Project Report - Project course in mathematical and statistical modelling (MMA520/MSA520)

Konrad Pohl, Calvin Smith, Harald Westling

Chalmers/University of Gothenburg

February 22, 2023

Abstract

This project report consists of a student groups work with a problem provided by Volvo Cars. The considered problem is an open set recognition problem for neural networks (NN). Volvo has a NN that is supposed to get data from two distributions; *In-distribution* and *Out-of-distribution.* The goal is to separate correctly classified data points from the *In-distribution, Inliers* from the joint set of points from *Out-of-distribution, (OOD)* and *Missclassified In-distribution, Outliers.* The predictors for this classification problem is the softmax output from the NN. The problem is attended with the three methods: Density-based spatial clustering of applications with noise (DBSCAN), Mixed discriminant analysis (MDA) and Isolation forest (IF). The findings are that identifying OOD data points is hard to do with the utilized methods. However, the methods performs well in separating between *Inliers* and *Missclassified In-distribution.* It is concluded that the latter separation is possible due to well defined limits in feature space and that the first problem needs to be attended with methods that does not directly use distance in feature space as means for classification. It is proposed that further work investigates methods that use deeper layers from the NN or mappings of the feature space to enable identification of OOD data points.

Contents

1 Introduction/Backround

Introduction

The open-set recognition problem with neural networks is a subject that has got an increased amount of attention in the last couple years.[2] A classification neural network is usually trained on carefully selected and annotated data, with a specific set on predefined classes in mind. And then ideally validated on datasets that share a lot of similarities with the trained datset, while having a hard time dealing with data from another distribution then the one trained on. This is called close-set recognition with neural networks.
For autonomous cars that rely on neural networks to navigate the closes-set approach might could lead to accidents related to the vehicle misinterpreting an object. Thus making sure that there exists a robust algorithms in place to ensure that out of distribution objects are correctly classified as such, and not acted on.

Volvo's background description

"In this project, neural networks have been trained on a specific dataset with 10 different classes. This is called the inlier dataset. This network has then been used to do inference on two types of data: in-distribution data that comes from the same distribution as the data used to train the network, and out-of-distribution data that comes from another distribution. The in-distribution and out-of-distribution data show different classes of objects, and thus no predictions for the out-of-distribution data are correct.

The project is concerned with performing modelling and statistical analysis on the activation patterns in the network to determine whether the activation pattern can be used to discriminate in-distribution datapoints from out-of-distribution datapoints.

The activation pattern used is the softmax output of the neural network. This layer represents the belief of the network that the input belongs to each of the 10 classes (note: while some literature would describe this output as probabilities this is erroneous). It is worthwhile to note that the task is NOT to separate the different classes of the inlier and the outlier dataset, as this is a trivial matter. When separating the two datasets, the model or technique used to separate the two classes can either be a single model for all 10 classes or one model for each class. The first is a harder problem and the second is a bit easier and generally shows much greater results. Which approach is chosen doesn't matter, only the final score is of interest."

Volvo's task description

Separate the inlier and the outlier dataset using some kind of algorithm, model or statistical method. How this separation is done is by purpose left open-ended, and creative ways of solving this problem are encouraged! The method used should, for each activation vector, output a single scalar value called the anomaly score. The higher the anomaly score, the more likely that the activation vector belongs to the out-of-distribution set. This anomaly score will be used to determine how well the method separates the two datasets. The metric used to determine this will be obtained by using the Receiver-Operating Characteristic Curve (ROC Curve), and the score will be the area under this curve (called the AUROC score).

Note: In the in-distribution dataset, not all datapoints are classified as the correct class. For the purposes of this project, misclassified datapoints should be classed as outliers and hence be given a high anomaly score. If you like, it could be very interesting to test how well your method can distinguish whether a datapoint is misclassified or whether it is from the out-of-distribution dataset.

Summary of the task

Data points may come from two sets of data: in-distribution and out-of-distribution. After the NN has classified data points there are three disjoint sets of data:

- **Inliers**: Correctly classified observations from the in-distribution set.
- **Missclassified**: Missclassified observations from the in-distribution set.
- **Out-of-distribution (OOD)**: Observations from the out-of-distributions set.

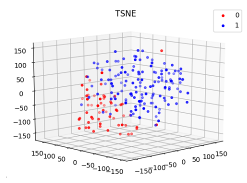
The main objective is to separate the **Inliers** from the joint set $\{$**missclassified** $\cup$ **OOD**$\}$. Observations from this joint set will henceforth be referred to as **Outliers**.

2 Theory/Methods

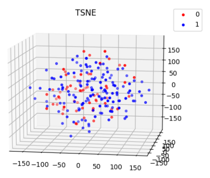
This section describes the theory of the methods utilized in the project.

2.1 Data

The data consists of 10 different classes, 0 to 9. Each row of the data represents an observation and each column represents the neural networks belief that the observation belongs to that class. We have access to 10.000 observations from the in-distribution set and 10.000 observations from the out-of-distribution set. The in-distribution set contains approximately 20% miss-classified data the out-of-distribution data observations are all miss-classified. In figure 1 a subset of the elements belonging to the neural network class 0 are shown after applying a t-distributed stochastic neighbor embedding[6](t-sne) transform with perplexity 20. In figure 1a the blue elements, class 1 in the figure, are the inliers belonging to the neural network class 0, the red elements, class 0 in the figure, are the elements, from the same distribution, that were misclassified as belonging to the neural network class 0. In figure 1b the blue elements, class 1 in the figure, are the inliers belonging to the neural network class 0, the red elements, class 0 in the figure, are the elements that came from the OOD but were classified by the neural network to belong to the class 0.



(a) A subset of the elements belonging to class 0 after a t-sne transform. The blue data points are inliers and the red are data points from OOD classified as belonging to the neural network class 0.



(b) A subset of the elements belonging to class 0 after a t-sne transform. The blue data points are inliers and the red are data points misclassified as belonging neural network class 0

Figure 1: A subset of the elements belonging to class 0 after a t-sne transform with perplexity 20.

2.2 AUROC

The area under the curve of the receiver operating characteristic (AUROC) score is an evaluation metric often used in binary classification problems. As the name suggests the metric is dependent on the receiver operating characteristic (ROC) curve. The ROC curve is created by plotting the True positive rate (TPR), on the y-axis, versus the False positive rate (FPR), on the x-axis, by iterating over the class probability and set a new classification cutoff, in each iteration, such that each class that get a probability score higher than the cutoff limit gets classified as belonging to the positive defined class and the elements with lower score as belonging to the negative defined class. And calculating the area under the ROC curve results in the AUROC score, which is defined between the values [0,1].

2.2.1 Manual AUROC

Since the ROC cuve needs elements with a bias or probability score associated with each element. In cases where the implemented method doesn't produce such a bias score a manual ROC curve can be constructed by constructing a confusion matrix and calculating the TPR as TPR $= \frac{TP}{TP+FN}$ and the FPR as FPR $= \frac{FP}{FP+TN}$ and constructing a curve through interpolation. This was the main evaluation metric used in this project. It should be noted that the manual AUROC score is in general lower than the proper AUROC score.

2.3 DBSCAN

DBSCAN (Density Based Spacial Clustering of Applications with Noise)[5] is an unsupervised clustering algorithm. It is a popular and well documented method when it comes to clustering and classification problems. The algorithm primarily has three main advantages compared to other clustering algorithms:

- You do not need to specify the number of clusters to be found.
- The algorithm can find clusters of any shape.
- The algorithm is able to detect outliers.

The main idea behind the algorithm is to identify dense regions of the data, by assigning a point to a cluster if it is "close" to many other points from that cluster. There are two main parameters in the **DBSCAN** algorithm, *eps* (epsilon) and *minPts* (minimum number of poins). The parameter *eps* defines the radius of the neighbourhood around a point x. Two points are considered to be neighbours if the distance between them is less than or equal to *eps*. The parameter *minPts* is defined as the minimum number of points required to define a cluster.

Based on these two parameters, the algorithm classifies points in three different categories:

- **Core point**: A point is classified as a core point if there are at least *minPts* number of points within its radius *eps* .
- **Border point**: A point is classified as a border point if it is within the radius of a core point but there are less than *minPts* observations within its own radius *eps* .
- **Outlier**: A point is marked as an outlier if it is neither a core point or a border point.

Figure 2 illustrates how the DBSCAN algorithm classifies points when $minPts = 4$. The red points are classified as **core points** since they all have at least 4 other points within their radius. The yellow points are classified as **border points** because they are reachable from a core point but have less than 4 points within their radius. Lastly, the blue point is classified as an **outlier** since it is clearly neither a core point nor a border point.

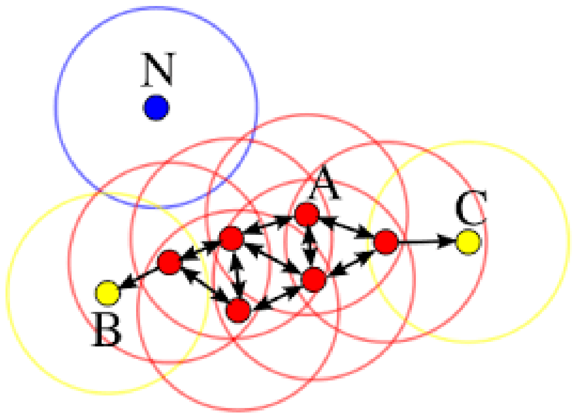


Figure 2: DBSCAN Clustering.

The algorithm works in the following steps:

1. Determine the parameters *eps* and *minPts*.

2. Randomly choose a starting point and determine its surrounding neighborhood using the parameter *eps*. If there are at least *minPts* number of points within the neighborhood the clustering formation begins. Otherwise the point is marked as an outlier. When the clustering formation begins, all the points within the neighborhood of the initial point become part of a cluster, and if these points are also core points then their neigborhood points are also added to cluster, and so on.

3. Randomly choose a new starting point that has not yet been visited and apply step 2.

4. Continue until all the points have been visited.

The algorithm assigns each observation to a specific cluster or as an outlier. The number of clusters produced by the algorithm is determined by the underlying "shape" of the data, but is also highly influenced by the choice of parameters *minPts* and *eps* .

The main challenge when using the DBSCAN algorithm is determining the optimal values of the parameters *minPts* and *eps*. The choice of these parameters are highly specific to the dataset you are working with. If k is the dimension of the data, a common recommendation is to choose *minPts* to be at least k or higher. If in addition you have a very large dataset setting k even higher can produce better results. Futhermore, there does not seem to be an obvious way of choosing the optimal value *eps*. The most common distance measure used for *eps* is euclidian distance. Thus, determining *eps* is also highly influenced by the characteristics of the data and its features [5].

2.4 Isolation forest

The Isolation forest[4] is an unsupervised algorithm that is used for anomaly detection. It works on the principle that anomaly's, or outliers, differ in feature space from the dataset. Consider the data $X = \{x_1, .., x_n\}$, and let $X' \subset X$. By creating a fully grown binary tree where each node, T, has either zero child nodes, an external node, or two daughter nodes (T_l, T_r) and one 'test'. The tree is constructed by recursively dividing X' by performing a test at node T were the test consists of randomly selecting an attribute, or feature, q and a splitting value p, such that the test $q_i < p$ divides the data points into T_l and T_r, where q_i are attribute values for each element. The algorithm ends once $|X'| = 1$ or all data in X' have the same value.

The outliers can then be identified using a anomaly score which is calculated based on the amount of edges needed in order to isolation each data point. The fewer the edges the more isolated, and the more abnormal. The score is calculated using the path length $h(x)$, with $x \in X$, which is defined as the amount of edges x needs to traverse from the root node in order to get to a external node. The anomaly score is calculated using the equation

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{1}$$

Where

$$c(n) = 2H(n-1) - 2(n-1)/n \tag{2}$$

where $c(n)$ is the average length of an unsuccessful search in a Binary search tree and n is the number of elements in a dataset. $H(i)$ can be estimated as $ln(i) + 0.5772156649$ and $E(h(x))$ is the average of $h(x)$ from a collection of trees.

Using scoring function $s(x, n)$ for each element in the data an outlier will be an element with a corresponding score close to 1, if an element scores a $s(x, n)$ value much smaller than 0.5 then they can be considered as inlier elements. If all elements score a $s(x, n)$ value close to 0.5 then the entire sample does not contain any distinct anomalies.

2.5 Mixture discriminant analysis (MDA)

MDA is a supervised algorithm that is suitable when classes are distributed over multiple Gaussian clusters within the same class. [3]

The algorithm assumes densities on the form: nequation P(X$|G = k) = \sum_{r=1}^{R_k} \pi_{kr}\phi(X; \mu_{kr}, \mathbf{\Sigma})$

And draws decision boundaries where the probability of belonging to class k is largest. The boundaries are found through the Expectation-Maximization algorithm.

Note that the algorithm

- Uses the same covariance matrix $\mathbf{\Sigma}$ for all clusters.
- Allows for different amounts of subclasses/clusters for each class R_k

3 Implementation

The methods have been implemented separately on each class $k = 0, ..., 9$ of the data. This is in contrast to implementing the methods directly over all classes.

3.1 Training, validation and testing data

The data has been divided into training, validation and testing sets. The in-distribution set contains a total of 10.000 observations equally distributed between classes, i.e 1000 observations per class. The proportion of *inliers* and *missclassified* is 80% and 20% respectively. The out-of-distribution set also contains 10.000 observations but are not equally distributed between classes. The training, validation and testing sets have all been sampled with respect to the class proportions of both the in-distribution and the out-of-distribution set. The validation and testing sets have been created in two different versions, a large version and a small version. To get accurate results, all validation and testing data consists of 50% *inliers* and 50% *outliers*.
The following sets have been constructed:

- **Training**: 12000 observations of which 6000 are from the in-distribution set (80% *inliers* and 20% *missclassified*) with 600 observations per class and 6000 observations from the out-of-distribution set, sampled proportionally to each class.
- **Large validation/Large testing**: The largest possible sets containing 50% *inliers* and 50% *outliers*. 3200 observations.
- **Small validation/small testing**: Subsets of the larger sets. 400 observations with 40 observations per class and exactly 50% *inliers* and 50% *outliers* per class.

3.2 DBSCAN

The DBSCAN method has been implemented in R using the **dbscan**-package.

3.2.1 Choosing *eps* and *minPts*

The main challenge is to determine the optimal values of *eps* and *minPts*. This was done using a grid search over possible values for the parameters and choosing the parameters for each class that yields the highest AUROC value based on predictions on the **validation**-sets.

For each combination of *eps* and *minPts*:

1. For each class $k = 0, ..., 9$,let X^k_{train} and X^k_{valid} be the subsets of the training and validation data respectively only containing class k.
2. Apply the DBSCAN algorithm on X^k_{train} , obtaining a clustering object with each observation assigned to a cluster or marked as an outlier.

3. Using the clustering object, make predictions on the X^k_{valid} . Each observation here is marked as 1 (*inlier*) and 0 (*outlier*)
4. Obtain a confusion matrix and calculate the AUROC-score.
5. Contine until all combinations of *eps* and *minPts* have been tested.
6. Choose the combination of **eps** and **minPts** that produces the highest AUROC value for each class.

This method yields the combination of *eps* and *minPts* that yield the highest AUROC value for each class. This combination is then stored and used for the final testing of DBSCAN's performance on the **testing**-sets.

3.2.2 Training data

To further analyze the performance of DBSCAN with respect to the "robustness" of the method against outliers, three different **training**-sets have been constructed:

1. The full training set $\boldsymbol{X}_{full}$: Contains 12000 observations with 50% in-distribution points (80% Inliers, 20% Missclassified) and 50% out-of-distribution.
2. Subset of full training set $\boldsymbol{X}_{in,miss}$: Only containing Inliers and Missclassified. 6000 observations (80/20).
3. Subset of full training set $\boldsymbol{X}_{in}$: Only containing Inliers. 4768 observations.

This means that for each training set:

1. Apply DBSCAN-algorithm.
2. Predict on the small and large validation sets.
3. Obtain best AUROC-score and optimal **eps** and **minPts**.
4. Apply DBSCAN on the training set again with the optimal parameters, predict on the small and large test sets.
5. Final result. Per class AUROC-score and confusion matrix.

3.3 Isolation forest

The isolation forest algorithm was implemented using the scikit-learn package in python. This algorithm was implemented using the 10 subset implementation.

1. Apply the algorithm on the training set and store the anomaly score, defined in equation (1), for each element.
2. Perform a grid search to find the threshold value which results in the highest manual AUROC score where all elements with an anomaly score higher than the threshold value are classified as out of distribution or as an outlier.

3. Store the best performing threshold value and apply the algorithm on test data and classify the elements according to the threshold value.

3.4 MDA

MDA was implemented with the `R`-function `mda` over the following settings:

1. Classifying between Inliers and Outliers with 10 subset implementation.
2. Classifying between Inliers and Missclassified with 10 subset implementation.

See appendix for complete `R`-code.

4 Results

The following sections concludes results from the implemented methods over Large data sets and small data sets. The last subsection concludes results for separation of inliers and missclassified.
In table 1 the manual AUROC scores are shown for each implemented algorithm, with the bold faced meaning the highest score on the dataset. We decided to report the proper AUROC score for the Isolation Forest implementation in order to illustrate the possible discrepancy between the two evaluation metrics. In table 2 the manual AUROC value on the test data is shown. The bold faced numbers are the manual AUROC scores that we would have got if we chose the best scoring algorithm for each class on the validation data.
In the tables 3 and 4 the manual AUROC scores are shown for small validation dataset and small test dataset. The bold faced numbers in the validaion table illustrate the best performing algorithm for each class, while bold faced numbers in the test table illustrate the manual AUROC values that would have been achieved if we chose the best scoring model on the validation set.
In table 5 the average manual AUROC score is show when the algorithms are tested on a dataset containing only misclasified and inlier elements.

4.1 Large sets

	0	1	2	3	4	5	6	7	8	9	Total
DBSCAN	**0.81**	**0.91**	0.72	**0.73**	**0.74**	**0.76**	**0.79**	**0.86**	**0.8**	**0.76**	**0.78**
MDA	0.78	0.84	**0.75**	0.65	0.70	0.69	0.76	0.80	0.75	0.71	0.74
IF (man. auroc)	0.79	0.86	0.71	0.71	0.73	0.73	0.71	0.83	0.73	0.71	0.75
IF (auto. auroc)	0.88	0.95	0.82	0.82	0.83	0.81	0.84	0.92	0.85	0.82	0.85

Table 1: Validation AUROC values - Large set

	0	1	2	3	4	5	6	7	8	9	Total
DBSCAN	0.79	**0.84**	0.75	0.72	0.73	**0.73**	**0.75**	**0.81**	**0.83**	**0.80**	**0.78**
MDA	0.74	0.84	**0.78**	0.72	**0.74**	0.73	0.73	0.78	0.77	0.76	0.76
IF (Manual auroc)	**0.81**	0.82	0.75	**0.74**	0.73	0.73	0.72	0.81	0.78	0.72	0.76
IF (Proper auroc)	0.88	0.94	0.84	0.84	0.84	0.83	0.86	0.9	0.91	0.85	0.87

Table 2: Test AUROC values - Large set

4.2 Small sets

	0	1	2	3	4	5	6	7	8	9	Total
DBSCAN	**0.92**	**0.92**	**0.82**	**0.8**	0.8	0.82	**0.95**	**0.82**	0.77	0.8	**0.84**
MDA	0.78	0.85	0.73	0.72	0.77	0.72	**0.95**	0.80	0.78	**0.81**	0.79
IF (Manual auroc)	0.88	0.83	0.78	0.75	**0.83**	**0.83**	0.83	0.73	**0.83**	0.73	0.8
IF (Proper auroc)	0.94	0.96	0.86	0.83	0.91	0.88	0.97	0.78	0.89	0.84	0.89

Table 3: Validation AUROC values - Small set

	0	1	2	3	4	5	6	7	8	9	Total
DBSCAN	0.67	0.82	0.7	0.65	0.70	0.72	**0.85**	**0.85**	**0.85**	0.77	**0.76**
MDA	**0.78**	**0.88**	0.70	**0.69**	**0.73**	0.72	0.85	0.68	0.79	**0.78**	0.76
IF (Manual auroc)	0.75	0.85	**0.75**	0.68	0.7	**0.77**	0.7	0.8	0.78	0.73	0.75
IF (Proper auroc)	0.82	0.93	0.77	0.79	0.74	0.85	0.89	0.86	0.92	0.82	0.84

Table 4: Test AUROC values - Small set

4.3 Separation Inliers - misclassified

	DBSCAN	MDA	Iso. Forest
Large set	1.0	0.988	0.861
Small set	1.0	0.987	0.88

Table 5: The manual AUROC score on a test set containing only inliers and misclassified elements.

5 Discussions/Conclusions

The main finding is that Inliers and Missclassified are rather easy to separate. Both DBSCAN and MDA classifies nearly all data points correctly. OOD on the other hand seems much harder to separate, and the methods used are inadequate for application in autonomous cars. At least it seems a bit unsafe with a car that may mistake its surroundings as often as 20% of the time.

All three methods perform quite similar results. This is probably because they are quite similar in the sense that they are somewhat straight-forward in using simple geometry, distances, in the feature space as grounds for classification.

There was some uncertainty about the influence of the proportion of OOD data points. We conclude that this proportion does not matter so much. Although, it is important for future problem formulations to know the value of correctly classified data points. As mentioned before, 20% wrongly classified data points seems much with regard to autonomous cars, but that is just a guess from the authors.

5.1 Proposed further work

This section will contain suggestions for further work with the problem.

It is clear that all the algorithms that we have implemented have with relative ease been able to separate the misclassified inlier from the correctly classified inliers. But they have all had problems when it comes to separating the inliers from the OOD data points. After working with the problem we recommend that further work covers:

- Constructing a statistical test in the penultimate layer of the neural network, before the softmax layer, in order to separate the inliers and misclassified from the OOD elements based on the magnitude vectors.[1]
- Prioritize investigation of possible mappings that could be used to separate OOD-data points from the inliers. This separation was hard to find satisfying results for.
- Suggestively, first separate the Missclassified data, then separate inliers and OOD.
- QDA is similar to MDA, but is more flexible in its geometry. This could make it an interesting method.

6 Acknowledgements

Thanks to Michael Fjellgren for supervising this project and giving valuable insights from the working life of a data scientist.
Thanks to Alexei Heintz for having the course, it has been a very valuable experience.
Thanks to Felix Held for providing insights on the considered problem.

References

[1] Bendale, A. and Boult, T. E. (2015). Towards open set deep networks. *CoRR*, abs/1511.06233.

[2] Geng, C., Huang, S., and Chen, S. (2018). Recent advances in open set recognition: A survey. *CoRR*, abs/1811.08581.

[3] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning : data mining, inference, and prediction.* Springer series in statistics. Springer.

[4] Liu, F. T., Ting, K., and Zhou, Z.-H. (2009). Isolation forest. pages 413 – 422.

[5] Martin Ester, Hans-Peter Kriegel, J. S. X. X. (1996). A density-based algorithm for discovering clusters.

[6] van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Appendix

6.1 DBSCAN additional results

Classes	0	1	2	3	4	5	6	7	8	9	AVG
X_{large}	0.794	0.845	0.756	0.723	0.728	0.735	0.753	0.810	0.830	0.804	**0.78**
X_{small}	0.675	0.850	0.675	0.625	0.650	0.725	0.850	0.850	0.850	0.775	**0.75**

Table 6: AUROC values when training on X_{full} and predicting on X_{large} and X_{small}.

		Actual	
		0	1
Pred	0	1280	376
	1	336	1209

Table 7: Confusion matrix for predictions on X_{large}, training X_{full}. Acc = 0.77.

		Actual	
		0	1
Pred	0	149	48
	1	51	152

Table 8: Confusion matrix for predictions on X_{small}, training X_{full}. Acc = 0.75.

	0	1	2	3	4	5	6	7	8	9	TOT
nr_miss	57	32	54	90	32	61	26	33	17	13	415
correct_miss	57	32	54	90	32	61	26	33	17	13	415
acc_miss	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**
nr_ood	61	44	159	168	147	120	201	69	98	134	1201
correct_ood	33	30	128	99	86	75	170	48	88	108	865
acc_ood	0.54	0.68	0.81	0.59	0.59	0.62	0.85	0.70	**0.90**	0.81	**0.72**

Table 9: Table showing how DBSCAN classifies *Missclassified* and *OOD* observations. Based on training with X_{full} and predicting on X_{large}.

	0	1	2	3	4	5	6	7	8	9	TOT
nr_miss	9	8	3	10	1	4	1	9	3	0	48.00
correct_miss	9	8	3	10	1	4	1	9	3	0	48
acc_miss	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00		**1.00**
nr_ood	11	12	17	10	19	16	19	11	17	20	152
correct_ood	2	8	6	7	11	13	17	7	14	16	101
acc_ood	0.18	0.67	0.35	0.70	0.58	0.81	**0.89**	0.64	0.82	0.80	**0.66**

Table 10: Table showing how DBSCAN classifies *Missclassified* and *OOD* observations. Based on training with X_{full} and predicting on X_{small}.

Classes	0	1	2	3	4	5	6	7	8	9	AVG
X_{large}	0.792	0.829	0.744	0.723	0.728	0.741	0.769	0.815	0.851	0.8	**0.78**
X_{small}	0.675	0.850	0.7	0.625	0.7	0.725	0.850	0.825	0.850	0.775	**0.75**

Table 11: AUROC values when training on $X_{in,miss}$ and predicting on X_{large}, X_{small}.

		Actual 0	Actual 1
Pred	0	1265	351
Pred	1	351	1235

Table 12: Confusion matrix for predictions on X_{large} training $X_{in,miss}$. Acc = 0.78.

		Actual 0	Actual 1
Pred	0	154	51
Pred	1	46	149

Table 13: Confusion matrix for predictions on X_{small}, training $X_{in,miss}$. Acc = 0.757.

	0	1	2	3	4	5	6	7	8	9	TOT
nr_miss	57	32	54	90	32	61	26	33	17	13	415
correct_miss	57	32	54	90	32	61	26	33	17	13	415
acc_miss	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**
nr_ood	61	44	159	168	147	120	201	69	98	134	1201
correct_ood	30	26	129	110	92	76	142	54	82	109	850
acc_ood	0.49	0.59	0.81	0.65	0.63	0.63	0.71	0.78	**0.84**	0.81	**0.71**

Table 14: Table showing how DBSCAN classifies $Missclassified$ and OOD observations. Based on training with $X_{in.miss}$ and predicting on X_{large}.

	0	1	2	3	4	5	6	7	8	9	TOT
nr_miss	9	8	3	10	1	4	1	9	3	0	48
correct_miss	9	8	3	9	1	4	1	8	3	0	46
acc_miss	1.00	1.00	1.00	0.90	1.00	1.00	1.00	0.89	1.00		**0.96**
nr_ood	11	12	17	10	19	16	19	11	17	20	152
correct_ood	2	10	7	8	13	13	17	8	14	16	108
acc_ood	0.18	0.83	0.41	0.80	0.68	0.81	**0.89**	0.73	0.82	0.80	**0.71**

Table 15: Table showing how DBSCAN classifies $Missclassified$ and OOD observations. Based on training with $X_{in.miss}$ and predicting on X_{small}.

Classes	0	1	2	3	4	5	6	7	8	9	AVG
X_{large}	0.789	0.828	0.741	0.743	0.734	0.735	0.769	0.815	0.851	0.79	**0.78**
X_{small}	0.675	0.825	0.7	0.650	0.700	0.725	0.850	0.850	0.850	0.775	**0.76**

Table 16: AUROC values when training on X_{in} and predicting on X_{large}, X_{small}.

		Actual	
		0	1
Pred	0	1171	327
	1	445	1258

Table 17: Confusion matrix when predicting on X_{large}, training X_{in} .Acc = 0.76

		Actual	
		0	1
Pred	0	155	51
	1	45	149

Table 18: Confusion matrix when predicting on X_{small}, training X_{in} .Acc = 0.76

	0	1	2	3	4	5	6	7	8	9	TOT
nr_miss	57	32	54	90	32	61	26	33	17	13	415
correct_miss	57	32	54	90	32	61	26	33	17	13	415
acc_miss	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**
nr_ood	61	44	159	168	147	120	201	69	98	134	1201
correct_ood	35	26	116	101	87	62	142	54	82	109	814
acc_ood	0.57	0.59	0.73	0.60	0.59	0.52	0.71	0.78	**0.84**	0.81	**0.68**

Table 19: Table showing how DBSCAN classifies $Missclassified$ and OOD observations. Based on training with X_{in} and predicting on X_{large}.

	0	1	2	3	4	5	6	7	8	9	TOT
nr_miss	9	8	3	10	1	4	1	9	3	0	48
correct_miss	9	8	3	10	1	4	1	9	3	0	48
acc_miss	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00		**1.00**
nr_ood	11	12	17	10	19	16	19	11	17	20	152
correct_ood	2	9	7	8	13	13	17	8	14	16	107
acc_ood	0.18	0.75	0.41	0.80	0.68	0.81	**0.89**	0.73	0.82	0.80	**0.70**

Table 20: Table showing how DBSCAN classifies $Missclassified$ and OOD observations. Based on training with X_{in} and predicting on X_{small}.

7 Code used to implement the algorithms

7.1 Creating data sets

```
# The following code is for splitting the data into training, validation and testing sets.

library(tidyverse)
library(dplyr)

### Data Preparation ####
classes <- seq(0,9)

# in-distribution data
one_inlier <- read.csv("network_1_inlier_activation_vectors.csv", sep = "," , header = F)
```

```
colnames(one_inlier) <- classes
# out-of-distribution data
one_outlier <-  read.csv("network_1_outlier_activation_vectors.csv", sep = "," , header = F
colnames(one_outlier) <- classes

# true classes in-distribution
true_inlier <- read.csv("inlier_true_labels.csv", sep = ",", header = F)
colnames(true_inlier) <- "class"

one_inlier$class <- true_inlier$class

#### classifying out-of-distribution ####
pred_classes <- numeric(10000)
for (i in 1:10000){
  pred_classes[i] <- which(one_outlier[i,1:10] == max(one_outlier[i,1:10]))
}
pred_classes <- pred_classes - 1

one_outlier$class <- pred_classes

#### Sampling in-distribution data ####

## classifying inliers and missclassified
pred_classes <- numeric(10000)
for (i in 1:10000){
  pred_classes[i] <- which(one_inlier[i,1:10] == max(one_inlier[i,1:10]))
}
pred_classes <- pred_classes - 1

c <- pred_classes == true_inlier

one_inlier$inlier <- 1
one_inlier$inlier[c != TRUE] <- 0

one_inlier$type <- "x_in"
one_inlier$type[c != TRUE] <- "x_miss"

index <- 1:10000
one_inlier$index <- index
one_inlier <- one_inlier %>% group_by(class)
train_inlier <- sample_n(one_inlier,600)
valid_inlier <- sample_n(one_inlier[-train_inlier$index,],200)
test_inlier <- one_inlier[-c(train_inlier$index,valid_inlier$index),]


#### Sampling out-of-distribution data ####
```

```
one_outlier$inlier <- 0
one_outlier$type <- "x_out"
one_outlier$index <- index
one_outlier <- one_outlier %>% group_by(class)
train_outlier <- sample_frac(one_outlier,0.6)
valid_outlier <- sample_frac(one_outlier[-train_outlier$index,],0.3)
test_outlier <- sample_frac(one_outlier[-c(train_outlier$index,valid_outlier$index),],3/7)


### FINAL DATASETS ####

X_train <- rbind(train_inlier,train_outlier)
X_valid <- rbind(valid_inlier,valid_outlier)
X_test <- rbind(test_inlier,test_outlier)

x_valid <- X_valid %>% group_by(class,inlier) %>% sample_n(20)
x_test <- X_test %>% group_by(class,inlier) %>% sample_n(20)

write.csv(X_train,"X_train.csv")
write.csv(X_valid,"X_valid.csv")
write.csv(X_test,"X_test.csv")

write.csv(x_valid,"small_valid.csv")
write.csv(x_test,"small_test.csv")
```

7.2 Python code for Isolation forest

```
import numpy as np
import sklearn
import matplotlib.pyplot as plt
import pandas as pandas


def LoadData():

    Train001=pandas.read_csv('NewData/X_train.csv',sep=',',header=0)
    Pen=pandas.read_csv('activations/network_1_inlier_activation_vectors_penultimate.csv',se
    #Val001=pandas.read_csv('NewData/X_valid.csv',sep=',',header=0)
    #Val001=pandas.read_csv('NewData/X_test.csv',sep=',',header=0)
    Val001=pandas.read_csv('NewData/small_valid.csv',sep=',',header=0)


    return(Train001,Val001)
```

```
def Init():
    [Net1In,Val]=LoadData()
    #[Combdf,Comblab]=CombineDf(Net1In,Net1Out)
    #IsolForest(Net1In)
    for i in range(10):
        cl=i
        [nLabT,nValT]=SeperateDists(Net1In,cl,True)
        [nLab,nVal]=SeperateDists(Val,cl,True)



        print(cl)
        IsolForestTrain(nValT,nLabT,nVal,nLab)


    #TSN(nVal,nLab)
    #TSN(Combdf,Comblab)

    #PCA(nValT,nLabT,'Net1')
    #TSN(nValT,nLabT)




def SeperateDists(X,c,train):
    inliers=X.loc[X['type']=='x_in']
    miss=X.loc[X['type']=='x_miss']
    out=X.loc[X['type']=='x_out']
    InClass=inliers.loc[inliers['class']==c]
    MissClass=miss.loc[miss['class']==c]
    OutClass=out.loc[out['class']==c]
    InClass=DropDfCols(InClass)
    MissClass=DropDfCols(MissClass)
    OutClass=DropDfCols(OutClass)
    if(train):
        #Outliers=MissClass.append(OutClass)
        AllElements=InClass.append(MissClass)
    else:
        Outliers=MissClass.append(OutClass)
        AllElements=InClass.append(Outliers)

    LabDf=AllElements['inlier']
    AllElements.drop(AllElements.columns[10],axis=1,inplace=True)

    return(LabDf.to_numpy(),AllElements.to_numpy())
```

```
def DropDfCols(X):
    X.drop(X.columns[14],axis=1,inplace=True)
    X.drop(X.columns[13],axis=1,inplace=True)
    X.drop(X.columns[11],axis=1,inplace=True)
    X.drop(X.columns[0],axis=1,inplace=True)
    return X


def ConfMatrix(RLab,Lab):
   # RLab predicted labels
   #Lab true labels
    Tp=0;
    Tn=0;
    Fn=0;
    Fp=0;
    for i in range(len(Lab)):
        if(RLab[i]==1 and Lab[i]==1):
            Tp+=1
        if(RLab[i]==0 and Lab[i]==1):
            Fn+=1
        if(RLab[i]==0 and Lab[i]==0):
            Tn+=1
        if(RLab[i]==1 and Lab[i]==0):
            Fp+=1
    #print('True pos:',Tp)
    #print('True neg:',Tn)
    #print('False neg:',Fn)
    #print('False pos:',Fp)
    return ManRoc(Tp,Fp,Tn,Fn)

def SVMTrain(X,lab,XV,labV):
    from sklearn.svm import OneClassSVM
    clf = OneClassSVM()
    clf.fit(X)
    sam=clf.decision_function(X)
    idx=np.argpartition(sam, 10)
    val=Search(sam,lab)
    clf = OneClassSVM()
    clf.fit(XV)
    sam=clf.decision_function(XV)
    MaxSam=np.amax(sam)
    #scikit roc-auc-score classifies elements as positive if large y_target score. We have t
    AucVal=-MaxSam+sam
    #LabC2=clf.predict(XV)
    #LabC22=(LabC2+1)/2

    LabC=Reclass(val,sam)

    #print(LabC)

    #[Tru,TN,FN,TP,FP]=Eval(labV,LabC)
    Metrics(LabC,AucVal,labV)
    ConfMatrix(LabC,labV)
    #Metrics(LabC22,sam,labV)
    #ConfMatrix(LabC22,labV)


def IsolForestTrain(X,lab,XV,labV):
    from sklearn.ensemble import IsolationForest
    clf = IsolationForest(random_state=0)
    clf.fit(X)
    sam=clf.decision_function(X)
    val=Search(sam,lab)
    print(val)
    #print(sam)
    #print(lab)
    clf = IsolationForest(random_state=0)
    clf.fit(XV)
    sam=clf.decision_function(XV)
    MaxSam=np.amax(sam)

    #scikit roc-auc-score classifies elements as positive if large y_target score. We have t
    AucVal=-MaxSam+sam
    LabC=Reclass(val,sam)


    ConfMatrix(LabC,labV)
    #[Tru,TN,FN,TP,FP]=Eval(labV,LabC)
    Metrics(LabC,AucVal,labV)

    #ConstructHistogram(clf.score_samples(X),'Hist','x','y',1000)




def ManRoc(TP,FP,TN,FN):
    Sens=TP/(TP+FN)
    Spec=TN/(FP+TN)
    return((Sens+Spec)/2)



def Metrics(y_pred,y_scores,y_true):
    print('Under curve Roc:',MetricRocAuc(y_true,y_scores))
    print("Manuel Auroc curve:",ConfMatrix(y_pred,y_true))

    #print('F1 score:',MetricF1(y_true,y_pred))
    #print('MCC score:',MetricMcc(y_true,y_pred))

def MetricRocAuc(y_true,y_scores):
    from sklearn.metrics import roc_auc_score
    from sklearn.metrics import roc_curve
    fpr, tpr, thresholds = roc_curve(y_true,y_scores)
    #print("RocAuc fpr:",fpr)
    #print("RocAuc tpr:",tpr)
    #PlotFigure(fpr,tpr,'title','FPR','TPR')
    return roc_auc_score(y_true, y_scores)

def MetricF1(y_pred,y_true):
    from sklearn.metrics import f1_score
    return f1_score(y_true, y_pred)

def MetricMcc(y_pred,y_true):
    from sklearn.metrics import matthews_corrcoef
    return matthews_corrcoef(y_true, y_pred)

def Search(Tscore,lab):
    #grid=np.array([-0.35,-0.40,-0.45,-0.50,-0.55,-0.60,-0.70])
    grid=np.array([0.2,0.15,0.13,0.12,0.11,0.1,0.09,0.06,0.05,0.03,0,-0.03])

    T=np.zeros(len(grid))
    for i in range(len(grid)):
        labC=Reclass(grid[i],Tscore)
        T[i]=ConfMatrix(labC,lab)

    ind=T.argmax()
    return(grid[ind])

def Reclass(val,Tscore):
    LabC = pandas.DataFrame(1, index=range(len(Tscore)), columns=range(1))
    for i in range(len(Tscore)):
        if(Tscore[i]<=val):
            LabC.iloc[i]=0
    return(LabC.to_numpy())

def PlotFigure(X,Y,title,xlab,ylab):
    plt.plot(X,Y,'.')
    plt.title(title)
    plt.xlabel(xlab)
    plt.ylabel(ylab)
    plt.show()

def ConstructHistogram(matrix,title,xlab,ylab,bi):
    #plt.hist((matrix.mean(axis=0)), bins = bi)
    plt.hist((matrix), bins=bi)
    plt.title(title)
    plt.ylabel(ylab)
    plt.xlabel(xlab)
    plt.show()

def TSN(X,lab):
    from sklearn.manifold import TSNE
    l=np.array([20])
    for i in l:
        X_emb=TSNE(n_components=3,perplexity=i).fit_transform(X)
        PlotAll(X_emb[:,0],X_emb[:,1],X_emb[:,2],lab,'TSNE','','','')




def PlotAll(scatter_x,scatter_y,scatter_z,group,title,xlab,ylab,zlab):
    from mpl_toolkits.mplot3d import Axes3D
    fig, ax = plt.subplots()
    ax = fig.add_subplot(111, projection='3d')
    cdict = {0: 'red', 1: 'blue'}
    #,2: 'green',3:'yellow',4:'black',5:'m',6:'c',7:'#efffff',8:'#730000',9:'#f005fb',10:'#7
    #
    for g in np.unique(group):
        ix = np.where(group == g)
        ax.scatter(scatter_x[ix[0]], scatter_y[ix[0]],scatter_z[ix[0]], c = cdict[g], label
    ax.legend()
    ax.set_title(title)
    ax.set_xlabel(xlab)
    ax.set_ylabel(ylab)
    ax.set_zlabel(zlab)

    plt.show()

def PCA(X,lab,title):
    from sklearn.decomposition import IncrementalPCA
    ipca = IncrementalPCA()
    fit=ipca.fit(X)
    Y=ipca.transform(X)
    print(fit.explained_variance_ratio_[2]*100)
    xlab='PCA 1 explains: '+str("%.2f" % round(fit.explained_variance_ratio_[0]*100, 2))+'%
    ylab='PCA 2 explains: '+str("%.2f" % round(fit.explained_variance_ratio_[1]*100, 2))+'%
    #PlotAll(Y[:,0],Y[:,1],lab,title,xlab,ylab)
    PlotFigure(Y[:,0],Y[:,1],title,xlab,ylab)

Init()
```

7.3 Code for MDA

```
# This file concludes a part of a problem in the course
   MSA520 year 2020 at GU
# The problem is for the VOLOVO-project
# This code examines the performance of MDA-
   classification on the validation data set
# The MDA concludes to performing quite well on
   distinguishing between in_in and in-Out data (first
   section of code)
# But does not perform well on any other relevant
   classification (other sections)

# setwd("Z:/volvo_project-master/volvo_project-master")


# Plan: 1. Differentiate between In and {Miss U Out}
#       2. Differentiate In and Miss
#       3. ...?
#       4. Final testing

# Note: import the right sized validation or test set for
    the purpose


install.packages("ROCR")
library(ROCR)
library(xtable)
```

```
# 1. Differentiate between In and {Miss U Out}
    ____________________________________________________________

# import data
data <- read.csv("X_train.csv")
data <- data[order(data$class),]

# validation

# large data set
valid <- read.csv("X_test.csv")
valid <- valid[order(valid$class),]


# Train and test MDA-model for each of the 10 classes of
    data
x <- c(0,0) # vector for storing predictions and indices


for (i in 1:10) {

  # import data and train class-specific model
  data_s <- data.frame(subset(data, data$class == i-1)) #
       load data of the specific class

  train_data <- data.frame((data_s[,2:11]))

  mda_out <- mda::mda(as.factor(data_s$inlier) ~ ., data
      = train_data, method = mda::mars)

  # validation data
  test_data <- data.frame(subset(valid, valid$class == i
      -1))

  # store predictions
  preds <- data.frame(as.numeric(predict(mda_out,
      test_data[2:11]))-1) # predictions

  x <- rbind(x, preds)



}
```

```
x <- x[-1,] # remove empty value

valid$predicted <- x

# quick performance check
t <- length(valid$class)
evaluation <- length(which(valid$inlier[1:t]!=
    valid$predicted[1:t]))
evaluation/t # so about 23% of data gets miss-classified


# class-wise cfmat's
cmatlist <- list()
AUROCvec <- matrix(0,1,10)

for (i in 1:10) {

valid_t <- data.frame(subset(valid, valid$class == i-1))

cmat <- matrix(0,2,2)

cmat[1,1] <- length(which(valid_t$inlier==0 &
    valid_t$predicted==0))
cmat[1,2] <- length(which(valid_t$inlier==0 &
    valid_t$predicted==1))
cmat[2,1] <- length(which(valid_t$inlier==1 &
    valid_t$predicted==0))
cmat[2,2] <- length(which(valid_t$inlier==1 &
    valid_t$predicted==1))

cmatlist[[i]] <- cmat

ROC <- c(cmat[1,1]/(cmat[1,1]+cmat[2,1]), cmat[1,2]/(cmat
    [1,2]+cmat[2,2]))
ROC

ROCx <- c(0,ROC[2],1)
ROCy <- c(0,ROC[1],1)

plot(ROCx,ROCy)


AUROC <- ROC[1]*(1-ROC[2])+(1-ROC[1]*(1-ROC[2])-ROC
    [2]*(1-ROC[1]))/2 #geometric calculation of auroc,
    divide the plot into 4 regions...
AUROCvec[1,i] <- AUROC
}

# 2. Differentiate In and Miss
    ------------------------------------------------------------

# import data
data <- read.csv("X_train.csv")
data <- data[order(data$class),]
data <- subset(data, data$type!= "x_out")

# validation

# large data set
valid <- read.csv("X_test.csv")
valid <- valid[order(valid$class),]
valid <- subset(valid, valid$type!= "x_out")


# Train and test MDA-model for each of the 10 classes of
    data
x <- c(0,0) # vector for storing predictions and indices

for (i in 1:10) {

  # import data and train class-specific model
  data_s <- data.frame(subset(data, data$class == i-1)) #
       load data of the specific class

  train_data <- data.frame((data_s[,2:11]))

  mda_out <- mda::mda(as.factor(data_s$inlier) ~ ., data
      = train_data, method = mda::mars)

  # validation data
  test_data <- data.frame(subset(valid, valid$class == i
      -1))

  # store predictions
  preds <- data.frame(as.numeric(predict(mda_out,
      test_data[2:11]))-1) # predictions

  x <- rbind(x, preds)
```

```
}


x <- x[-1,] # remove empty value

valid$predicted <- x

# quick performance check
t <- length(valid$class)
evaluation <- length(which(valid$inlier[1:t]!=
    valid$predicted[1:t]))
evaluation/t # so about 23% of data gets miss-classified


cmat <- matrix(0,2,2)

cmat[1,1] <- length(which(valid$inlier==0 &
    valid$predicted==0))
cmat[1,2] <- length(which(valid$inlier==0 &
    valid$predicted==1))
cmat[2,1] <- length(which(valid$inlier==1 &
    valid$predicted==0))
cmat[2,2] <- length(which(valid$inlier==1 &
    valid$predicted==1))

cmat

ROC <- c(cmat[1,1]/(cmat[1,1]+cmat[2,1]), cmat[1,2]/(cmat
    [1,2]+cmat[2,2]))
ROC

ROCx <- c(0,ROC[2],1)
ROCy <- c(0,ROC[1],1)

plot(ROCx,ROCy)


AUROC <- ROC[1]*(1-ROC[2])+(1-ROC[1]*(1-ROC[2])-ROC
    [2]*(1-ROC[1]))/2 #geometric calculation of auroc,
    divide the plot into 4 regions...
AUROC
```

7.4 DBSCAN Implementation

```
library(fpc)
library(dbscan)
library(tidyverse)
library(dplyr)
library(factoextra)
library(caret)
library(ROCR)
library(xtable)

classes <- seq(0,9)

labels <- c(1,0)

### training data ####
X_train <- read.csv("X_train.csv")
colnames(X_train)[2:11] <- classes

### validation data ####
X_valid <- read.csv("X_valid.csv")
colnames(X_valid)[2:11] <- classes

small_valid <- read.csv("small_valid.csv")
colnames(small_valid)[2:11] <- classes


#### DBSCAN Classwise #####

#choose training data
train_data <- X_train


#choose validation data
valid_data <- small_valid


# finding optimal classwise eps and minpts

eps <- seq(0.001,1.0,length.out = 5)
minpts <- c(10,20,30,40,50,100,200,300,500)
### Training subsets

# inliers and missclassified inliers
train_data <- subset(train_data,type == "x_in" | type == "x_miss")



# only inliers
```

```
train_data <- subset(train_data, type =="x_in")

# inliers and outliers
train_data <- subset(train_data, type == "x_in" | type == "x_out")


auc_class <- numeric(length(10))
auc_index <- list(length(10))

for (i in 1:10){
  train <- subset(train_data, class == i-1)
  valid <- subset(valid_data, class == i-1)
  auc_matrix <- matrix(data = NA, nrow = length(eps), ncol = length(minpts))
  print(i/10)
  for (j in 1:length(eps)){
    for( k in 1:length(minpts)){
      set.seed(123)
      db <- dbscan(train[,2:11], eps = eps[j], minPts = minpts[k])
      pred <- predict(db, newdata = valid[,2:11], data = train[,2:11])
      pred[which(pred != 0)] <- 1

      conf <- confusionMatrix(as.factor(pred),as.factor(valid$inlier),positive = "0")

      #print(conf$table)

      #AUROC
      predi <- prediction(pred, valid$inlier)
      perf <- performance(predi,"auc")

      auc_matrix[j,k] <- perf@y.values[[1]][1]

    }
  }
  auc_class[i] <- max(auc_matrix)
  auc_index[[i]] <- which(auc_matrix == max(auc_matrix), arr.ind = TRUE)
}

### Optimal Eps and Minpts for each class ###

optimal <- list(length(10))

for (i in 1:10){
  opt <- auc_index[[i]]
  eps_minpts <- matrix(data = NA, nrow = nrow(opt), ncol = 2)

  for (j in 1:nrow(opt)){
    optimum <- opt[j,]

    eps_minpts[j,] <- c(eps[optimum[1]],minpts[optimum[2]])

  }
  optimal[[i]] <- eps_minpts
}


###### CODE FOR EVALUATION OF VALIDATON DATA ######

AUC <- numeric(10)

CONF_MAT <- list(length(10))

temp <- list(length(10))

TOTAL_CONF <- matrix(data = 0,2,2)

for (i in 1:10){
  train <- subset(train_data, class == i-1)
  test <- subset(valid_data, class == i-1)

  set.seed(123)
  db <- dbscan(train[,2:11], eps = optimal[[i]][1,1], minPts = optimal[[i]][1,2])
  #db <- dbscan(train[,2:11], eps = 0.068, minPts = 100)
  pred <- predict(db, newdata = test[,2:11], data = train[,2:11])
  pred[which(pred != 0)] <- 1

  temp [[i]]<- cbind(test$type,pred)

  conf <- confusionMatrix(as.factor(pred),as.factor(test$inlier))

  CONF_MAT[[i]] <- conf$table

  TOTAL_CONF[1,1] <- TOTAL_CONF[1,1] + conf$table[1,1]
  TOTAL_CONF[1,2] <- TOTAL_CONF[1,2] + conf$table[1,2]
  TOTAL_CONF[2,1] <- TOTAL_CONF[2,1] + conf$table[2,1]
  TOTAL_CONF[2,2] <- TOTAL_CONF[2,2] + conf$table[2,2]

  #AUROC
  predi <- prediction(pred, test$inlier)
  perf <- performance(predi,"auc")

  AUC[i] <- perf@y.values[[1]][1]
```

```
}

### CODE FOR BREAKING DOWN CLASSIFICATIONS ####
correct_miss <- numeric(length(10))
correct_ood <- numeric(length(10))

nr_miss <- numeric(length(10))
nr_ood <- numeric(length(10))


for (i in 1:10){

  class_pred <- temp[[i]]

  nr_miss[i] <- length(which(class_pred[,1] == 2))
  nr_ood[i] <- length(which(class_pred[,1] == 3))

  correct_miss[i] <- length(which(class_pred[,1] == 2 & class_pred[,2] == 0))
  correct_ood[i] <- length(which(class_pred[,1] == 3 & class_pred[,2] == 0))

}

acc_miss <- correct_miss/nr_miss
acc_ood <- correct_ood/nr_ood

tot_acc_miss <- sum(correct_miss)/sum(nr_miss)
tot_acc_ood <- sum(correct_ood)/sum(nr_ood)



##### CODE FOR FINAL TESTING ######

X_train <- read.csv("X_train.csv")
colnames(X_train)[2:11] <- classes

X_test <- read.csv("X_test.csv")
colnames(X_test)[2:11] <- classes

small_test <- read.csv("small_test.csv")
colnames(small_test)[2:11] <- classes


# ALL, SMALL , LARGE
full_train <- X_train

opt_all_large <- list.load("optimal_X_all_large.rdata")
```

```
opt_all_small <- list.load("optimal_all_small.rdata")



# IN_MISS, SMALL , LARGE
train_in_miss <- subset(X_train,type == "x_in" | type == "x_miss")

opt_inmiss_large <- list.load("optimal_in_miss_large.rdata")

opt_inmiss_small <- list.load("optimal_in_miss_small.rdata")


# IN , SMALL , LARGE
train_in <- subset(X_train,type == "x_in")

opt_in_large <- list.load("optimal_in_large.rdata")

opt_in_small <- list.load("optimal_in_small.rdata")



# choose training, testing and optimal

train_data <- train_in
test_data <- small_test
optimal <- opt_in_small


AUC_test <- numeric(10)

CONF_MAT_test <- list(length(10))

temp_test <- list(length(10))

TOTAL_CONF_test <- matrix(data = 0,2,2)

for (i in 1:10){
  train <- subset(train_data, class == i-1)
  test <- subset(test_data, class == i-1)

  set.seed(123)
  db <- dbscan(train[,2:11], eps = optimal[[i]][1,1], minPts = optimal[[i]][1,2])
  pred <- predict(db, newdata = test[,2:11], data = train[,2:11])
  pred[which(pred != 0)] <- 1
  conf <- confusionMatrix(as.factor(pred),as.factor(test$inlier))

  temp_test[[i]]<- cbind(test$type,pred)

  CONF_MAT_test[[i]] <- conf$table

  TOTAL_CONF_test[1,1] <- TOTAL_CONF_test[1,1] + conf$table[1,1]
  TOTAL_CONF_test[1,2] <- TOTAL_CONF_test[1,2] + conf$table[1,2]
  TOTAL_CONF_test[2,1] <- TOTAL_CONF_test[2,1] + conf$table[2,1]
  TOTAL_CONF_test[2,2] <- TOTAL_CONF_test[2,2] + conf$table[2,2]

  #AUROC
  predi <- prediction(pred, test$inlier)
  perf <- performance(predi,"auc")

  AUC_test[i] <- perf@y.values[[1]][1]
}


### CODE FOR BREAKING DOWN CLASSIFICATIONS ####
correct_miss <- numeric(length(10))
correct_ood <- numeric(length(10))

nr_miss <- numeric(length(10))
nr_ood <- numeric(length(10))


for (i in 1:10){

  class_pred <- temp_test[[i]]

  nr_miss[i] <- length(which(class_pred[,1] == 2))
  nr_ood[i] <- length(which(class_pred[,1] == 3))

  correct_miss[i] <- length(which(class_pred[,1] == 2 & class_pred[,2] == 0))
  correct_ood[i] <- length(which(class_pred[,1] == 3 & class_pred[,2] == 0))

}

acc_miss <- correct_miss/nr_miss
acc_ood <- correct_ood/nr_ood

tot_acc_miss <- sum(correct_miss)/sum(nr_miss)
tot_acc_ood <- sum(correct_ood)/sum(nr_ood)

nr_miss[11] <- sum(nr_miss)
```

```
nr_ood[11] <- sum(nr_ood)

acc_miss[11] <- sum(correct_miss)/nr_miss[11]
acc_ood[11] <- sum(correct_ood)/nr_ood[11]

correct_miss[11] <- sum(correct_miss)
correct_ood[11] <- sum(correct_ood)


stat <- rbind(nr_miss,correct_miss,acc_miss,nr_ood,correct_ood,acc_ood)
colnames(stat) <- c(0:9,"TOT")
```