Automation of an image processing software for images of nanoparticles using a Machine Learning model

Klara Baumeister[1], Oscar den Buurman[2], Yoran de Vos[3]

[1] *Faculty IT & Design, The Hague University of Applied Sciences, Johanna Westerdijkplein 75, 2521 EN Den Haag, Netherlands*

[2] *Faculty Technologie, Innovatie & Samenleving, The Hague University of Applied Sciences, Rotterdamseweg 137, 2628 AL Delft, Netherlands*

[3] *Faculty Science and Technologie, University of Applied Sciences Leiden, Le Carrefour Dellaertweg 7F, kern 4, 2316 WZ Leiden, Netherlands*

Abstract

The study of nanoparticles continues to be of great significance in various fields of work, including calculating their sizes from microscopic images for further research, as the Dutch company VSParticle does. This paper focuses on finding a Machine Learning model that predicts the optimal threshold method to use in the image processing software created by VSParticle. To assist VSParticle in solving this problem, we conducted and analysed an experiment with the given data and Machine Learning models in various ways. Our research resulted in a classification problem that was solved through a Logistic Regression model which predicts a manually given user score, and from there chooses the optimal threshold method.

1 Introduction

Nanoparticles have proved to be of high significance in various fields of work, such as medicine, engineering and chemistry**Fout! Verwijzingsbron niet gevonden.**, for products like quantum computers, chemical sensors and photochemical devices such as flat panel displays, among others.**Fout! Verwijzingsbron niet gevonden.** The particles have a core diameter of 1 to 100 nm and range from one single atom to atomic clusters or bulk crystals. The particles both exist in the natural world and are created from human activity.**Fout! Verwijzingsbron niet gevonden.** When analysing nanoparticles, microscopic techniques are used to make the particles visible to the human eye. For analysation and documentation purposes, microscopic images of the particles are often saved and used for further research.**Fout! Verwijzingsbron niet gevonden.**

VSParticle, a company based in Delft, has conducted a lot of research in the field of nanoparticles over the past years.**Fout! Verwijzingsbron niet gevonden.** Calculating the size of nanoparticles using microscopic images is one branch they focus their work on. They have developed a software tool that aids them in doing so by processing the nanoparticle images through different steps, resulting in a final image that allows for the size calculations. One of the processing steps is the so-called thresholding step. Thresholding is the process of turning a greyscale image, where each pixel has a value between 0 and 255, into a bitmap, where each pixel has a binary value of either 0 or 1 (black or white).**Fout! Verwijzingsbron niet gevonden.** The algorithm for such a procedure can be saved as a threshold method in order to be reapplied. VSParticle employs established threshold methods like Yen or Otsu in their software.**Fout! Verwijzingsbron niet gevonden. Fout! Verwijzingsbron niet gevonden.** The process of thresholding can be of great significance in the field of nanoparticles, as it has been employed before to achieve pixel classification**Fout! Verwijzingsbron niet gevonden.**, and allows VSParticle to have a distinct differentiation between background pixels and nanoparticle pixels.

It is important to note that we did not work with the actual images but instead had feature scores that indicate the characteristics of the image after each step in the software. They are created by the software itself and vary based on each step, but generally they indicate a better image the closer they are to 1. The user score however is a score from 1 to 10, given manually by the user of the software after each run.

The paper is mainly intended for employees at VSParticle and the lecturers of our minor at The Hague University of Applied Sciences, in which context our research is taking place. Our research question reads as follows:

How can a Machine Learning model, that predicts the optimal thresholding algorithm, assist VSParticle to analyse nanoparticle images?

To answer our research question in detail, we focused on the following four sub-questions:

1. How does the given data need to be restructured in order to be useful for the model?
2. What features of the dataset should be selected for the model?
3. What type of model do we need and how is it structured?
4. How can the predictions of the model, graded by VSParticle's user, be employed to improve the model over time?

This paper focuses on a way to aid VSParticle by developing a Machine Learning model that predicts the best threshold method to be used on an image, based on previous selections made manually by the software's users. This is done by predicting the user scores for each threshold method, thus determining which is the best.

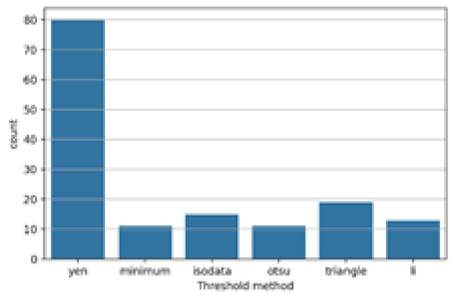
2 Methodology

2.1 Dataset description

In Word I have the right document formatting

The dataset used for this research is a collection of 220 processed images. It contains the settings used by the tool to process each image, the scores which are calculated by the tool to give an indication of how well each image processing step turned out and the user score.

After transforming the received dataset from VSParticle to a Dataframe containing all the necessary information, we quickly discovered that the threshold method distribution is quite imbalanced as seen in figure X. This makes sense because the tool currently uses the Yen algorithm as the default algorithm. One of the other algorithms can only be used when an image is already processed with the Yen method. Because of this imbalanced threshold distribution within the dataset, we decided to focus on predicting if the best algorithm for an given image would be Yen or not Yen.



Creating a model that predicts Yen or not Yen for an image, requires features which can tell the model for which values Yen is preferable and for which ones it is not. VSParticle created their own features which are a representation of the image for each of the threshold algorithms. Using just these features to train a model is not enough, the model also need something to tell which feature scores are good and which ones are bad. There are scores that are given by the user that can be used for this, but this results in a new obstacle.

In the tool, the model will be implemented in an image processing step that results in a user score.In the tool, the model will be implemented in an image processing step that results in a user score.In the tool, the model will be implemented in an image processing step that results in a user score. This means that when the model needs to make predictions, it can't use the user score. It is possible to create two models that can achieve the same predictions but with different inputs, but this seemed quite difficult for us to achieve. So, we decided to instead try to predict what the user score will be when a threshold algorithm is used.

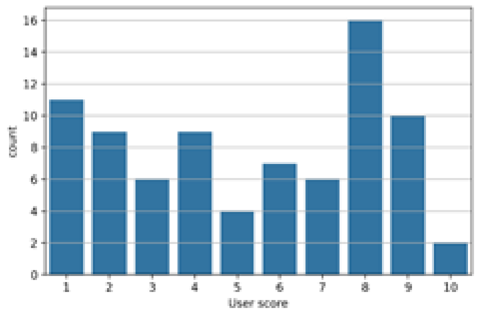
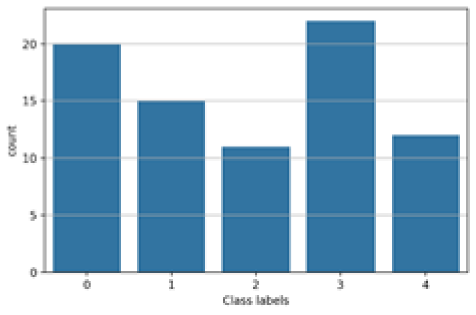


Figure X shows an overview of the distribution of the user scores for the Yen algorithm. This data is still balance the distribution but still represent the user scores clearly.

Figure X shows the user scores from figure X-1 separated into 5 linearly distributed classes. This means that the user scores of 1 and 2 get the class label 0, 3 and 4 get the label 1 and so on. By combining 2 user scores into the same class, it becomes easier for the model to make train on the few samples we have, but it still represents the user scores quite well. If the model would predict a class label of 3 (user scores of 7 and 8) we can tell that the result isn't optimal but is still good.



If for example we would separate the user scores into 3 classes, then it becomes quite a bit more difficult to tell how well the result is. When the user scores are separated into 3 linearly distributed classes, the user scores of 5,6 and 7 would get a class label of 1. In this case it is a lot harder to tell how well the result is.

So, we decided to separate the user scores into 5 classes and let a Machine Learning model make predictions on that.

In the end, the dataset that we can use to train and validate Machine Learning models on contains 58 samples. Each sample has 6 features and 1 class label.

2.2 Model comparison

Our project presents itself as a classification problem due to the prediction of user scores, which are a finite set of categories (limited from 1 to 10). Thus, different classification models can be considered: logistic regression, decision tree, random forest, and multilayer perceptron.

2.2.1 Logistic Regression

Logistic regression brings the advantage of being able to deal with continuous-level predictor variables, similarly to the generated scores being used as features in this project.[9)] [11)] Therefore, especially multivariate logistic regression is relevant to this project, because even if the classes we aim to predict (user scores ranging from 1 to 10) were to be collapsed into less categories containing more records each, keeping a high number of classes leads to a more detailed distinction in the prediction of user scores, which is what we aim for.

2.2.2 Decision Tree

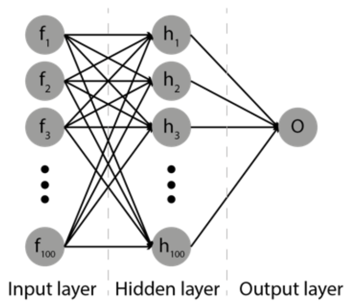
Decision trees are beneficial in a sense that they are able of interpreting the model and identifying important features, which again can be useful to the design of further research.[12)] [13)] Hence, a decision tree model can show a lot of potential for our project, especially for interpreting the data and recognizing important features to employ.

2.2.3 Random Forest

Random forest is a so-called ensemble method, where each decision tree is trained on a random subset of the training set with its own random subset of features. Same as a decision tree, it can be helpful to transform the data into useful signals and find effective parameters. Additionally, does not depend on any specific set of features due to the random selection of such.[10)] This leads to a better generalization over the data, which could be beneficial to our project to discourage the model from overfitting.

2.2.4 Multilayer perceptron

The multilayer perceptron has a structure of one hidden layer and 100 nodes (figure X). Even artificial neural networks like a multilayer perceptron are especially beneficial for pattern recognition tasks in images or audio, it also shows potential for this project's purpose because of the automatic feature-extraction through the hidden layer(s). **Fout! Verwijzingsbron niet gevonden. Fout! Verwijzingsbron niet gevonden. Fout! Verwijzingsbron niet gevonden. Fout! Verwijzingsbron niet gevonden.**



2.3 Experiment

The experiment is a script that generates Data that gives insight into how good a combination of model and Balancing method is going to fit on given parameters. The script is going to generate the data for every combination. Inside the data there are 5 parameters that are important, the Accuracy, Recall, Precision, f1 and the Difference.**Fout! Verwijzingsbron niet gevonden.** These are being calculated using the *sklearn.metrics* library in python. For the difference score we calculate the mean absolute error. The data consists of these scores for each subset (Subset is a combination of the features) of the features.

Principal Component Analysis and Recursive Feature Elimination are algorithm that can be used to select a minimal subset of features that still can give good results. In the early stages of the project we found out that these algorithms gave conflicting results, that's why we decided to create our own way to select a minimal subset of features.

By training and evaluating different models on all the feature combinations, we can get an idea of which feature combination performs well in general. If a feature combination helps different models perform well, then we can consider this feature combination as a good predictor.

To generate the data, we train a model for each of the possible combination that consists of: Model, Balancing method, class amount and features used. This ends up giving a function to calculate the number of iterations the script must do to create and confirm all the models to generate the data:

$$i = (2^f * p) * m * b$$

f = features amount, p = classes amount, m = model amount, b = balancing method amount, i = iterations

The output of the function with the parameters used in the experiment shows us that the script will run 5120 times.

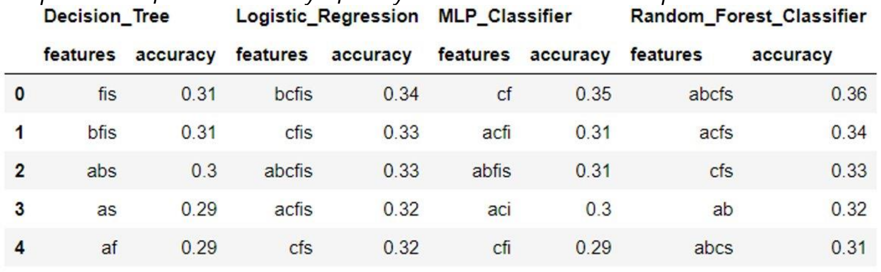
This method is valid since there are a small number of samples available to train the models with.
During testing single models, the results were surprisingly low and given the limited number of samples the results were not accurate enough to draw conclusions. Therefore, this method is used to be able to compare every combination available.

3 Results

In this paragraph the results of the experiment will be shown. To better analyse and find out what the best possible model will be we created different visualisations. These visualisations are created to give us more insight into the separate parts of the data.

3.1 Features

The top 5 results for the accuracy of every model are shown in the picture below.



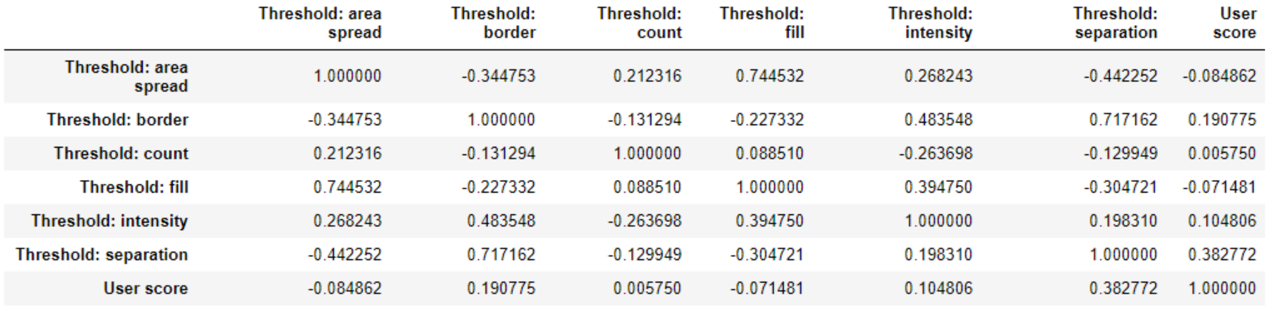
	Decision_Tree		Logistic_Regression		MLP_Classifier		Random_Forest_Classifier	
	features	accuracy	features	accuracy	features	accuracy	features	accuracy
0	fis	0.31	bcfis	0.34	cf	0.35	abcfs	0.36
1	bfis	0.31	cfis	0.33	acfi	0.31	acfs	0.34
2	abs	0.3	abcfis	0.33	abfis	0.31	cfs	0.33
3	as	0.29	acfis	0.32	aci	0.3	ab	0.32
4	af	0.29	cfs	0.32	cfi	0.29	abcs	0.31

For the features we created a scoring metric that would enable us to see which feature combinations performed the best overall. The score is calculated by looking at how many times a combination is in the top 5 for every model. This number would be subtracted of the total models we have (4). This number will be multiplied by 100. To make sure that we could see a more specific ranking we decided that the rank in the top 5 for every row it's in will be added to this number. After applying this metric, we ended up with the result shown below.

```
Rank of the feature combinations
cfs       206
abcfs     300
cf        300
bcfis     300
fis       300
cfis      301
acfi      301
bfis      301
acfs      301
abcfis    302
abs       302
abfis     302
as        303
acfis     303
ab        303
aci       303
cfi       304
af        304
abcs      304
```

The lower the number is the better the feature combination is in general. So this result doesn't apply to only one model.

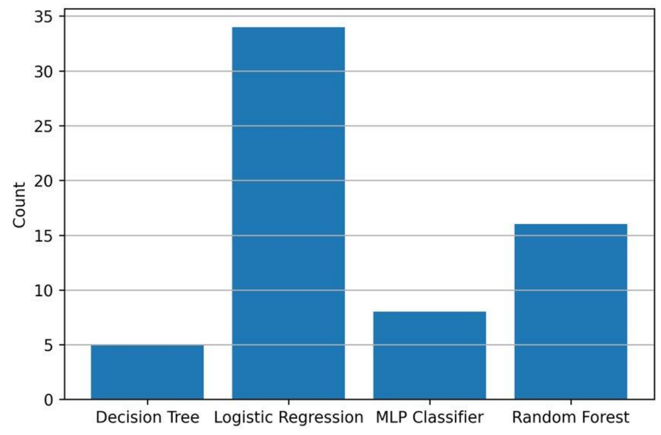
Using the features and User score we created a correlation matrix. The values from this matrix can be seen below.



	Threshold: area spread	Threshold: border	Threshold: count	Threshold: fill	Threshold: intensity	Threshold: separation	User score
Threshold: area spread	1.000000	-0.344753	0.212316	0.744532	0.268243	-0.442252	-0.084862
Threshold: border	-0.344753	1.000000	-0.131294	-0.227332	0.483548	0.717162	0.190775
Threshold: count	0.212316	-0.131294	1.000000	0.088510	-0.263698	-0.129949	0.005750
Threshold: fill	0.744532	-0.227332	0.088510	1.000000	0.394750	-0.304721	-0.071481
Threshold: intensity	0.268243	0.483548	-0.263698	0.394750	1.000000	0.198310	0.104806
Threshold: separation	-0.442252	0.717162	-0.129949	-0.304721	0.198310	1.000000	0.382772
User score	-0.084862	0.190775	0.005750	-0.071481	0.104806	0.382772	1.000000

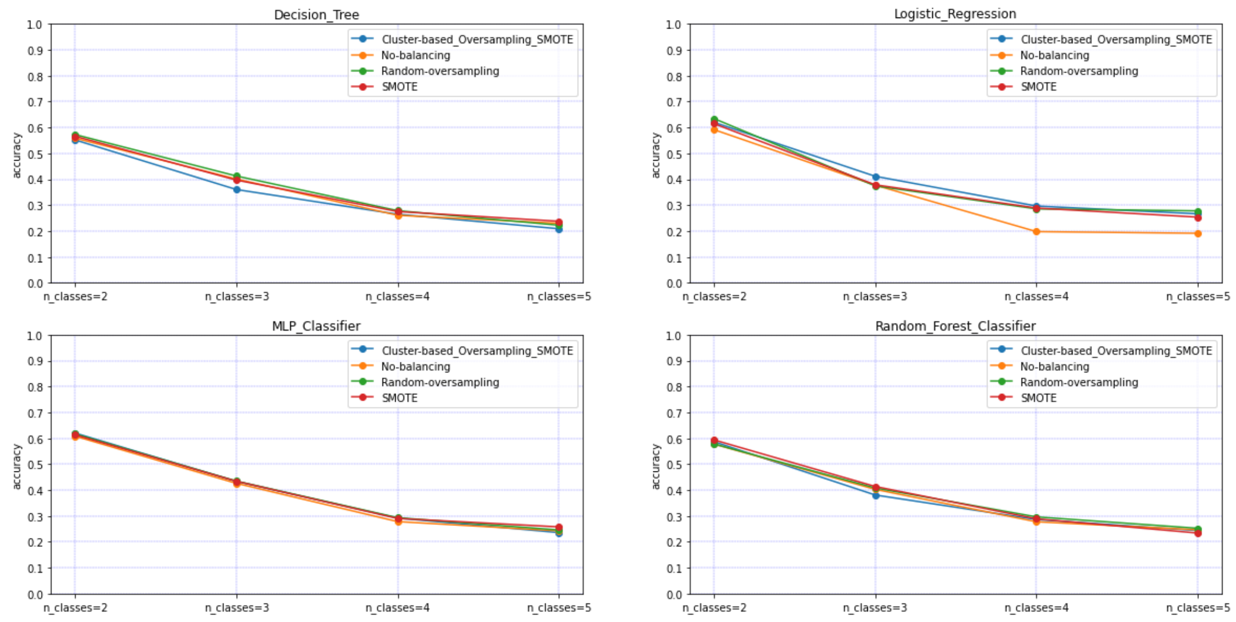
3.2 Models

When looking at the performance of each model we made a bar graph that shows how many times a model was the best when looking at all possible combinations of features.



3.3 Balancing methods

To analyse the Balancing methods, we wanted to know what balancing method would be best for each model. For this we created a line graph that indicates the accuracy when increasing the class size of the user scores for each balancing method. The accuracy represents the average of all the feature combinations.



4 Discussion

4.1 Review

Classes

As the results of the balancing methods show (figure X), the accuracy of a model decreases with an increasing number of classes, which is predictable. However, the threshold of random chance also decreases with more classes: 1/n = random chance, where n = number of classes. This results in a random chance of 0.5 for two classes and 0.2 for five classes. In the experiment, the highest average accuracy two classes achieve is around 0.60, which is still close to the random chance. For five classes, the highest average accuracy possible is around 0.30, which is also close to the random chance. Knowing this, having more classes and a higher distinction between them is preferable despite the drop in accuracy. Therefore, we decided on collapsing the user scores into five classes:

Class label	User score
0	1-2
1	3-4
2	5-6
3	7-8
4	9-10

Model

From figure X we can extract that a Logistic Regression model clearly performs the best the large majority of the time. The success of a Logistic Regression model is also backed up by figure X, where it achieves the second highest accuracy in comparison to the other models. Even though here, an MLP scores a slightly higher accuracy, in order to select a model that not only works best on the given but also on all future data, we decided to use a Logistic Regression model.

Features

Regarding feature selection, the ranking in figure X shows that the feature combination of "cfs" works generally best most of the time, followed by "abcfs", "cf", "bcfis" and "fis", where a=area spread, b=border, c=count, f=fill, i=intensity and s=separation. However, this ranking considers all number of classes and all models, but having already selected these, we can make use of figure X which shows us the highest scoring feature combinations specifically for Logistic Regression: "bcfis", "cfis" and "abcfis".

The combination "cfis" proves to be the most valuable combination in this case. This is because the features "border" and "count", as well as "area spread" and "fill" show to be related to each other as seen in figure X. Collinear features are less useful for a model and potentially lead to non-convergence and unstable models, therefore it is better to drop "border" and "area spread" as features in this case and only use their related counterparts, "count" and "fill" in the combination "cfis".[20] This decision is also backed up by the scoring in figure X, where "bcfis" scores second highest (among others) and "cfis" scores third highest (among others).

Balancing method

Based on figure X, no balancing appears to be harmful to a Logistic Regression model with five classes and leads to a distinct drop in accuracy. Therefore, a balancing method needs to be applied. According to figure X, Random Oversampling works slightly better than SMOTE or Cluster-based Oversampling, and it shows multiple other advantages like simple implementation, which is why we decided to use it for balancing our data.[19]

4.2 Final model

Our analysation leads to the result of a Logistic Regression model with five classes, the input features "cfis" (count, fill, intensity, separation) and the method of Random Oversampling to balance our data.

4.3 Conclusion

4.4 Recommendations

User score dependency

In this paper, we are trying to predict what the score the user will give based on just the features that are used in the thresholding step. One major problem with this is that the user score is not only dependent on this threshold step. Besides, image quality probably affecting the score the user gives, the invert and smoothing step also play a big role in the user score.

By using the features from the invert and smoothing step, combined with the features from the thresholding step, the user score prediction will become more justified. The current dataset has too few samples for this to be viable. When also using the features for the invert and smooth steps the number of dimensions increase to at least 3 times the current number of features used.

Using a test set

To get an idea of a Machine Learning model's performance during research, cross validation is used. In the simplest form, the dataset is divided into a training and validation set. This allows us to assess how a model will generalize to unseen data and detect overfitting or if the model is biased.

Besides the training and validation set, a test set can also be used. This is a set of data that at the beginning of research has been separated from the dataset. The test set is only used to assess the performance of a finalized model to draw concrete conclusions from.

The conclusions in this paper are based on the evaluation metrics applied to the model's predictions on the validation set. To get the most reliable conclusions, we used n-fold cross validation to get an idea of how well the model performs on the entire dataset.

Ranking instead of classification Insert link to paper

The original goal was to find the best threshold method out of 6 thresholding algorithms. This is more of a ranking problem than predicting the best one. Learning to Rank (LTR) algorithms are supervised Machine Learning algorithms that can do this.

The main difference between the two is the following:

- Machine Learning algorithms try to solve a prediction problem on a single item at a time. This can be in the form of a classification or regression model.
- Learning to Rank solves a ranking problem on a set of items. It tries to come up with an optimal order of those items. It cares more about the order of the items, not as much on the individual score. Training data consists of inputs, outputs and a relevance score indication how relevant the two are to each other.

Unfortunately, in this dataset there are very few cases where different thresholding algorithms were used on the same image. For this to work more thresholding methods need to be applied and scored on the same image.

It would make the most sense to use the feature values that were calculated in the smoothing step, with the 6 threshold method and use the user score for each threshold algorithm as a relevance score.

Bibliography

1) Woehrle, G. H. et al. (2005). *Analysis of Nanoparticle Transmission Electron Microscopy Data Using a Public-Domain Image-Processing Program, Image.*
2) Li, F. et al. (2007). *High content image analysis for human H4 neuroglioma cells exposed to CuO nanoparticles.*
3) VSParticle. (2020). *Publications.* Retrieved December1, 2020 from www.vsparticle.com.
4) Wang, Z. L., Lee, J. L. (2008). *Electron Microscopy Techniques for Imaging and Analysis of Nanoparticles – Chapter 9.*
5) Mitchell, H. B. (2010). *Image Thresholding.* Springer, Berlin, Heidelberg.
6) Yen, J., Chang, F., Chang, S. (1995). *A new criterion for automatic multilevel thresholding.*
7) Otsu, N. (1979). *A Threshold Selection Method from Gray-Level Histograms.*
8) King, S., Jarvie, H. et al. (2019). *Nanoparticle.* Encyclopaedia Britannica.
9) LaValley, M. P. (2008). *Logistic Regression*.
10) Sharma, A. (2020). *A Simple Analogy to Explain Decision Tree vs. Random Forest*.
11) Morgan, S., & Teachman, J. (1988). *Logistic Regression: Description, Examples, and Comparisons.*
12) Myles, A. J. (2004). *An introduction to decision tree modeling*.
13) Sehra, C. (2018). *Decision Trees Explained Easily.*
14) Noriega, L. (2005). *Multilayer Perceptron Tutorial.*
15) Ramchoun, H. (2016). *Multilayer Perceptron: Architecture Optimization and Training*.
16) Tiwari, P. (2017). *Performance Evaluation of Lazy, Decision Tree Classifier and Multilayer Perceptron on Traffic Accident Analysis*.
17) Dencelin, L. (2016). *Analysis of multilayer perceptron machine learning approach in classifying protein secondary structures*.
18) Sokolova, M. (2009). *A systematic analysis of performance measures for classification tasks*.
19) Longadge, R. (2013). *Class Imbalance Problem in Data Mining: Review.*
20) Midi, H. (2010). *Collinearity diagnostics of binary logistic regression model.* [1]

[1]

Appendix