of an incorrect class between training samples. All other distances could be thrown away to save a lot of memory. After this, the non-conformity score of a training instance possibly needs to be recalculated, because of the addition of the tried test instance to the (then extended) sequence. So, to recalculate the non-conformity score of a training instance, we need the distance between that training instance and the added test instance. If that distance is very small, smaller than its largest (k -th) nearest neighbor, the non-conformity score of the training instance will change and needs to be recalculated. For clarification, see Figure 3.

All distances between test and training instances are actually needed to calculate all p values precisely, but it takes a lot memory to store them all and it also takes a lot of time to search for the correct distance so many times (nr. of labels \times nr. of test instances \times nr. of training instances) or, to sort all the distances in the right way. This is because when the distances between test and training instances were provided, they were also (like the distances between the training instances) sorted by distance, while they had to be sorted by training instance to easily search for the correct distance.

Only a small percentage of the non-conformity scores of the training instances needs to be recalculated, so to save a lot of memory, we do not use (and save) all the distances between the test and training instances but only the (say m , with $m \geq k$) smallest number of distances for every class. These could be easily picked out, because the provided distances were already sorted (see above). If the distance of the k -th nearest neighbor (of the correct class and of the set of incorrect classes) is already smaller than the m -th smallest distance between the test and training instances, it is very unlikely that the non-conformity score of the training instance would change (and if it would, it would be very minimal), so we do not change it. If the distance of the k -th smallest nearest neighbor (of the correct class or of the set of incorrect classes) is larger than the m -th smallest distance between the test and training

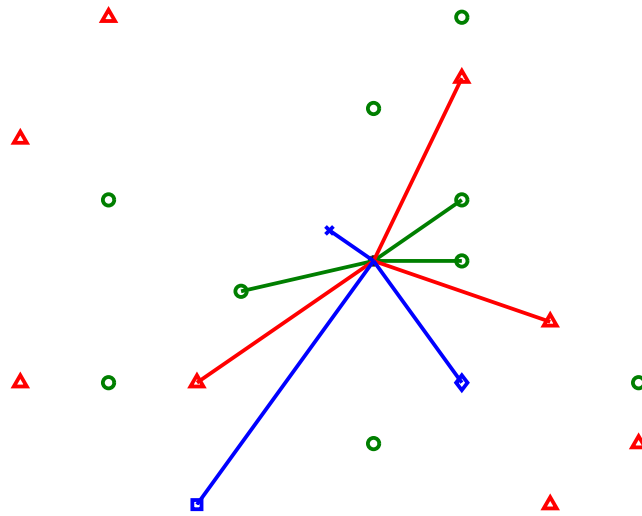


Figure 3: Recalculation of the nonconformity score of the (black) star training instance. The (green) circles are again its nearest neighbors of class y_i , while the (red) triangles are its nearest neighbors of other classes. Here $k = 3$, so the three smallest distances are being summed. The (blue) cross, diamond and square are test instances. In case of the (blue) cross test instance, the non-conformity score of the training instance has to be recalculated if the test instance has class y_i and if it has another class. In case of the (blue) diamond test instance, the non-conformity score of the training instance only has to be recalculated if the test instance does not have class y_i . In case of the (blue) square test instance, the non-conformity score of the training instance does not have to be recalculated, regardless of its class.

instances, the chance that the non-conformity score would change is higher. In this case we search for the particular training instance within the m smallest distances. If it is found, we know the exact distance between the test and training instance and can recalculate the non-conformity score of the training instance precisely. If it is not found, we do not know the exact distance and fill in the m -th smallest distance as an approximation. In that case, the non-conformity score will be slightly smaller (and the p value slightly larger) than the real score if the k -th nearest neighbor of the correct class is substituted. If the k -th nearest neighbor of all incorrect classes is substituted, the non-conformity score will be slightly larger (and the p value slightly smaller) than the real score. Later, we will see that this is quite a good approximation.

Finally, the non-conformity score of the unlabeled instance for a tried label has to be recalculated. This is very easy to calculate, because we only need the k nearest neighbors of the assigned label and of an unassigned label. We can calculate this score precisely when we have only m nearest neighbors per label. This also takes less time in comparison with the original algorithm, because we have to sort a much smaller number of distances.

Please consider algorithm 2, which summarizes the modifications. Some words are in (green) italic, which point out the differences between the original algorithm. First, all distances still have to be calculated, but only a small number of distances is stored in memory for further use. They are directly sorted in the way we want to, so that we can simply see what the ‘basic’ nonconformity scores are and which nonconformity scores we want to recalculate. Therefore, we do not have to go through all training instances, but only the ones that need recalculation of their nonconformity score. The recalculation itself is also a lot easier, because we search only in the m nearest neighbors for the correct distance and if it is not there we just use the m -th nearest neighbor. In that way, we only need to search in a small number of distances. Finally, the calculation of the nonconformity score of the tried test instance is also easier. This is because the needed distances are already sorted, so we do not need to go through all distances between test and training instances for this. Overall, the algorithm is similar, but uses much less distances and sorts them directly so that all calculations are a lot easier. Also, the inner loop does not have a complexity of $O(\text{nr. of training instances})$ anymore, but a complexity of $O(m)$. The algorithm also does not get into that loop as much as before.

```

1 Calculate distances between training instances;
2 Calculate 'basic' nonconformity scores of training instances;
3 Calculate distances between test and training instances;
4 foreach Test instance do
5   foreach Possible label do
6     Add tried test instance to all training instances;
7     foreach Training instance that needs recalculation do
8       Recalculate nonconformity score;
9     end
10    Calculate nonconformity score of tried test instance;
11    Calculate  $p$  value of tried test instance;
12  end
13 end
14 Create prediction sets;

```

Algorithm 2: The modified TCM algorithm

We estimated that 5.0% of the non-conformity scores (of training instances with the same class) have to be recalculated for every tried test instance with $k = 10$. This becomes less when k is smaller and more when k is bigger. This is not much, so the p values will certainly not change much in this way.

In the next section, we will see that we got our best results on the WI set with a high k . This means that there are more non-conformity scores that need to be recalculated. This was again, computationally too expensive. Therefore, we only used 1000 test instances (out of 9590). These had a recognition performance of 98.6% instead of the 94.7% for the whole set. Further, we used the whole WD set.

3 Experiments and Results

This Section provides the results of our experiments. We first compared our modified TCM algorithm with the original one (Subsection 3.1). In Subsection 3.2, we describe our results on the WD set and in Subsection 3.3 the results on the WI set. In Subsection 3.4, we describe our results on the error samples of the DTW-classifier in [5]. These are the test instances that were incorrectly classified by that classifier.

3.1 Modified TCM algorithm

In Subsection 2.3 we described why and how we changed the original TCM algorithm. Before we were going to use this modified TCM algorithm, we needed to test it in order to see if the modifications had any effect on the results.

We did this by using the well-known `pima` data set from the UCI benchmark repository [3].

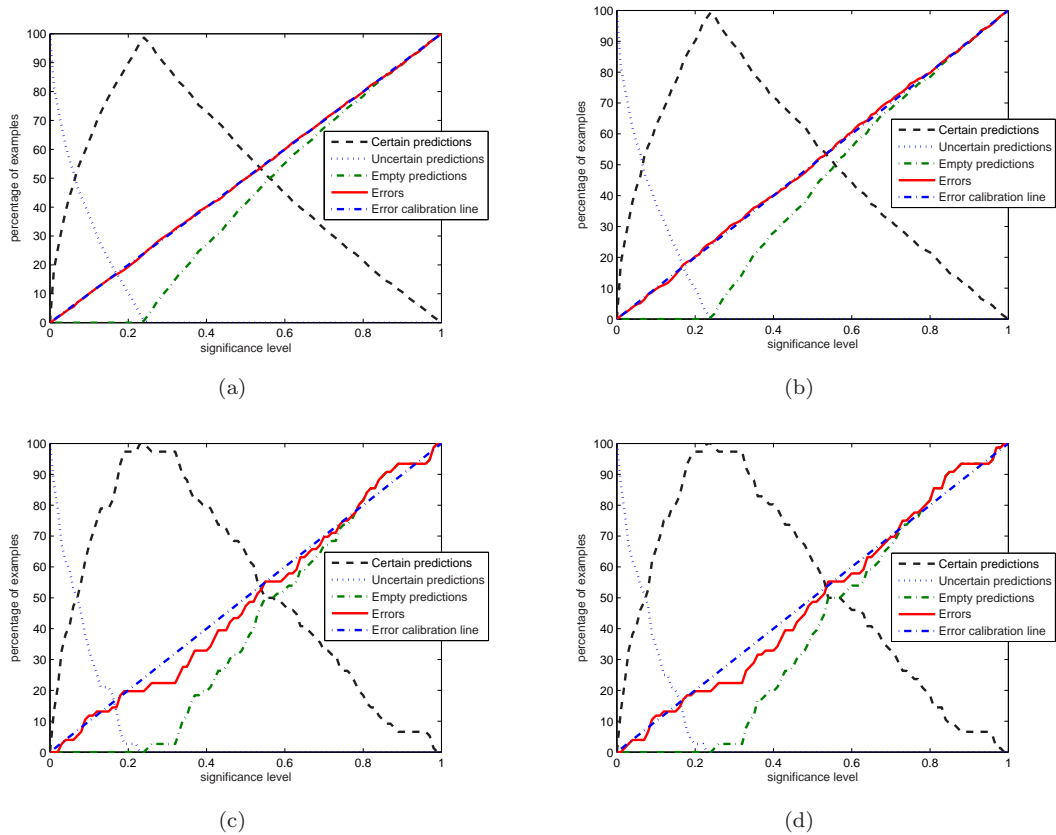


Figure 4: Results on the `pima` data set with $k = 10$: (a) Original TCM algorithm with 10-fold cross validation and 5 random permutations, (b) Original TCM algorithm with 10-fold cross validation without random permutations, (c) Original TCM algorithm without cross validation and (d) Modified TCM algorithm.

The results of the original TCM algorithm were already available in [13], but there they used a 10-fold cross validation for 5 random permutations of the training data. Because we used two predefined datasets (WD and WI) and wanted to compare our results with the results of the DTW-classifier in [5], we could not divide the data into arbitrary subsets and thus could not use the 10-fold cross validation. Therefore, we computed the results of the original TCM algorithm on the `pima` data set without cross validation. These can be seen in Figure 4.

Figure 4 is the standard way to represent the results of a TCM. We can see the number of certain (a prediction set of size = 1), uncertain (size > 1) and empty (size = 0) predictions. We can also see the number of errors (the correct prediction is not in the prediction set) and the error calibration line (number of errors must be under this line: calibration property of Equation 5). Note that all empty predictions are also errors and that certain or uncertain predictions are also errors if the correct prediction is not in the prediction set.

Because there is hardly a difference noticeable between Figure 4(a) and 4(b), the number of times cross validation is performed does not matter. The very small difference could be caused by the random creation of the train and test set. In Figure 4(c), we can see that the lines are not smooth anymore. The variance becomes bigger. This could be expected, because we only have one random train and test set in this case. Besides the bigger variance, it could be expected that the figure looks the same. The exact results are dependent of the random sets.

Therefore, we used exactly the same randomly picked train and test set for our modified TCM algorithm. For our modified TCM algorithm we need less distances between test and train instances (see Section 2.3). The minimal number of distances needed per class is the number of nearest neighbors. In our experiment we used 10 nearest neighbors and tried different numbers of distances the algorithm used. In Figure 4(d) we can see the results of our modified TCM algorithm, while it uses the minimal number of distances needed. It is clear that the results are barely different from the results in Figure 4(c), where all distances between test and train instances are used.

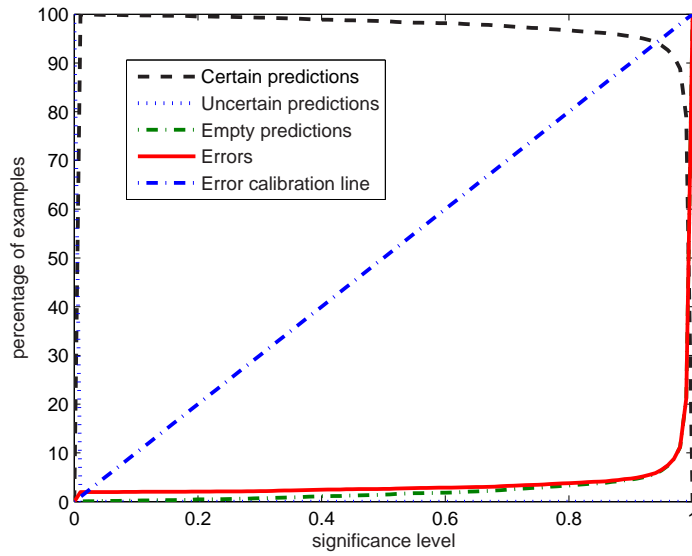
This indicates that our modified TCM algorithm could compute (approximately) the same results as the original TCM algorithm much faster. This modified algorithm will be used for the rest of our experiments. For the WD set we used 10 distances between test and train instances per class and for the WI set 50. This is because we tried k 's bigger than 10 for the WI set.

3.2 Writer Dependent Set

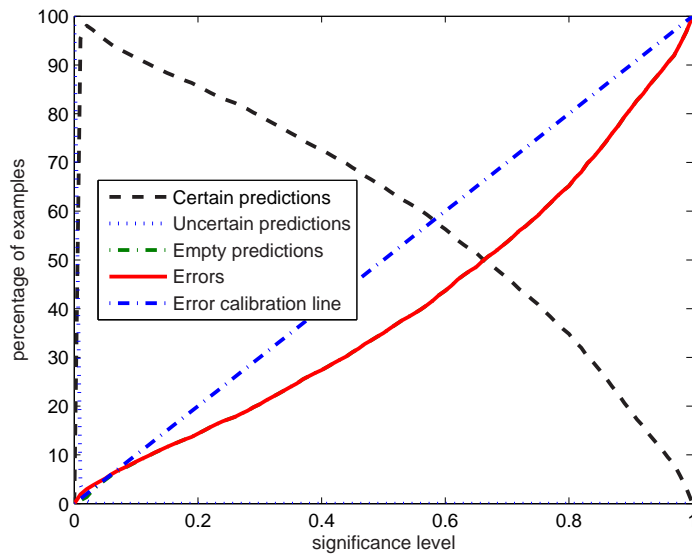
It is obvious that handwritings of various writers can be very different. This makes handwriting recognition often very difficult. In the writer dependent set, we do not have that problem. Every writer that is in the test set is also present in the training set. This makes it easier to recognize an icon. As said, the performance of the DTW classifier on this set was 98.06%.

It could be expected that the results on this set are highest, when using a small k . A larger k often reduces noise in the data, but because this is a relatively easy set to recognize (so there is probably not much noise in the data) it is not necessary to reduce the noise. A larger k would only make the boundaries between the classes less distinct, while it is likely that they are pretty clear. It was important in this case to have this kind of prediction, because it is very time-consuming to try a lot of different k 's with this huge data set. Because of this prediction, we began experimenting with small k 's. In Figure 5, we can see the results on the WD set for $k = 1$ and $k = 8$.

It is clear that the number of empty predictions is much larger with a larger k , so a larger k makes the boundaries between the classes indeed less distinct (less convincing p values). Further, it is striking to see the minimal number of uncertain predictions. Almost all predictions are certain or empty, even with a high significance level. This probably means that the icons are very easy to distinguish (this was also a goal of the creators of the icons,



(a)



(b) Note that the error line and the empty predictions line run almost identical

Figure 5: Results on the WD set: (a) $k = 1$, (b) $k = 8$.

see Section 2.2). In the case of an empty prediction set, the drawing was probably just not convincing enough.

Because the DTW classifier always gives a (certain) prediction and because a big part of the errors the TCM makes are empty predictions, we also looked at the results of the TCM when there are no empty prediction sets allowed. In that case, the TCM adds the label with the highest p value to the prediction set (even if it is smaller than or equal to the significance level), so it becomes a certain prediction. Now, a somewhat larger k is better, because it does not matter that this creates more empty prediction sets while it reduces the noise. The best result was a recognition of 98.62%, with $k = 4$. There were no uncertain predictions created, while this was actually possible. This is slightly better than the DTW-classifier, but relatively a big progress because there is not much to improve anymore.

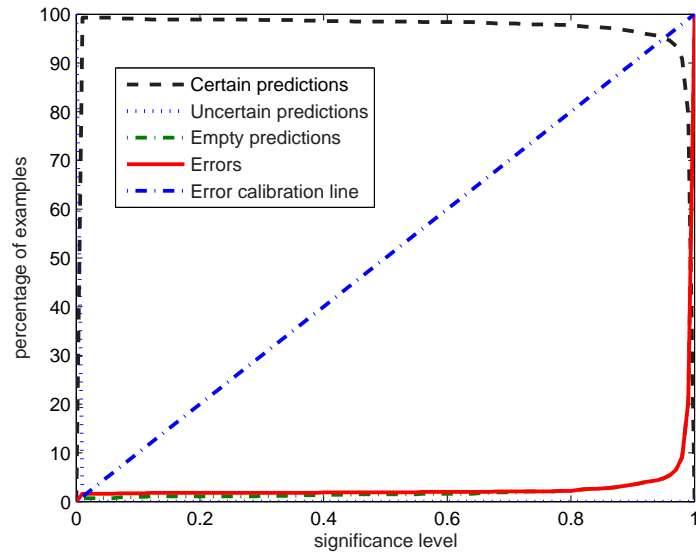
3.3 Writer Independent Set

In many situations, the (handwritten) text that has to be recognized is written by a writer that is not present in the data set. Therefore, we can not train the classifier on text, written by the same writer as the text to be recognized. This is the idea of the writer independent set. The training set only includes drawings of writers that are not in the test set. This is a more realistic and (normally) a more difficult situation.

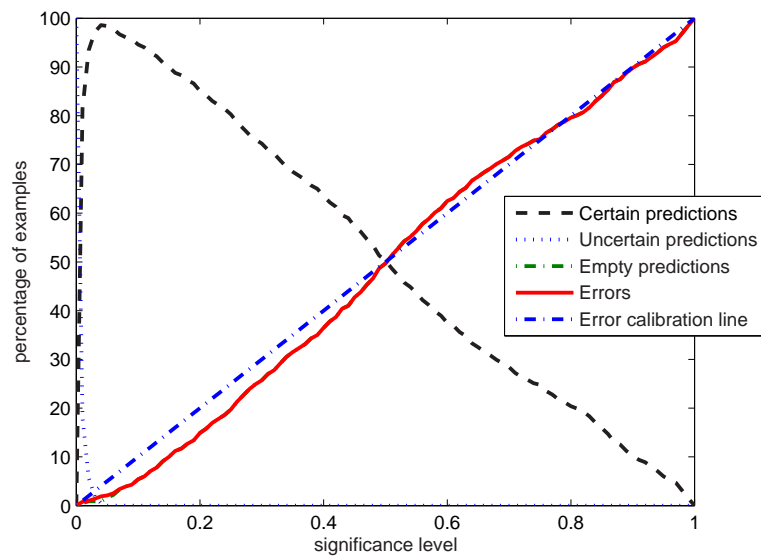
We can see this in the performance of the DTW classifier over the whole set: 94.7%. It could be expected that a higher k is needed to get the best results on this set. This increases the computation time, so we only used 1000 test samples out of this set to decrease the computation time again. Unfortunately the performance of the DTW classifier on these 1000 test samples was 98.6%. This is (again) already a very high performance, so maybe a small k still gives the best results.

We tried many different k 's. Figure 6 shows the results on the WI set for $k = 2$ and $k = 27$. It is clear that a very small k still gives the best results. With $k = 27$, the percentage of incorrect predictions at each significance level lies approximately on the error calibration line. This means that the TCM satisfies the calibration property. With smaller k 's, the percentage of incorrect predictions at each significance level always stays (far) below the error calibration line. This also satisfies the calibration property, because this is a non-randomized TCM, and is even a better result. If the percentage of incorrect predictions at each significance level lies below the error calibration line, it means that despite of the high significance level the TCM still makes a prediction. Otherwise it just refuses to make a prediction, while empty predictions count as errors. In this case, it just means that the data set is very easy to classify.

When empty prediction sets are not allowed, $k = 10$ gave the best result. This value of k is higher than with the WD set. The performance was 99.6%, so this is an improvement of 1.0% with regard to the DTW classifier. This is again a relatively big progress.



(a)



(b) Note that the error line and the empty predictions line run almost identical

Figure 6: Results on the WI set: (a) $k = 2$, (b) $k = 27$.

3.4 Error Samples

Finally, we looked at all the error samples of both sets. That are the test instances that were incorrectly classified by the DTW classifier. It is expected a higher k is needed to get the best results, because all test samples are very difficult to classify (or easy to classify incorrectly). This is also the reason why we only look at the case where empty prediction sets are not allowed. If these are allowed, almost every prediction will be empty and the TCM will not satisfy the calibration property. This could be expected, because the noise in the data is in this case very big and a lot of test samples will always be classified incorrectly. Even with a relatively low significance level, just because some drawings really look like the wrong icon.

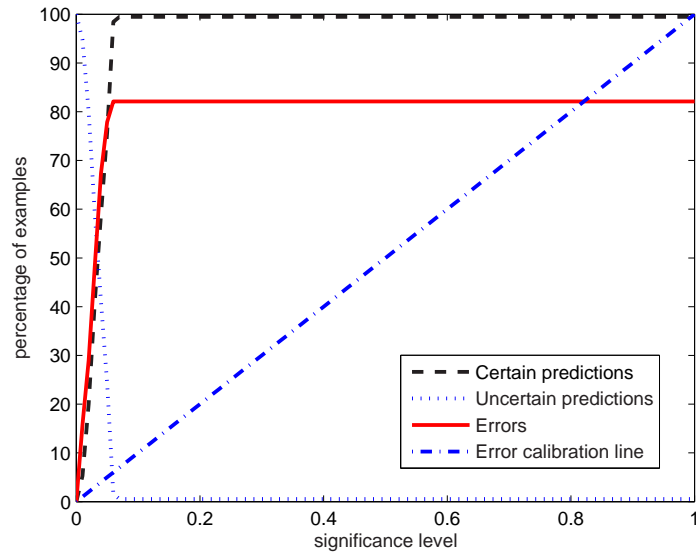
In the WD set, 190 test samples were incorrectly classified and in the WI set 508. These sets are much smaller than before, so we were able to try many different k 's. It is difficult to say precisely which k gave the best results. This is because uncertain prediction could be correct, while we still do not know exactly which class the test sample has. The only 'certain' improvement is therefore the number of certain predictions minus the number of certain errors.

In that case, 35 nearest neighbors give the best results on the WD set: 99.47% of the predictions is then certain, while there is 81.58% incorrectly predicted. This gives a certain improvement of 17.89%. On the WI set, 100 nearest neighbors gave the best results. This is, as expected, a higher number in comparison with the WD set. Here, 99.80% of the predictions is certain and 81.50% is incorrectly predicted. This results in an even higher improvement on the WI set: 18.30%. These are the results on high significance levels. In Figure 7, we can see that the results do not change anymore when the significance level is higher than (approximately) 0.1. When the significance level is lower, there are naturally some uncertain predictions (of which we do not know if they are correct).

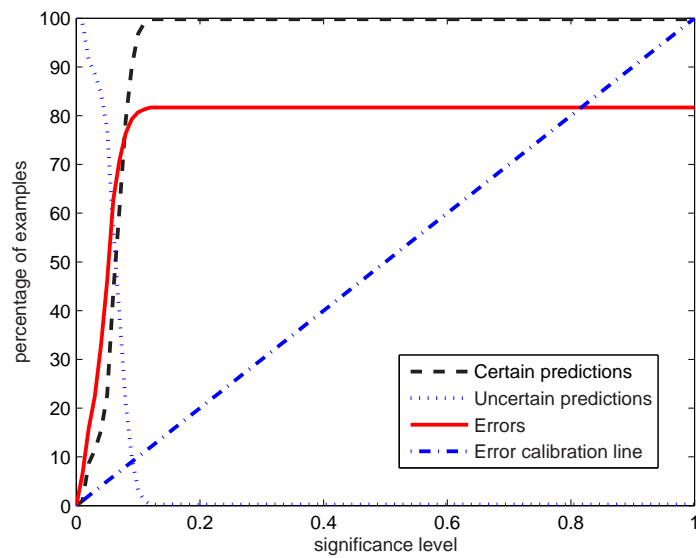
4 Conclusion and Discussion

In this paper, we focused on the applicability of transductive confidence machines (TCMs) in on-line handwriting recognition. TCMs allow to make predictions such that the error rate is controlled a priori by the user. This property is called the calibration property. It is proven that the calibration property holds in many different application domains in the on-line and off-line learning setting. Nevertheless, not much research is done in the domain of handwriting recognition, while the interest in on-line handwriting is growing because of the development of many pen-computing applications.

We applied TCMs in the domain of on-line handwriting recognition, which is typically a multi-class problem with a high variability of input. Because there is often a lot of data needed to train the classifier for such a problem, we modified the TCM algorithm to reduce the computation time. We plugged the k -nearest neighbor (k -NN) classifier into our modified TCM algorithm and used the dynamic time warping (DTW) algorithm to calculate the



(a)



(b)

Figure 7: Results on the error samples: (a) WD set, (b) WI set.

distances. This was then applied on the NicIcon database of iconic gestures, which contains 14 icons important in the domain of crisis management.

From the results of our experiments we may conclude that our modified TCM algorithm gives (approximately) the same results as the original TCM algorithm, only in a much more efficient way. Further we improved the (already high) performance of the DTW classifier on both the writer dependent (WD) and the writer independent (WI) set of the NicIcon database. We also looked at the test samples that were incorrectly classified by the same DTW classifier and managed to classify almost 20% certainly correct.

Since the results of our experiments show that TCMs are a promising way to classify handwriting, further research could focus on different data sets. Handwriting is often a difficult task, but this data set was rather easy to classify. It is interesting to see what effects a more difficult data set has on the results of the (modified) TCM. It could be expected that the number of nearest neighbors needed is a bit higher and therefore also the number of distances needed between test and training instances (m). Moreover, the number of uncertain predictions will probably a bit higher. This number was now very minimal.

It is also interesting to investigate if there are other classifiers (than k-NN) to plug into the TCM that work on handwriting recognition. Further, testing a TCM within a multiple classifier system would be interesting, since this method can identify empty and uncertain instances. Finally, it could be very useful to investigate how much distances (between test and train instances) are needed by the modified TCM algorithm to make it most efficient, while still obtaining good results.

Acknowledgment

We would like to thank Ralph Niels for providing all necessary data from the NicIcon database.

References

- [1] A. M. Kaptein. Meta-classifier approaches to reliable text classification. Master's thesis, Universiteit Maastricht, Maastricht, NL, August 2005.
- [2] Stefan Kennedie, Ralph Niels, and Louis Vuurpijl. Mapping online data on offline documents. BSc thesis, Radboud Universiteit Nijmegen, Nijmegen, NL, February, 2008.
- [3] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases, 1998.
- [4] Ralph Niels, Louis Vuurpijl, and Lambert Schomaker. Automatic allograph matching in forensic writer identification. In *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, volume 21, pages 61–82, 2007.

- [5] Ralph Niels, Don Willems, and Louis Vuurpijl. The NicIcon database of handwritten icons. In *11th International Conference on the Frontiers of Handwriting Recognition*, pages 296–301, Montreal, Canada, August 19-21 2008.
- [6] Eugene H. Ratzlaff. Inter-line distance estimation and text line extraction for unconstrained online handwriting. In *In Proc. of the 7 th IWFHR*, pages 33–42, 2000.
- [7] Craig Saunders, Alex Gammerman, and Vladimir Vovk. Transduction with confidence and credibility. In Thomas Dean, editor, *16th International Joint Conference on Artificial Intelligence (IJCAI-1999)*, pages 722–726, Stockholm, Sweden, July 31 - August 6 1999. Morgan Kaufmann.
- [8] Ernesto Tapia and Raul Rojas. A survey on recognition of on-line handwritten mathematical notation. Technical Report 07-01, Freie Universitat Berlin, Germany, 2007.
- [9] C. C. Tappert, A. S. Fox, J. Kim, S. E. Levy, and L. L. Zimmerman. Handwriting recognition on transparent tablet over flat display. *Society for Information Display Digest of Technical Papers*, XVII:308–312, May 1986.
- [10] C. C. Tappert, C. Y. Suen, and T. Wakahara. The state of the art in online handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(8):787–808, 1990.
- [11] Stijn Vanderlooy and Ida Sprinkhuizen-Kuyper. A comparison of two approaches to classify with guaranteed performance. In *PKDD 2007: Proceedings of the 11th European conference on Principles and Practice of Knowledge Discovery in Databases*, pages 288–299, Berlin, Heidelberg, 2007. Springer-Verlag.
- [12] Stijn Vanderlooy, Ida Sprinkhuizen-Kuyper, Evgueni Smirnov, and H. Jaap van den Herik. The ROC isometrics approach to construct reliable classifiers. *Intelligent Data Analysis*, (13):2–37, 2009.
- [13] Stijn Vanderlooy, Laurens van der Maaten, and Ida Sprinkhuizen-Kuyper. Off-line learning with transductive confidence machines: an empirical evaluation. In Petra Perner, editor, *Proceedings of the 5th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM 2007)*, volume 4571 of *Lecture Notes in Computer Science*, pages 310–323, Leipzig, Germany, July 18-20 2007. Springer.
- [14] Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*. Springer, New York, NY, USA, 2005.