Textons

Maria Camila Escobar
Universidad de los Andes
Bogotá D.C, Colombia
mc.escobar11@uniandes.edu.co

Laura Gongas
Universidad de los Andes
Bogotá D.C, Colombia
l.gongas10@uniandes.edu.co

***Abstract*-Image texture classification is a powerful tool that has expanded in the latest years. Several solution algorithms to problems, such as segmentation, have benefited with the use of texture recognition to achieve better results. Here we apply a method for texture classification in a database with 25 different textures. We then identify the best classifier and tune the parameters in order to obtain the best possible outcome. It was possible to obtain an average classification accuracy (ACA) of 42.8% wit the Random Forest classifier.**

1. Introduction

Image textures are patterns or elements that may or may not have defined structures. Texture analysis can be approached through structure or statistics. The statistical approach defines image texture as a measure of intensity arrangement in a region, while the structured approach considers it as a set of primitive units in a regular or repeated pattern [3].

Textons are fundamental structures of images which are able to represent pixel relationships in a region. Therefore, they're useful for image texture analysis [3]. Texton calculation consists of selecting a filter bank, then applying it to train images and finally creating a texton dictionary by clustering filter responses. Afterwards, a texton map is constructed by filtering test images and assigning them the nearest cluster label. Finally, image textures are usually represented with a texton histogram.

In this paper we use texton histograms as texture descriptors to classify images through K Nearest Neighbor (KNN) and Random Forest (RF) classifiers. KNN consists of creating a Voronoi diagram to partition the train dataset. Then, each test point is compared to its k nearest neighbors through a specified metric and it's assigned to the same class as the nearest category. The metrics include distances such as euclidean or others adapted for histograms such as chi squared or intersection. Furthermore, RF classifiers are a composition of different or similar decision trees (depending on parameters). It's important to take into account that RF models don't work without randomness due to overfitting issues. Some hyperparameters used to include randomness are: bagging which gives different subsamples of the dataset to different trees and number of features a node can access. Other parameters commonly varied in RF models are number of trees and depth of trees (number of nodes per tree) [2].

The objective of this paper is to classify images based on their texture represented by textons and employing RF and KNN as classifiers. The database used is provided by the ponce group and it consists of 25 texture classes, 40 images per each category [4]. The format of the images is JPG, grayscale and all of them have 640x480 pixels. Each category only has one type of texture but with different orientations varying between images. A large amount of categories have line based patterns while some have circular shaped patterns or others.

2. Materials and methods

To classify the images using textons we first created a filter bank, that is a serie of filters that represent certain edges or features that a texture can have. To do this we used the method *FbCreate*. This method uses by default 8 orientations so it creates 32 different filters to apply to each texture. Then we read the train and test images and obtained the corresponding labels for each, the labels were obtained by extracting the number in the image name. We then applied the filter bank to each image. The explanation behind this is that if we apply all the textons to every pixel of the image and then compute which texton is closer to the information of the pixel then we will be able to identify certain patterns.By doing a qualitative analysis we can identify that the filters that are in diagonal orientations will end up being more discriminative than others. This would happen because the patterns of the textures that are going to be classified are mostly in diagonal orientation. However, for some specific patters that do not have a determined orientation, such as marble or granite, every filter will be able to provide valuable information.To apply the filter bank to each train and test dataset and obtain the texture descriptors we used *fbRun*.

Afterwards we classified the textons by running kmeans on the train filter bank, this was done via *computeTextons*. Then we assigned the different textons to the filtered datasets to obtain a texton map for each image and we calculated the normalized texton histograms to represent the textures. After we are done with this process we will have a texton representation of the image, this tells us the specific details and patterns that are repeated in the image and how frequently we may find them.

After representing the texture of the images with histograms, these were used as descriptors for two different classifiers: KNN and RF. First, the different models were created with specified parameters, then trained with the train dataset, the test images were classified by the model and the results were evaluated with the ACA of the normalized confusion matrix.

Various experiments were performed to determine the best hyperparameter values by fixing all except one in each run. For both classifiers we varied: the amount of random pixels of each image used to compute textons and the number of train and test images. Also, for the KNN classifier, the parameters taken into account were the amount of neighbors and the metric used to obtain distances between histograms (euclidean, intersection and chi squared). On the other hand, the hyperparameters evaluated for the RF classifier were: number of trees, maximum features to consider at each node and depth of the trees.

3. Results

We ran experiments to identify good configurations of hyperparameters for both KNN and RF classifiers. We consider a good configuration is one that has the biggest ACA but also that doesn't take much time running. The time consideration is an extra metric that we decided to measure because it's a direct measurement of how complex the algorithm is and sometimes we do not need as much complexity to achieve similar results.

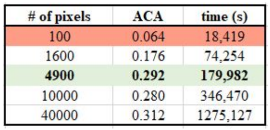
Two hyperparameters were varied in the same way for both classifiers. First, we tested the number of pixels that were going to be filtered from each image. Since every image had a resolution of 640x480 pixels, it was nearly impossible to filter all of it without taking a significant amount of time and computer memory. Therefore, we decided that the best possible way to solve this problem was by choosing randomly a certain number of pixels, creating a new image with these and then filtering it. This is a valid solution because textures are local patterns, so if we choose a correct number of random pixels we could get similar filtered information as if we worked with the entire image.

Another hyperparameter tested for both classifiers was the amount of images in the train and test dataset. We decided to choose randomly a certain quantity of images for training and the same quantity for testing. By doing so, we were trying to ensure that every time the method was executed we obtained the best possible outcome because of the optimization that was made and not because the train and test models were perfectly fitted. Also, we wanted to determine if there was a significant change of results with an increase of number of images taking into account the running time.

K Nearest Neighbor classifier

First, we decided to change the number of random pixels with the values of table 1. We found that the worst ACA belonged to 100 random pixels. This makes sense because with only 100 pixels you cannot possibly get information regarding the texture, so every classification would be made by chance, hence the ACA value very similar to the chance of choosing one out of 25 categories. We decided that the best configuration was with 4900 pixels because the processing time is 7 times lesser than with the best ACA and the ACA difference is only of 2%. Fixed parameters of the table include number of images(ni), nearest neighbors (nn) and metric (m).

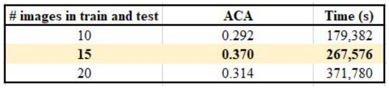
Table 1. Results of hyperparameter testing with KNN. Parameter tested: number of pixels (np). Other parameters: ni-10 nn-25 m-chi



# of pixels	ACA	time (s)
100	0.064	18,419
1600	0.176	74,254
4900	**0.292**	**179,982**
10000	0.280	346,470
40000	0.312	1275,127

The next hyperparameter that was tested was the number of images used for the training and testing phase. The result of this test is shown in table 2, here we found that the best value for this parameter is 15 images for each step. This result can be because if we used only 10 images per step the algorithm would not be able to extract as much similar features for each category as using 15 images per step. However, when we switch to 20 images per step we find that the ACA value decreases, this can happen because some images in each category may differ so the algorithm will start making features that adjust to every image in the training dataset but that does not properly describe the texture.

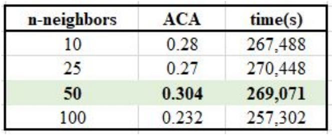
Table 2. Results of hyperparameter testing with KNN. Parameter tested: number of images per dataset (ni). Other parameters: np-4900 nn-25 m-chi



# images in train and test	ACA	Time (s)
10	0.292	179,382
15	**0.370**	**267,576**
20	0.314	371,780

Moving on to the intrinsic parameters of the classifier we chose to evaluate the difference in outcomes when we changed the number of neighbors that would be compared in the K nearest neighbor method. Choosing a proper number of neighbors is relevant to the outcome because a small number of neighbors will present biased results to certain categories but a very large number of neighbors will only confuse the outcome. We conducted this test and the results can be seen in table 3. As we mentioned before, a small and a big number of neighbors did not provide an ACA value as good as the one in an intermediate number of comparing neighbors.

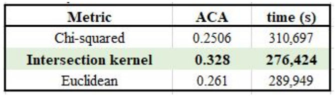
Table 3. Results of hyperparameter testing with KNN. Parameter tested: number of neighbors (nn). Other parameters: ni-10 np-4900 m-chi



n-neighbors	ACA	time(s)
10	0.28	267,488
25	0.27	270,448
50	**0.304**	**269,071**
100	0.232	257,302

The last hyperparameter that we decided to evaluate was the distance metric used for the classification method. We compared Euclidean distance, chi-squared distance and the intersection kernel. Out of these three metrics we found that the intersection kernel is widely used for histogram based indexing and classification [1]. The results that can be seen in table 4 are consistent with our finding because it shows that the best ACA was obtained with the intersection kernel.

Table 4. Results of hyperparameter testing with KNN. Parameter tested: metric (m). Other parameters: ni-10 nn-25 np-4900



Metric	ACA	time (s)
Chi-squared	0.2506	310,697
Intersection kernel	**0.328**	**276,424**
Euclidean	0.261	289,949

We provide the confusion matrix for the best and the worst configurations in the test dataset. Our worst configuration for KNN is the one that can be seen in table 1 with an ACA of 6.4%. The confusion matrix for this configuration is provided in table 13. Here we can see in light red the number of categories that did not have any prediction in them, we can also observe that the second category was the one with the most predicted values. We cannot say, however, that the percentage of 70% images correctly classified on the second category is a real measure because the algorithm was classifying merely by chance. If we analyze visually the textures we will find that the second category, the one that corresponds to bark 2, has smaller patterns and they go in multiple directions. Because of that, if we take only a sample of 100 pixels out of the image, this category will have the higher correlation with almost every filter and when we run the classification method for the test dataset they will be classified as the second category most of the time.

Continuing with our best configuration for KNN, with an ACA of 37%, we found that it was the one in table 11 which belonged to using 15 images for each dataset. Here we found that only one category did not have any prediction in it and the performance of the classification was significantly better than the one for the worst-case scenario. However, we found that 4 categories were not classified correctly in any prediction. If we analyze the textures these categories belong to we find that the images from these categories are quite different from one another. These differences are due to different scales and orientation of the image with the pattern. For example, for category 9, which corresponds to marble, the pattern is not the same for every image, therefore, the classifier does not correctly predict images in this category. A possible way to solve this problem would be adding even more images to the training dataset so that the classifier has a more robust amount of information to correctly predict the different variants of the texture.

On the other hand, the category that was classified the best was upholstery (category 20). We consider that this was the easiest category to classify because of two reasons. First, the images in this category have the exact same pattern in the same scale so it less complicated to classify. Additionally, the images from this texture are very different from the ones in every other category, they have specific traits, for example the cross-like pattern, that can be represented in a texton map and the classifier will immediately recognize that it belongs to the upholstery category.

In the tables 12 and 14 we can see the confusion matrix for the best and the worst case scenario of the KNN classifier. Something interesting about this trained classifier is that it does not work perfectly for the train dataset, giving ACAs of 40% and 9% for each case. Even though the ACAs given are higher than the ACAs obtained with the test dataset in each case the difference is not very significant. This results show that perhaps the KNN classifier was underfitting the data, this means that the KNN classifier had too little model capacity to correctly predict the results. Therefore, we recommend adding more parameters to the classifier in order to increase the complexity and thus the model capacity of it.

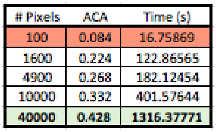
Some additional parameters that weren't taken into account for the KNN classifiers are: weights and the algorithm used to compute nearest neighbors. The former assigns different weights to points so near points can contribute more to classification than faraway points. Choosing the latter parameter is complicated because it depends in the number of samples, data structure, among others. However, it's generally recommended for big problems and KDTree or BallTree are commonly used [5].

Random Forest classifier

As mentioned before, common parameters varied for both classifiers were the number of random pixels used to represent the images (np) and the amount of images in the dataset (ni). The intrinsic parameters considered for the RF classifier were: number of trees (nt), maximum features per node (mf) and tree depth (td). The latter was the last parameter tested and was set to default for other experiments, this means that the nodes were expanded until all leaves were pure.

Table 5 shows an ACA increase with more random pixels. This was an expected result because more pixels means there's more available information for the RF model. However, the computing time significantly increases and therefore we used 10000 pixels as a fixed value for other experiments. This table contains the overall worst (8.4%) and best (42.8%) ACA results which demonstrates the high relevance of this parameter for the RF classifier.

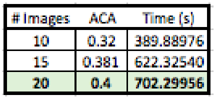
Table 5. Results of RF hyperparameter testing. Parameter tested: number of pixels. Other parameters: ni-10, nt-30, mf-50%, td-default



# Pixels	ACA	Time (s)
100	0.084	16.75869
1600	0.224	122.86565
4900	0.268	182.12454
10000	0.332	401.57644
40000	**0.428**	**1316.37771**

Table 6 shows the experiments performed varying the amount of images in the train and test datasets. As in the previous experiment we expected to get better results with more images because this meant more available infromation for the classifier model. However, neither the ACA nor the time varied as much like in the number of pixels experiment.

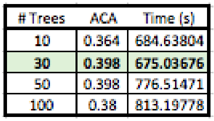
Table 6. Results of RF hyperparameter testing. Parameter tested: number of images per dataset. Other parameters: np-10000, nt-30, mf-50%, td-default



# Images	ACA	Time (s)
10	0.32	389.88976
15	0.381	622.32540
20	**0.4**	**702.29956**

With respect to the intrinsic parameters of the RF classifier, we started varying the number of trees (Table 7). According to literature, the larger amount of trees the better results until a critical number of trees were results stop getting significantly better [5]. This is consistent with our results which show an increase of ACA from 10 trees to 30 and a lack of improvement afterwards with increasing number of trees. Based on computing time and ACA, 30 trees was chosen as an optimal value.

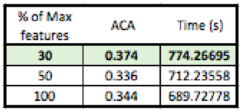
Table 7. Results of RF hyperparameter testing. Parameter tested: number of trees. Other parameters: np-10000,ni-20, mf-50%, td-default



# Trees	ACA	Time (s)
10	0.364	684.63804
30	**0.398**	**675.03676**
50	0.398	776.51471
100	0.38	813.19778

Also, we evaluated the performance of the model with varying maximum features considered when splitting a node (Table 8). For low values of the parameter there's a lower variance but a greater bias [5]. In other words, less features considered per node increase the randomness of the model and therefore trees are more different from one another. However, a high bias and low variance correspond to an underfitted model which is too simple to completely represent significant characteristics of each class. On the other hand, if the number of maximum features is increased, then there would be an overfitting and the model will consider irrelevant characteristics. This is consistent with our results because when each node can view all the features (100%) there's a worst result than when they have access to 30% of the characteristics. However, it has been reported that for classification problems, good results are achieved when maximum features are equivalent to the square root of the total number of features [5]. For future experiments it's recommended to test the previous configuration.

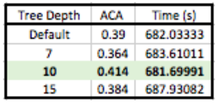
Table 8. Results of RF hyperparameter testing. Parameter tested: maximum features considered per node. Other parameters: np-10000,ni-20,nt-30, td-default



% of Max features	ACA	Time (s)
30	**0.374**	**774.26695**
50	0.336	712.23558
100	0.344	689.72778

Finally, we tested the depth of the trees (Table 9). We obtained that a low an high tree depth provide a worst ACA than an intermediate. We expected this result because a high depth tree memorizes the train dataset and there's a higher test error (overfitting). On the other hand, a low depth makes very simple trees which can't represent all the relevant characteristics (underfitting). Even though we didn't obtain good results for the default setting of this parameter, literature reports that a combination between default depth and minimum samples for split of 2 results in good ACA values [5]. Therefore, for future experiments we recommend that if the depth is set to default the minimum samples to split a node should be specified.

Table 9. Results of RF hyperparameter testing. Parameter tested: tree depth. Other parameters: np-10000,ni-20,nt-30, mf=30%



Tree Depth	ACA	Time (s)
Default	0.39	682.03333
7	0.364	683.61011
10	**0.414**	**681.69991**
15	0.384	687.93082

Confusion matrices for the overall best and worst configurations found with the previous experiments are presented in tables 15,16,17, 18. These configurations correspond to the green and red labels in Table 5.

The best configuration for the test dataset classification yielded an ACA of 42.8%. From the normalized confusion matrix (Table 15) we can see that only one category was perfectly classified and it corresponds to the upholstery category (20). This category was also the best classified by the KNN classifier and as mentioned before, this could be due to their difference from other groups and the same scale in all images from this category. Also, from the same matrix we can see that 3 categories didn't have any images correctly classified. These groups correspond to marble (9), wallpaper (21) and plaid (25). Group 9 was also problematic for the KNN classifier and this was attributed to drastic changes of pattern in different images. On the other hand, category 25 had a large amount of incorrect classifications and the majority of this images were classified in category 14 (brick1). This might be caused because plaid has lines in horizontal and vertical directions in order to conform squares similar to bricks. However, a lot of images from the bark (2) and granite (8) categories were classified as plaid even though those images have more irregular patterns that don't include straight lines. Additionally, a lot of images were classified as bricks probably due to the straight lines and different orientations that are similar to other textures.

Table 17 presents the confusion matrix for the worst configuration of RF classifier (ACA of 8.4%). As in the KNN classifier, categories 12 (pebbles) and 19 (carpet2) don't have any correct prediction. It's important to take into account that both categories have patterns that don't correspond to straight lines. In contrast, the best classified categories include wall (13), brick (15) and corduroy (24), all of which have images that include straight lines. Based on this, the filter bank may not be very adequate for textures containing patterns different to straight lines. Therefore, it's recommended to add filters that detect more circular shapes.

Finally, Tables 16 and 18 represent confusion matrices for the model trained with the train dataset and evaluated with it too. The best configuration resulted with a worst ACA for train images prediction. Based on this we can say that a model that's completely adjusted to the train dataset (ACA of 100%) will result in bad classification of the test images. Nonetheless, the ACA in the best configuration is still too high (97.4%) and therefore it's recommended to construct a simpler model that doesn't have a large variance. In other words, we constructed a model very sensitive to the sample.

Some additional hyperparamaters that weren't used for this classifier were the purity threshold required to split a node and bagging. The latter is used to show different subsets of data to trees and therefore reduce the variance of the model by increasing randomization. It has been reported that bagging methods can significantly improve results so it's recommended to include this hyperparameter in future experiments. This parameter could also be used with the KNN classifier but due to the fact that it reduces overfittting in strong and complex models, it might not be a good idea to use it for this specific problem [5].

Each classifier had its own limitations, KNN produced underfitting while RF did overfitting. In other words, the KNN model was too simple to represent all the characteristics while RF fitted irrelevant data. However, the over fitting in RF classifiers can be controlled with randomization. Consequently, another limitation arises and it's the large amount of hyperparameters that have to be tuned. Also, the borders of the Voronoi diagram in KNN are very sensitive to noise in annotation.

Another limitation of the method was that it was time expensive. Table 10 shows the computing time required for different sections necessary for the creation of the texton dictionary and considers two configurations of number of pixels and of images. We can see that for a large amount of data the time significantly increases specially due to the last part of the process which includes kmeans. Also, applying the filter bank is time expensive when there's a lot of data. The method could be improved by optimizing the step were new images are created with random selected pixels. Also, based on the tables of all the experiments we can observe that RF takes much longer than KNN but this is caused by the chosen hyperparameters which makes this last classifier more complex.

Finally, some general limitations we found regardless of the method were the fact that our train and test dataset were randomly selected each time we ran the method. This was made in order to compensate the results from being overfitted, because if the datasets changed every time and the ACA result was still consistent then it would mean that the classifier was doing a good job. However, this was not always the case because, as we can observe in the KNN hyperparameter testing figures, the ACA varied significantly in runs with the same hyperparameters. This might be related to the sensibility of the KNN classifier to noise in annotations.

4. Conclusions

By doing a comparison of the ACA measure obtained for the best configurations of both the KNN and the RF classifiers we are able to confirm that the classifier that worked best for this problem was Random Forest. This difference of 5.8% in ACA may be attributed to the fact that RF has more parameters to tune, therefore, when all relevant parameters have been optimized to get the best possible outcome we observe better performance in multi class classification.

Images weren't preprocessed before obtaining the textons which may be a source of error due to noise in the images. Also, we noticed that RF was overfitted so for more randomization we recommend to perform bagging. On the other hand, KNN was underfitted so an algorithm such as KDTree might improve results, however it's recommended to use RF instead of KNN for this problem. Finally, for this dataset it's necessary to use not only straight line filters but others that recognize circular patterns too.

References

[1] S.-H. Cha and S. N. Srihari. On measuring the distance between histograms. *Pattern Recognition*, 35(6):1355–1370, 2002.

[2] A. Criminisi, J. Shotton, and E. Konukoglu. *Decision forests*. Now, 2012.

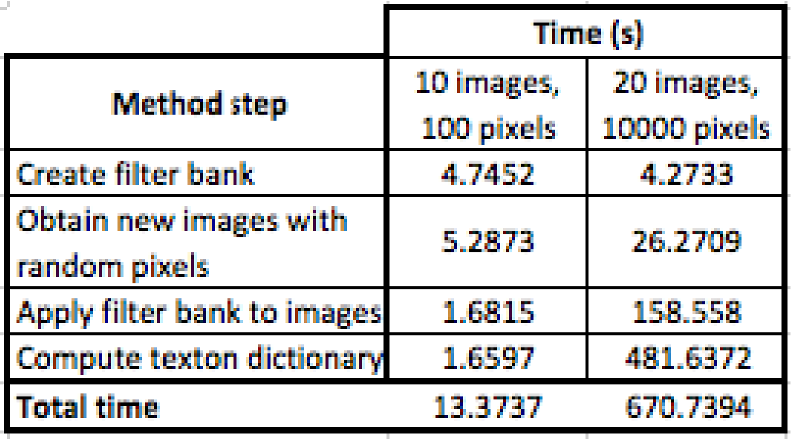
[3] Y. Javed and M. M. Khan. Image texture classification using textons. *Emerging Technologies*, 2011.

[4] S. Lazebnik, C. Schmid, and J. Ponce. A sparse texture representation using local affine regions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1265–1278, 2005.

[5] 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. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Additional attachments

Table 10. Computation time of the different steps required to create a texton dictionary.



Method step	Time (s)	
	10 images, 100 pixels	20 images, 10000 pixels
Create filter bank	4.7452	4.2733
Obtain new images with random pixels	5.2873	26.2709
Apply filter bank to images	1.6815	158.558
Compute texton dictionary	1.6597	481.6372
Total time	13.3737	670.7394

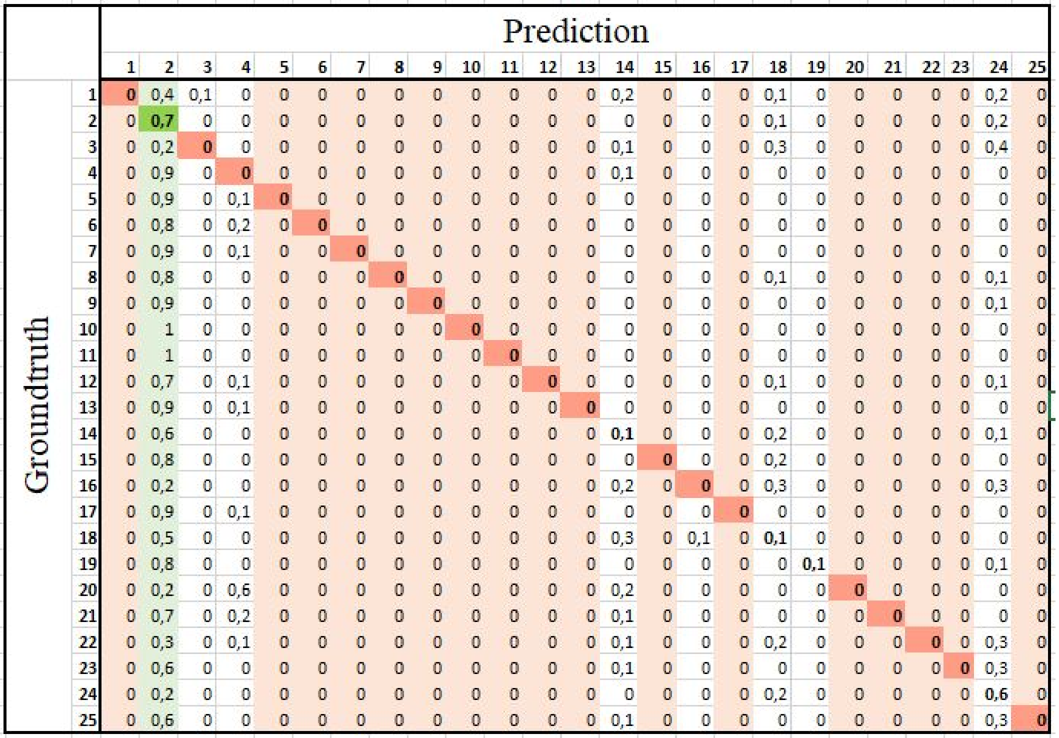
Table 11. Confusion matrix for the test dataset in the best configuration of KNN. ACA=37%

		Prediction																								
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Groundtruth	1	**0,4**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,4	0	0	0	0	0	0	0	0,1	0,1
	2	0	**0,1**	0	0	0	0	0	0,1	0	0	0,1	0	0	0,3	0	0	0	0,4	0	0	0	0	0	0	0
	3	0,2	0	**0,6**	0	0	0	0	0	0	0	0	0	0	0	0	0,1	0	0	0	0	0	0	0	0,1	0
	4	0	0	0	**0,3**	0,1	0	0,1	0	0	0,1	0	0	0	0	0,1	0	0,1	0	0	0	0,1	0,1	0	0	0
	5	0	0	0	0,1	**0,1**	0	0,1	0	0	0	0,1	0,1	0,2	0	0	0	0,3	0	0	0	0	0	0	0	0
	6	0	0	0	0,1	0	**0,3**	0,1	0	0	0,1	0	0,1	0	0	0	0	0,3	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	**0,7**	0	0	0,1	0	0	0	0	0	0	0,2	0	0	0	0	0	0	0	0
	8	0	0	0	0,1	0	0	0	**0,4**	0,1	0	0	0,2	0	0	0,1	0	0	0	0	0	0	0	0	0	0,1
	9	0,1	0	0	0	0	0,1	0	0,2	**0**	0	0,3	0	0	0	0	0	0	0	0	0	0	0,2	0	0	0,1
	10	0	0	0	0	0	0	0,4	0	0	**0,4**	0	0	0	0	0	0	0,2	0	0	0	0	0	0	0	0
	11	0	0,1	0	0	0,2	0	0	0,1	0	0	**0,1**	0,2	0	0,1	0	0	0	0	0,1	0	0	0,1	0	0	0
	12	0	0	0	0,2	0	0	0	0,3	0	0	0	**0**	0	0,1	0,1	0	0	0	0	0	0,2	0	0	0	0,1
	13	0	0	0	0,4	0,1	0	0	0	0	0	0	0	**0,5**	0	0	0	0	0	0	0	0	0	0	0	0
	14	0,4	0,2	0	0	0	0	0	0,1	0	0	0	0	0	**0,3**	0	0	0	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0,3**	0	0,1	0,1	0	0	0,1	0,4	0	0	0
	16	0,3	0,1	0,2	0	0	0	0	0	0	0	0	0	0	0	0,1	**0,2**	0	0	0	0	0	0	0	0,1	0
	17	0	0	0	0,1	0	0	0,2	0	0	0,3	0	0	0	0	0	0	**0,4**	0	0	0	0	0	0	0	0
	18	0	0,1	0	0	0	0	0	0,2	0	0	0	0	0	0,2	0	0	0	**0,5**	0	0	0	0	0	0	0
	19	0,1	0	0	0,1	0	0	0	0,2	0	0	0	0	0	0	0	0	0	0	**0**	0	0,2	0,1	0	0	0,3
	20	0	0	0	0	0	0	0,1	0	0	0	0	0	0	0	0	0	0	0	0	**0,9**	0	0	0	0	0
	21	0	0	0	0,1	0	0	0	0,3	0	0	0	0,2	0	0,1	0	0	0	0,1	0,1	0	**0,1**	0	0	0	0
	22	0	0	0	0,2	0	0	0	0,3	0	0	0	0	0	0	0	0	0	0,1	0,1	0	0,1	**0,2**	0	0	0
	23	0	0	0	0,1	0	0	0	0,2	0,1	0	0,1	0,1	0	0,1	0	0	0	0,2	0	0	0,1	0	**0**	0	0
	24	0,3	0	0	0	0	0	0	0	0	0	0	0	0	0,1	0	0,3	0	0,1	0	0	0	0	0	**0,2**	0
	25	0	0,2	0	0	0	0	0	0,3	0	0	0	0	0	0	0	0	0	0,2	0	0	0	0,1	0	0	**0,2**

Table 12. Confusion matrix for the train dataset in the best configuration of KNN. ACA=40%

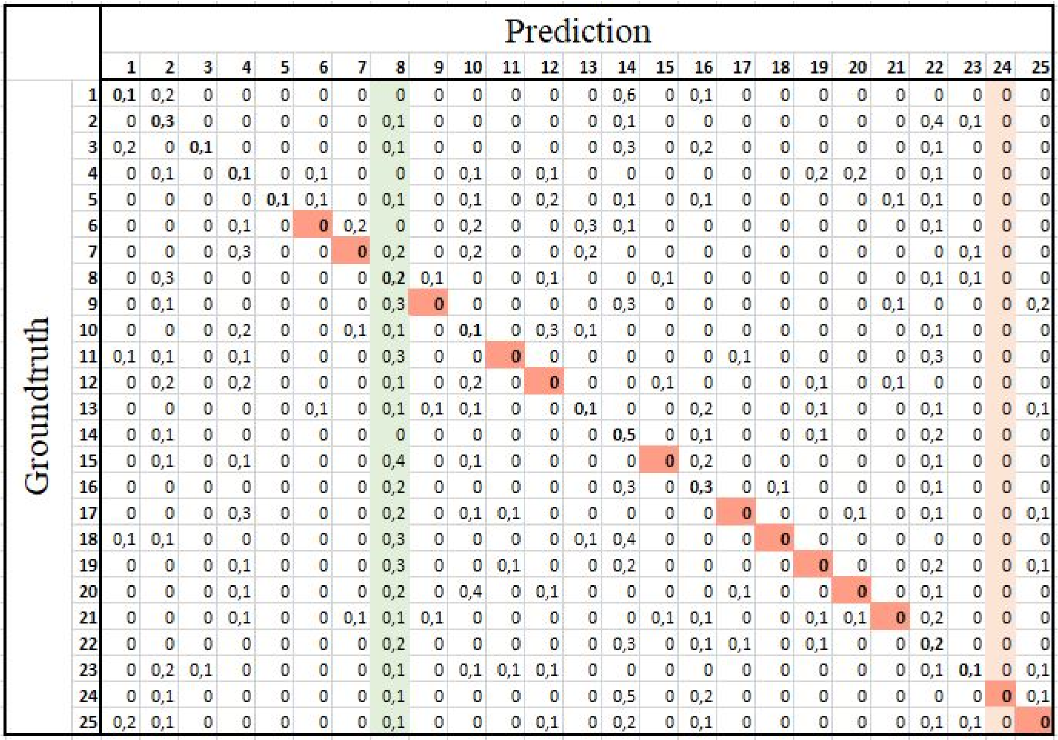
		Prediction																								
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Groundtruth	1	**0,6**	0,1	0	0	0	0	0	0	0	0	0	0	0	0,1	0	0	0	0	0	0	0	0	0	0,1	0,1
	2	0	**0,2**	0,1	0	0	0	0	0,2	0	0	0	0	0	0,1	0	0	0	0,2	0	0	0	0	0,1	0,1	0,1
	3	0,4	0,2	**0,3**	0	0	0	0	0	0	0	0	0	0,1	0,1	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	**0,1**	0	0	0,1	0,1	0,1	0	0	0,3	0	0	0	0	0,3	0	0,1	0	0	0	0	0	0
	5	0	0	0	0,1	**0,1**	0,1	0,1	0,1	0,1	0	0,2	0,1	0	0	0	0	0,1	0	0,1	0	0	0	0	0	0
	6	0	0	0	0,2	0	**0,1**	0,3	0,1	0	0	0	0	0	0	0	0	0,2	0	0	0	0,1	0	0	0	0
	7	0	0	0	0	0	0	**0,7**	0	0	0,1	0	0	0	0	0	0	0,2	0	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0	**0,7**	0	0	0	0	0	0,1	0	0	0	0	0,1	0	0	0,1	0	0	0
	9	0	0	0	0	0,1	0	0	0,2	**0,2**	0	0,1	0	0,1	0,1	0	0	0	0,1	0,1	0	0	0,1	0	0,1	0
	10	0	0	0	0	0	0	0,1	0	0	**0,7**	0	0	0	0	0	0	0,1	0	0	0	0	0,1	0	0	0
	11	0	0,1	0	0	0,1	0	0	0,1	0	0	**0,2**	0	0	0,1	0	0	0	0	0,3	0	0	0	0,1	0	0
	12	0	0	0	0,1	0,1	0	0	0,1	0,1	0	0	**0,3**	0	0	0	0	0,1	0	0,1	0	0	0,1	0	0	0,1
	13	0	0	0	0	0,1	0,1	0	0	0,1	0	0	0,1	**0,7**	0	0	0	0	0	0	0	0	0	0	0	0
	14	0	0,1	0	0	0	0	0	0,2	0	0	0	0	0	**0,4**	0	0	0	0	0	0	0	0	0	0,2	0,1
	15	0	0	0	0	0	0	0	0,1	0	0,1	0	0	0	0	**0**	0	0,1	0,4	0	0	0,1	0,2	0	0	0
	16	0,4	0,1	0	0	0	0	0	0,1	0	0	0	0	0	0	0	**0**	0	0,1	0	0	0	0	0	0,4	0
	17	0	0	0	0	0	0	0,3	0	0	0	0	0	0	0	0	0	**0,7**	0	0	0	0	0	0	0	0
	18	0	0,1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0,5**	0	0	0	0	0	0,2	0,1
	19	0	0,1	0	0	0	0	0	0,1	0,1	0	0	0,1	0	0,1	0	0	0	0	**0,3**	0	0	0,1	0,1	0	0,1
	20	0	0	0	0	0	0	0,1	0	0	0	0	0	0	0	0	0	0	0	0	**0,9**	0	0	0	0	0
	21	0	0	0	0,1	0	0	0	0,2	0	0	0,1	0,1	0	0	0	0	0,1	0	0,1	0	**0,1**	0,1	0	0	0
	22	0	0	0	0	0	0	0	0,1	0	0,1	0	0	0	0,1	0	0	0	0	0,1	0	0,1	**0,4**	0	0	0,2
	23	0	0	0	0	0	0	0	0,2	0	0	0,2	0	0	0,2	0	0	0,1	0,1	0	0	0	0	**0,1**	0	0,1
	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0,1	0	0	0	0	0	0	0	0	0	**0,9**	0
	25	0	0,1	0	0	0	0	0	0,3	0	0	0	0	0	0	0	0	0	0,1	0	0	0	0,1	0	0	**0,5**

Table 13. Confusion matrix for the test dataset in the worst configuration of KNN. ACA=6.4%



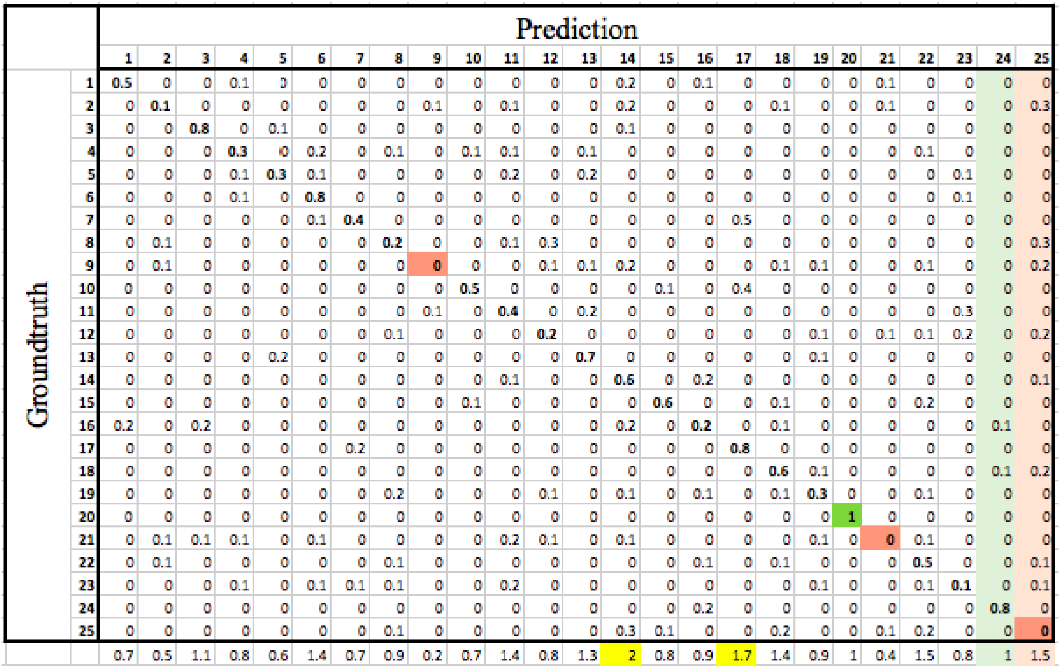
		Prediction																								
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Groundtruth	1	**0**	0,4	0,1	0	0	0	0	0	0	0	0	0	0	0,2	0	0	0	0,1	0	0	0	0	0	0,2	0
	2	0	**0,7**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,1	0	0	0	0	0	0,2	0
	3	0	0,2	**0**	0	0	0	0	0	0	0	0	0	0	0,1	0	0	0	0,3	0	0	0	0	0	0,4	0
	4	0	0,9	0	**0**	0	0	0	0	0	0	0	0	0	0,1	0	0	0	0	0	0	0	0	0	0	0
	5	0	0,9	0	0,1	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	6	0	0,8	0	0,2	0	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	7	0	0,9	0	0,1	0	0	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8	0	0,8	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0	0,1	0	0	0	0	0	0,1	0
	9	0	0,9	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,1	0
	10	0	1	0	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	11	0	1	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	12	0	0,7	0	0,1	0	0	0	0	0	0	0	**0**	0	0	0	0	0	0,1	0	0	0	0	0	0,1	0
	13	0	0,9	0	0,1	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0	0	0	0
	14	0	0,6	0	0	0	0	0	0	0	0	0	0	0	**0,1**	0	0	0	0,2	0	0	0	0	0	0,1	0
	15	0	0,8	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0,2	0	0	0	0	0	0	0
	16	0	0,2	0	0	0	0	0	0	0	0	0	0	0	0,2	0	**0**	0	0,3	0	0	0	0	0	0,3	0
	17	0	0,9	0	0,1	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0
	18	0	0,5	0	0	0	0	0	0	0	0	0	0	0	0,3	0	0,1	0	**0,1**	0	0	0	0	0	0	0
	19	0	0,8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0,1**	0	0	0	0	0,1	0
	20	0	0,2	0	0,6	0	0	0	0	0	0	0	0	0	0,2	0	0	0	0	0	**0**	0	0	0	0	0
	21	0	0,7	0	0,2	0	0	0	0	0	0	0	0	0	0,1	0	0	0	0	0	0	**0**	0	0	0	0
	22	0	0,3	0	0,1	0	0	0	0	0	0	0	0	0	0,1	0	0	0	0,2	0	0	0	**0**	0	0,3	0
	23	0	0,6	0	0	0	0	0	0	0	0	0	0	0	0,1	0	0	0	0	0	0	0	0	**0**	0,3	0
	24	0	0,2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,2	0	0	0	0	0	**0,6**	0
	25	0	0,6	0	0	0	0	0	0	0	0	0	0	0	0,1	0	0	0	0	0	0	0	0	0	0,3	**0**

Table 14. Confusion matrix for the train dataset in the worst configuration of KNN. ACA=9%



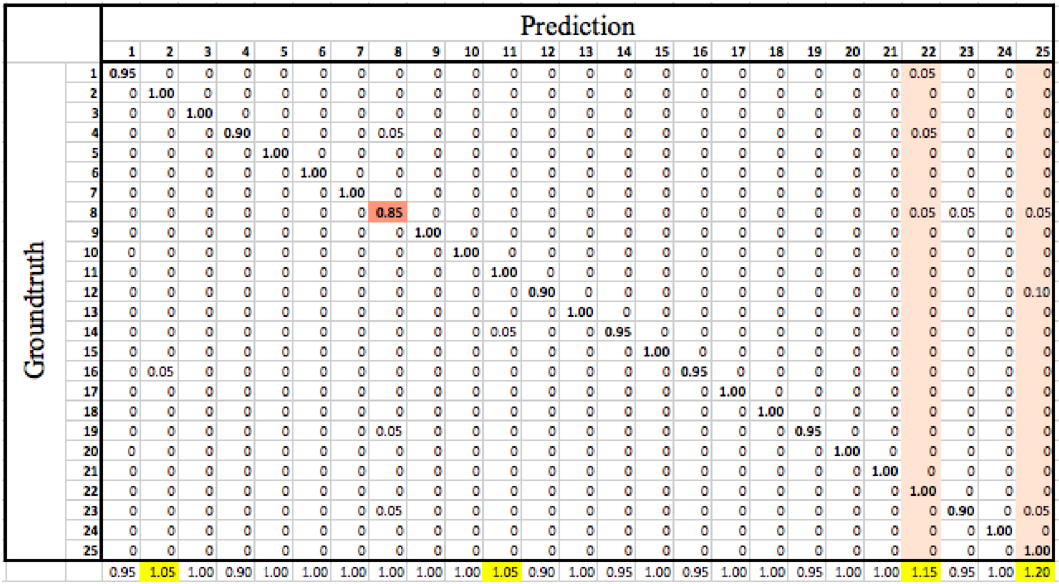
		Prediction																								
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Groundtruth	1	**0,1**	0,2	0	0	0	0	0	0	0	0	0	0	0	0,6	0	0,1	0	0	0	0	0	0	0	0	0
	2	0	**0,3**	0	0	0	0	0	0,1	0	0	0	0	0	0,1	0	0	0	0	0	0	0	0,4	0,1	0	0
	3	0,2	0	**0,1**	0	0	0	0	0,1	0	0	0	0	0	0,3	0	0,2	0	0	0	0	0	0,1	0	0	0
	4	0	0,1	0	**0,1**	0	0,1	0	0	0	0,1	0	0,1	0	0	0	0	0	0	0,2	0,2	0	0,1	0	0	0
	5	0	0	0	0	**0,1**	0,1	0	0,1	0	0,1	0	0,2	0	0,1	0	0,1	0	0	0	0	0,1	0,1	0	0	0
	6	0	0	0	0,1	0	**0**	0,2	0	0	0,2	0	0	0,3	0,1	0	0	0	0	0	0	0	0,1	0	0	0
	7	0	0	0	0,3	0	0	**0**	0,2	0	0,2	0	0	0,2	0	0	0	0	0	0	0	0	0	0,1	0	0
	8	0	0,3	0	0	0	0	0	**0,2**	0,1	0	0	0,1	0	0	0,1	0	0	0	0	0	0	0,1	0,1	0	0
	9	0	0,1	0	0	0	0	0	0,3	**0**	0	0	0	0	0,3	0	0	0	0	0	0	0,1	0	0	0	0,2
	10	0	0	0	0,2	0	0	0,1	0,1	0	**0,1**	0	0,3	0,1	0	0	0	0	0	0	0	0	0,1	0	0	0
	11	0,1	0,1	0	0,1	0	0	0	0,3	0	0	**0**	0	0	0	0	0	0,1	0	0	0	0	0,3	0	0	0
	12	0	0,2	0	0,2	0	0	0	0,1	0	0,2	0	**0**	0	0	0,1	0	0	0	0,1	0	0,1	0	0	0	0
	13	0	0	0	0	0	0,1	0	0,1	0,1	0,1	0	0	**0,1**	0	0	0,2	0	0	0,1	0	0	0,1	0	0	0,1
	14	0	0,1	0	0	0	0	0	0	0	0	0	0	0	**0,5**	0	0,1	0	0	0,1	0	0	0,2	0	0	0
	15	0	0,1	0	0,1	0	0	0	0,4	0	0,1	0	0	0	0	**0**	0,2	0	0	0	0	0	0,1	0	0	0
	16	0	0	0	0	0	0	0	0,2	0	0	0	0	0	0,3	0	**0,3**	0	0,1	0	0	0	0,1	0	0	0
	17	0	0	0	0,3	0	0	0	0,2	0	0,1	0,1	0	0	0	0	0	**0**	0	0	0,1	0	0,1	0	0	0,1
	18	0,1	0,1	0	0	0	0	0	0,3	0	0	0	0	0,1	0,4	0	0	0	**0**	0	0	0	0	0	0	0
	19	0	0	0	0,1	0	0	0	0,3	0	0	0,1	0	0	0,2	0	0	0	0	**0**	0	0	0,2	0	0	0,1
	20	0	0	0	0,1	0	0	0	0,2	0	0,4	0	0,1	0	0	0	0	0,1	0	0	**0**	0	0,1	0	0	0
	21	0	0	0	0,1	0	0	0,1	0,1	0,1	0	0	0	0	0	0,1	0,1	0	0	0,1	0,1	**0**	0,2	0	0	0
	22	0	0	0	0	0	0	0	0,2	0	0	0	0	0	0,3	0	0,1	0,1	0	0,1	0	0	**0,2**	0	0	0
	23	0	0,2	0,1	0	0	0	0	0,1	0	0,1	0,1	0,1	0	0	0	0	0	0	0	0	0	0,1	**0,1**	0	0,1
	24	0	0,1	0	0	0	0	0	0,1	0	0	0	0	0	0,5	0	0,2	0	0	0	0	0	0	0	**0**	0,1
	25	0,2	0,1	0	0	0	0	0	0,1	0	0	0	0,1	0	0,2	0	0,1	0	0	0	0	0	0,1	0,1	0	**0**

Table 15. Confusion matrix for the test dataset in the best configuration of RF. ACA=42.8%



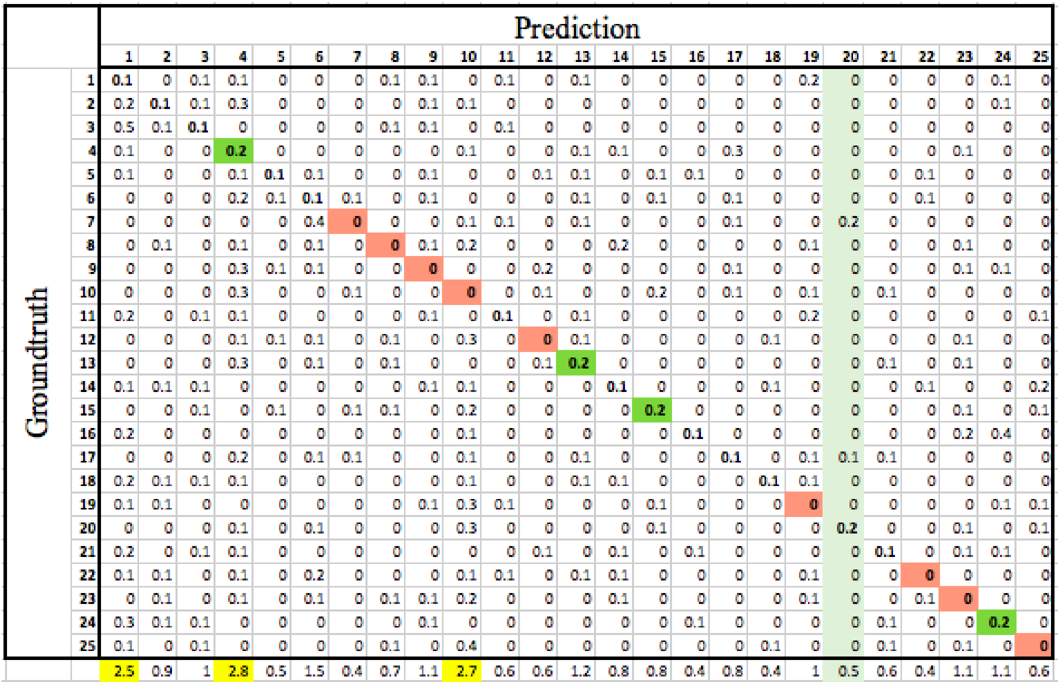
		Prediction																								
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Groundtruth	1	0.5	0	0	0.1	0	0	0	0	0	0	0	0	0	0.2	0	0.1	0	0	0	0	0.1	0	0	0	0
	2	0	0.1	0	0	0	0	0	0	0.1	0	0.1	0	0	0.2	0	0	0	0.1	0	0	0.1	0	0	0	0.3
	3	0	0	0.8	0	0.1	0	0	0	0	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	0.3	0	0.2	0	0.1	0	0.1	0.1	0	0.1	0	0	0	0	0	0	0	0	0.1	0	0	0
	5	0	0	0	0.1	0.3	0.1	0	0	0	0	0.2	0	0.2	0	0	0	0	0	0	0	0	0	0.1	0	0
	6	0	0	0	0.1	0	0.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0
	7	0	0	0	0	0	0.1	0.4	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0
	8	0	0.1	0	0	0	0	0	0.2	0	0	0.1	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0.3
	9	0	0.1	0	0	0	0	0	0	0	0	0	0.1	0.1	0.2	0	0	0	0.1	0.1	0	0	0.1	0	0	0.2
	10	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0.1	0	0.4	0	0	0	0	0	0	0	0
	11	0	0	0	0	0	0	0	0	0.1	0	0.4	0	0.2	0	0	0	0	0	0	0	0	0	0.3	0	0
	12	0	0	0	0	0	0	0	0.1	0	0	0	0.2	0	0	0	0	0	0	0.1	0	0.1	0.1	0.2	0	0.2
	13	0	0	0	0	0.2	0	0	0	0	0	0	0	0.7	0	0	0	0	0	0.1	0	0	0	0	0	0
	14	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0.6	0	0.2	0	0	0	0	0	0	0	0	0.1
	15	0	0	0	0	0	0	0	0	0	0.1	0	0	0	0	0.6	0	0	0.1	0	0	0	0.2	0	0	0
	16	0.2	0	0.2	0	0	0	0	0	0	0	0	0	0	0.2	0	0.2	0	0.1	0	0	0	0	0	0.1	0
	17	0	0	0	0	0	0	0.2	0	0	0	0	0	0	0	0	0	0.8	0	0	0	0	0	0	0	0
	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.6	0.1	0	0	0	0	0.1	0.2
	19	0	0	0	0	0	0	0	0.2	0	0	0	0.1	0	0.1	0	0.1	0	0.1	0.3	0	0	0.1	0	0	0
	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	21	0	0.1	0.1	0.1	0	0.1	0	0	0	0	0.2	0.1	0	0.1	0	0	0	0	0.1	0	0	0.1	0	0	0
	22	0	0.1	0	0	0	0	0	0.1	0	0	0	0	0	0	0	0.1	0	0.1	0	0	0	0.5	0	0	0.1
	23	0	0	0	0.1	0	0.1	0.1	0.1	0	0	0.2	0	0	0	0	0	0	0	0.1	0	0	0.1	0.1	0	0.1
	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0	0	0	0	0	0	0	0.8	0
	25	0	0	0	0	0	0	0	0.1	0	0	0	0	0	0.3	0.1	0	0	0.2	0	0	0.1	0.2	0	0	0
		0.7	0.5	1.1	0.8	0.6	1.4	0.7	0.9	0.2	0.7	1.4	0.8	1.3	2	0.8	0.9	1.7	1.4	0.9	1	0.4	1.5	0.8	1	1.5

Table 16. Confusion matrix for the train dataset in the best configuration of RF. ACA=97.4%



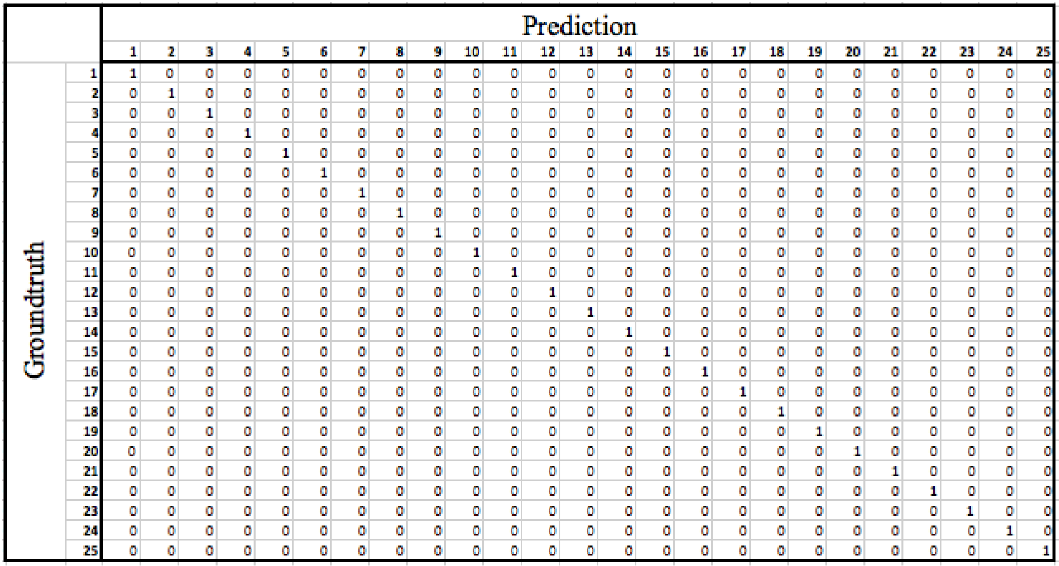
		Prediction																								
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Groundtruth	1	0.95	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.05	0	0	0
	2	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	3	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	0.90	0	0	0	0.05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.05	0	0	0
	5	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0	0.85	0	0	0	0	0	0	0	0	0	0	0	0	0	0.05	0.05	0	0.05
	9	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	10	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	11	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0	0	0	0	0.90	0	0	0	0	0	0	0	0	0	0	0	0	0.10
	13	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0
	14	0	0	0	0	0	0	0	0	0	0	0.05	0	0	0.95	0	0	0	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0
	16	0	0.05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.95	0	0	0	0	0	0	0	0	0
	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0
	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0
	19	0	0	0	0	0	0	0	0.05	0	0	0	0	0	0	0	0	0	0	0.95	0	0	0	0	0	0
	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0
	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0
	22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0
	23	0	0	0	0	0	0	0	0.05	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.90	0	0.05
	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0
	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00
		0.95	1.05	1.00	0.90	1.00	1.00	1.00	1.00	1.00	1.00	1.05	0.90	1.00	0.95	1.00	0.95	1.00	1.00	0.95	1.00	1.00	1.15	0.95	1.00	1.20

Table 17. Confusion matrix for the test dataset in the worst configuration of RF. ACA=8.4%



		Prediction																								
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Groundtruth	1	0.1	0	0.1	0.1	0	0	0	0.1	0.1	0	0.1	0	0.1	0	0	0	0	0	0.2	0	0	0	0	0.1	0
	2	0.2	0.1	0.1	0.3	0	0	0	0	0.1	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0
	3	0.5	0.1	0.1	0	0	0	0	0.1	0.1	0	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4	0.1	0	0	0.2	0	0	0	0	0	0.1	0	0	0.1	0.1	0	0	0.3	0	0	0	0	0	0.1	0	0
	5	0.1	0	0	0.1	0.1	0.1	0	0	0.1	0	0	0.1	0.1	0	0.1	0.1	0	0	0	0	0	0.1	0	0	0
	6	0	0	0	0.2	0.1	0.1	0.1	0	0.1	0	0	0	0.1	0	0.1	0	0.1	0	0	0	0	0.1	0	0	0
	7	0	0	0	0	0	0.4	0	0	0	0.1	0.1	0	0.1	0	0	0	0.1	0	0	0.2	0	0	0	0	0
	8	0	0.1	0	0.1	0	0.1	0	0	0.1	0.2	0	0	0	0.2	0	0	0	0	0.1	0	0	0	0.1	0	0
	9	0	0	0	0.3	0.1	0.1	0	0	0	0	0	0.2	0	0	0	0	0.1	0	0	0	0	0	0.1	0.1	0
	10	0	0	0	0.3	0	0	0.1	0	0	0	0	0.1	0	0	0.2	0	0.1	0	0.1	0	0.1	0	0	0	0
	11	0.2	0	0.1	0.1	0	0	0	0	0.1	0	0.1	0	0.1	0	0	0	0	0	0.2	0	0	0	0	0	0.1
	12	0	0	0	0.1	0.1	0.1	0	0.1	0	0.3	0	0	0.1	0	0	0	0	0.1	0	0	0	0	0.1	0	0
	13	0	0	0	0.3	0	0.1	0	0.1	0	0	0	0.1	0.2	0	0	0	0	0	0	0	0.1	0	0.1	0	0
	14	0.1	0.1	0.1	0	0	0	0	0	0.1	0.1	0	0	0	0.1	0	0	0	0.1	0	0	0	0.1	0	0	0.2
	15	0	0	0.1	0	0.1	0	0.1	0.1	0	0.2	0	0	0	0	0.2	0	0	0	0	0	0	0	0.1	0	0.1
	16	0.2	0	0	0	0	0	0	0	0	0.1	0	0	0	0	0	0.1	0	0	0	0	0	0	0.2	0.4	0
	17	0	0	0	0.2	0	0.1	0.1	0	0	0.1	0	0	0.1	0	0	0	0.1	0	0.1	0.1	0.1	0	0	0	0
	18	0.2	0.1	0.1	0.1	0	0	0	0	0	0.1	0	0	0.1	0.1	0	0	0	0.1	0.1	0	0	0	0	0	0
	19	0.1	0.1	0	0	0	0	0	0	0.1	0.3	0.1	0	0	0	0.1	0	0	0	0	0	0	0	0	0.1	0.1
	20	0	0	0	0.1	0	0.1	0	0	0	0.3	0	0	0	0	0.1	0	0	0	0	0.2	0	0	0.1	0	0.1
	21	0.2	0	0.1	0.1	0	0	0	0	0	0	0	0.1	0	0.1	0	0.1	0	0	0	0	0.1	0	0.1	0.1	0
	22	0.1	0.1	0	0.1	0	0.2	0	0	0	0.1	0.1	0	0.1	0.1	0	0	0	0	0.1	0	0	0	0	0	0
	23	0	0.1	0	0.1	0	0.1	0	0.1	0.1	0.2	0	0	0	0.1	0	0	0	0	0.1	0	0	0.1	0	0	0
	24	0.3	0.1	0.1	0	0	0	0	0	0.1	0	0	0	0	0	0	0.1	0	0	0	0	0.1	0	0	0.2	0
	25	0.1	0	0.1	0	0	0	0	0.1	0	0.4	0	0	0	0	0	0	0	0.1	0	0	0.1	0	0.1	0	0
		2.5	0.9	1	2.8	0.5	1.5	0.4	0.7	1.1	2.7	0.6	0.6	1.2	0.8	0.8	0.4	0.8	0.4	1	0.5	0.6	0.4	1.1	1.1	0.6

Table 18. Confusion matrix for the train dataset in the worst configuration of RF. ACA=100%



		Prediction																								
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Groundtruth	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1