university of groningen

faculty of science and engineering

Feature relevance bounds in classification problems

Rogchert Zijlstra

university of groningen

faculty of science and engineering

University of Groningen

Feature relevance bounds in classification problems

Master's Thesis

To fulfil the requirements for the degree of
Master of Science in Computing Science
at University of Groningen under the supervision of
Prof. dr. M. Biehl,
Prof. dr. K. Bunte
and
A.F. Nolte

Rogchert Zijlstra

March 25, 2022

Contents

Acknowledgments

Firstly, I want to thank Michael Biehl and Aleke Nolte. Their supervision was vital for this thesis. With the help of their expertise and suggestions, we were able to push this thesis in the right direction.
Secondly, I want to thank Rick van Veen. His help was very useful for implementing the method in SKLVQ. He helped a lot when we were starting to use SKLVQ and gave helpful advice on the inner workings of SKLVQ and how to adapt them to our needs.
Thirdly, I want to thank William Pearson and Lingyu Wang. They were able to shine some light on the GAMA-KiDS data set. They helped us put the results into context and gave us guidance on topics within the data set that are interesting.
Lastly, I want to thank both KiDS and GAMA. Their data set gave us the ability to test our method on a data set which was not artificially generated.

Abstract

Generalized Matrix Learning Vector Quantization is a powerful tool for prototype-based classification. The results can also be interpreted relatively easily through the relevances obtained. The relevances obtained from GMLVQ do not show the full picture. Correlated features can greatly affect results between different trainings and ambiguous relevances can appear, which limits the reliability of the interpretation. In this thesis, we look at the interpretation of relevances. We introduce a method which allows us to look at the relevance bounds, which show the range of values a relevance can take while still retaining classification accuracy.In addition, we introduce a new dimensionality reduction technique which is able to significantly reduce the number of features during training. This allows us to calculate the relevance bounds for bigger, more complicated data sets. We apply our methods to several data sets. A set of mock data sets is used to examine the advantages and challenges of this method. Finally, we apply the method to a real world data set which classifies merger and non-merger galaxies. This data set is big enough to require the use of dimensionality reduction. The results show us that this method can effectively improve the interpretability or the relevances.

Glossary

AUROC Area Under the Receiver Operating Characteristics 28

LVQ Learning Vector Quantization 2, 21

GMLVQ Generalized Matrix Learning Vector Quantization 2, 21

PCA Principal Component Analysis 3, 10

N Number of features 7

P number of datapoints 7

C Number of classes 7

K Number of prototypes 7

X The data set 5, 7

X_j The j'th feature of of the data set 5

ξ_i Sample i from the data set 7

ω A prototype of the classifier 7

Λ The relevance matrix of the classifier 9

Ω Alternate representation of the relevance matrix 9

$\cdot^*$ Any element after the dimensionality reducing transformation 10

List of Figures

List of Tables

1 Introduction

Problems involving classification have always been important to humans. Whether it is identifying if food is spoiled or determining if wild animals are dangerous predators. We have always used patterns to recognize these classes. For example, if bread turns green, it is spoiled. These kinds of classification problems are very relevant to our survival. We have learned from our experiences and are able to do these classifications without too much trouble.

One important thing to note is that in many cases we do not know how we recognize these classes. Asking a person how they differentiate hundreds of different people or how they read different handwritings will only give us some vague answers, but no definitive methods. These are cases where a human can classify something but do not have the details of the process for doing so.

There are many more patterns which humans cannot identify. Some are simply too big in scope for humans, for some we lack the senses to properly measure them, or we are simply too slow at them. This is where machine learning classification algorithms come into the picture. Detecting illnesses from a large amount of medical data can be highly beneficial. Digitizing large numbers of documents can be done a lot faster by machines than humans once a machine is trained to do so.

There are several methods to train a classifier. The simplest method can only be used when we know explicitly how to classify something. If we know by which rules to identify something, we can simply implement the rules and let the machine take it from there. However, in many cases we do not know how to classify something. In this case, we need to rely on the machine to learn the way to classify an object. This is primarily done in one of two ways, supervised and unsupervised learning. Supervised learning method involves learning from examples. We take many examples of our problem for which we know the classification. Each example has several features that could be relevant for the classification. The machine then uses this information to learn which of these features is important for classification and how they can be used. For unsupervised learning, these examples are not present. In this case, we often employ a function whose value can tell us how well our machine is performing. The machine can then try to optimize that value. In both cases the machine will be able to classify new data that it has not seen before using the lessons learned from training.

A very intuitive idea for classification is Prototype-based classification. It relies on the idea that each class can be represented by a small number of prototypes. For each class we want to identify, we can look for prototypes that represent these classes. For example, when you think of a banana certain features might spring to mind. It is a slim, elongated fruit, often slightly curved with a yellow or green colour. It should be about 15 to 20 cm in length and around 120 grams. We can

compare other fruits to this prototype of a banana to see if it is also a banana. A lemon might agree on the colour, but the other features all do not match up enough for us to say that it is not a banana. When we have prototypes for all classes, we can classify an example simply by looking at which prototype is the closest given a predefined metric.

Some features can be more important than others. This importance, which we will refer to as the relevance of a feature, is unknown unless we have prior information. When training on the data you might expect there is a single combination of relevances that represents how to properly classify a sample. In practice this is not the case. Due to interplay between features or overlap of the classes different sets of relevances might preform equally well. For this reason simply training a model and extracting the relevances does not fully capture the importance of a feature. The method employed in this thesis forces predetermined values for the relevances. When we notice the forced relevance value decreases the performance to a significant degree, we can conclude this relevance is outside the boundaries we would call optimal.

1.1 Research Questions

This thesis focuses on Generalized Matrix Learning Vector Quantization (GMLVQ) [1]. GMLVQ is a prototype-based classification algorithm. We have altered it in such a way that we can force the relevances to the wanted values. Using this, we want to find the boundaries for the relevances of each feature. Using these boundaries, we want to see if we can determine if features are redundant or not. Furthermore, we want to investigate if these methods can be used in combination with methods to reduce dimensionality. Our research questions are as follows:

- Can meaningful relevance bounds be determined by imposing constraints on individual relevances and optimizing the remaining ones?
- Is it possible to extract relevance bounds in the original feature space from classifiers trained in a space with reduced dimensions?

1.2 Thesis outline

The remainder of this thesis is organized as follows. In chapter 2 we provide the background of GMLVQ. We look at the origins of Learning Vector Quantization (LVQ) and the developments that lead to GMLVQ. Additionally, we look at the existing methods which are used to compute the relevance bounds.
In chapter 3 we look at the mathematical basis of GMLVQ. We mathematically define LVQ and the variants that emerged from it. We also define the tools we

need for computing the relevance bounds. For the dimensionality reduction, we look at two methods. Firstly, we look at a common method for dimensionality reduction, Principal Component Analysis (PCA). Then we also introduce the Λ-transformation, which is a method of dimensionality reduction which uses information of a prior GMLVQ classifier to reduce the dimensionality. We also define the relevance bounds and introduce the concept of the null-space reduction.
Chapter 4 addresses the implementation of GMLVQ that was used and the alterations made which allows us to compute the relevance bounds.
The experimental setup is elaborated on in chapter 5. We show all the data sets used in out experimentation. This includes multiple mock data sets and a real world data set. The settings used for GMLVQ in our experiments are also specified here. Lastly, this section defines the performance metrics used in the thesis.
Chapters 6-9 contains our experimentation. We start by looking at relevance bounds for the data sets without dimensionality reduction. We then compare the methods of dimensionality reduction and show why we prefer the Λ-transformation. We investigate the effects of the Λ-transformation and look at the advantages and problems with it. Then we apply the method to the data sets we have.

2 Background

In this section we take a look at the background of the LVQ and the variants which led to GMLVQ. We also look at the different techniques required for this. After this, we look at previous attempts to obtain relevance bounds [2, 3].

2.1 LVQ and variants

Before we can look at GMLVQ, we need to understand the ideas behind LVQ. LVQ is a prototype-based classification method which was introduces by Kohonen in 1986 [4]. LVQ uses a supervised learning process to create prototypes to represent each class in the data set. During training, the prototypes are moved towards the data with the same class that the prototypes represent. Novel data can be compared to the prototypes and the class can be decided by looking at the closest prototype. Unfortunately, the method is not without problems [1]. LVQ does not have a strong mathematical rigour behind it. The main ideas behind LVQ are based in heuristics. Slightly tweaking the methods can greatly influence the results [5]. There is potential for slow convergence or highly unstable results. For this reason, many variants have been created to alleviate these issues.

We look at the two big variants that lead to GMLVQ. the first variant is the generalized LVQ (GLVQ) [6]. It changes the way the prototypes are updates. In addition to this, it introduces a cost function to evaluate the prototypes. This cost function can be minimized by using methods like gradient descent. The second big variant was the introduction of the relevance matrix. The concept of using relevance in LVQ was introduced in 2001 [7]. This method adds a relevance value for each feature. These values can be varied during training, which is able to increase or decrease the influence of features to improve performance. Both ideas were brought together to form GRLVQ (Generalized Relevance LVQ) in which the relevances was used for GLVQ [8]. The relevance allowed the method to give a low relevance to features that are not relevant but it did not have tools to deal with correlated features. For this, GMLVQ was introduced [1]. Instead of a single relevance value for each feature, a full relevance matrix is used. The values on the diagonals correspond to the relevances used in GRLVQ. On the off diagonal are the relations between the features.

All of these variants tried to eliminate the limitations of the original LVQ method, though they mostly reduce the downsides instead of fully eliminating them. The method still has significant uncertainty and the results between separate runs of the training process can differ due to factors such as correlation and local minima. With the extra layers of complexity, it has also become more influence by settings, randomized values or initialization methods.

2.2 Previous work on relevance bounds

Previous attempts have been made to search for relevance bounds in GMLVQ. In [2] a method was investigated to differentiate different types of relevant features using relevance bounds. The different types of relevances introduced are:

- Highly relevant features: Features that can not be removed from the data set without impacting classification accuracy.
- Weakly relevant features: Features that improve classification results but are not strictly necessary due to overlapping information with other features.
- Irrelevant features: Features which hold no relevance and can be discarded without impacting classification accuracy.

To achieve this, the idea of relevance bounds are introduced. For a data set X the lower relevance bound of feature X_j they look at the difference between the relevance of a feature subset S with $X_j \in S$ and the relevance of $S \setminus \{X_j\}$. The difference between the two relevances is considered the minimum X_j contributed to the relevance. For the maximum, a greedy forward-backward search is used. It looks for the subset of features which cause X_j to have the highest relevance. This method a few problems. Firstly, the lower bounds of many features tends to go to zero due to overfitting. Secondly, the heuristic used in search of the upper bound is not exhaustive and can leave a significant amount of feature groupings unexplored.

A second method explored uses the concept of a null space. Consider a data set with two features that are exactly the same, $X_1 = X_2$. For any $c \in \mathbb{R}$ we can see that $c \cdot X_1 - c \cdot X_2$ equals 0. Adding this combination of features will have a net zero influence on the classifier. Any combination of features that add up to 0 are considered to be in the null space. If we have a trained classifier which performs well, we should be able to add or subtract features in the null space without changing the performance.

To find the combinations of features in the null space features, we can use the eigenvectors of the covariance matrix of the data, $XX^\top$ [9]. All eigenvectors which have an eigenvalue of 0 are combinations of features that add up to 0. In practice having features with an eigenvalue of 0 is rare and a minimum cut-off is used. To find the upper and lower bounds can then be found by adding features from the null space to a known configuration that performs well. For each added combination, we can find the relevance of each feature and find the minimum and maximum relevance that still performs within an error margin.

This approach got introduced to GMLVQ in [3]. While both papers were able to compute relevance bounds that show which feature are relevant or irrelevant, they were still struggling with proper identification of weakly relevant features.

Furthermore, this method is unstable. Slight changes to error margins or how the null space is determined have a very significant effect on the resulting relevance bounds. In addition to this, the problem of the lower relevance bound tending to 0 still remained.

3 Methods

In this section, we go through the mathematical foundations of GMLVQ. Furthermore, we be looking at the two methods used for dimensionality reduction. We look at the relevance bounds before and after the dimensionality reduction. An extension of the optimization method is introduced. Finally, we introduce the null-space correction.

3.1 LVQ and LVQ variants

GMLVQ is an extension of LVQ. We build up towards GMLVQ by looking at some of the variations which lead to GMLVQ. Afterwards, the constrained GMLVQ is introduced.

3.1.1 LVQ

The first description of LVQ was made in 1986 by Kohonen [4, 10]. We have a data set with N dimensions and C classes. Each of the P samples is a combination of the features and a single class, $(\xi_i, y_i) \in \mathbb{R} \times \{1,\ldots,C\}, i \in \{1,2,\ldots,P\}$. The dataset with only the features and not the classes is X. We want to create prototype vectors ω that represent each class (They are called "Codebook vectors" in [4,10]). We create K prototypes with each class being represented by at least 1 prototype, $c(\omega_l) \in \{1,\ldots,C\}, l \in \{1,2,\ldots,K\}$. We move the prototypes during the training step such that each prototype ends up close the samples it represents. To train the classifier you look at a sample and the closest prototype. If they have the same class it pulls the prototype towards it, otherwise it pushes it away. Only the Euclidean distance was used at first, but we can use a generalized distance measure d^{λ} where λ are the parameters used. λ does not have to be static and may change during training as long as it is a valid distance metric.
In the original method, LVQ1, data point ξ_i is chosen at random. The random sample is compared to every prototype. The closest prototype ω_J is then adjusted using

$$\omega_J(t+1) = \omega_J(t) + \alpha(t) \cdot (\xi_i(t) - \omega_J(t)) \text{ if } c(\xi_i) = c(\omega_J), \quad (1)$$
$$\omega_J(t+1) = \omega_J(t) - \alpha(t) \cdot (\xi_i(t) - \omega_J(t)) \text{ if } c(\xi_i) \neq c(\omega_J), \quad (2)$$

where the learning rate α is between 0 and 1, $0 < \alpha(t) < 1$. $\alpha(t)$ is frequently decreased with time.
To classify a new data point ξ you compare it to all the prototypes and pick the closest one,

$$c(\xi) = c(\omega_i) \text{ where } d^{\lambda}(\omega_l, \xi) \leq d^{\lambda}(\omega_m, \xi)\ \forall l \neq m. \quad (3)$$

3.1.2 Generalized LVQ

The generalized LVQ method was introduced by Sato and Yamada [6] in 1996. The aim of generalized LVQ is to add a cost function which can be used to measure the quality of the prototypes. The classifier can then be trained by minimizing the cost function:

$$\sum_i \Phi(\mu_i) \text{ where } \mu_i = \frac{d^\lambda(\omega_J, \xi_i) - d^\lambda(\omega_K, \xi_i)}{d^\lambda(\omega_J, \xi_i) + d^\lambda(\omega_K, \xi_i)}. \tag{4}$$

Here $\Phi(x)$ is a monotonic function. ω_J is the closest prototype with the correct class label, $c(\omega_J) = c(\xi_i)$. ω_K is the closest prototype with a different class label, $c(\omega_K) \neq c(\xi_i)$. μ_i can decrease in two ways, either $d^\lambda(\omega_J, \xi_i)$ becomes smaller or $d^\lambda(\omega_K, \xi_i)$ grows bigger. This means cost function decreases as a correct prototype get closer and all the wrong prototypes get further away from the data point. The cost function becomes negative when the closest prototype to ξ has the right class. If $d^\lambda(\omega_J, \xi_i) < d^\lambda(\omega_K, \xi_i)$ then the numerator of μ_i in Equation 4 becomes negative. Minimizing the cost function can be done using an optimization method like gradient descent.

3.1.3 Generalized Relevance LVQ

This thus far we used the euclidean distance which assumes every feature is equally important. We can alter this distance measure by including a relevance value to each feature. As the importance of features is not known beforehand, we need to find the correct relevance during the training. This was implemented with Generalized Relevance LVQ [8, 11]. The metric d^λ is altered such that a weight is added to each dimension of the space:

$$d^\lambda(\omega, \xi) = \sum_j \lambda_j (\omega_j - \xi_j)^2, \tag{5}$$

$$\sum_j \lambda_j = 1. \tag{6}$$

Where λ_j is the weight assigned to feature i, λ_i, also referred to as the relevance and ω_j is the j'th component of the prototype ω. In the previous variants normalization could be used to improve performance, but for Generalized Relevance LVQ it is necessary for relevances to make sense. Ideally, the relevance would represent the importance of the feature, which is not always true without normalization. If X_j has a low relevance, it can still have an high influence if the scale of X_j is large. A feature could have a high relevance but be on a very small scale, adding almost northing to the equation 5. We need the data to be on the same scale. To achieve this we use z-scoring. With the z-score transformation we get

the same mean, 0, and standard deviation, 1, for each feature. This ensures all features are roughly on the same scale.

3.1.4 Generalized Matrix LVQ

The concept of relevances can be expanded. In a paper published in 2009 GMLVQ is introduced [1]. GMLVQ uses a $N \times N$ matrix for the relevance, Λ, instead of a single relevance per feature to account for correlations between features. With this we define the following distance metric:

$$d^{\Lambda}(\omega,\xi) = (\xi-\omega)^{\top}\Lambda(\xi-\omega). \tag{7}$$

d^{Λ} is not guaranteed to be a valid distance measure. For it to become valid we need two properties. Λ needs to be symmetric and non-negative. We can guarantee these properties if

$$\Lambda = \Omega^{\top}\Omega \tag{8}$$

is enforced with an arbitrary $M \times N$ matrix Ω. Often $M = N$ and Ω is a square matrix, but there are cases where choosing $M \leq N$ to reduce the dimensions of Ω is preferable [12]. For our experiments we have chosen $M = N$. We need to enforce these conditions throughout training. This means Λ should not be changed directly. Any time Λ needs to be altered the alterations are made to Ω instead. By only changing Ω we guarantee d^{Λ} stays a valid distance measure. The precise of these alterations can be found in the original paper [1]. We still need the relevances of the features to sum to one. These relevances are now on the diagonal of Λ. Using the decomposition in equation 8 we can write this in terms of Ω:

$$\sum_j \Lambda_{j,j} = \sum_{k,j} \Omega_{k,j}\Omega_{k,j} = \sum_{k,j} (\Omega_{k,j})^2 = 1. \tag{9}$$

To achieve this we divide by the sum of squared roots: $(\sum_{kj}(\Omega_{k,j})^2)^{1/2}$. We need to do this after every step to ensure valid relevances.

3.1.5 Constrained GMLVQ

For our methods, we want to fix the relevance of a single feature during the training. As the main diagonal hold the relevances we can normalize similarly to GMLVQ (See equation 9). We need to split the normalization process into two parts. The first step fixes the relevance of our fixed feature f to the fixed value Y, $\Lambda_{f,f} = Y$. Afterwards, the remaining feature are normalized to the remaining

relevance, $1-Y$. This gives us the following normalization process:

$$\sum_{j,f}(\Omega_{j,f})^2 = Y, \tag{10}$$

$$\sum_{j,k\neq f}(\Omega_{j,k})^2 = 1-Y. \tag{11}$$

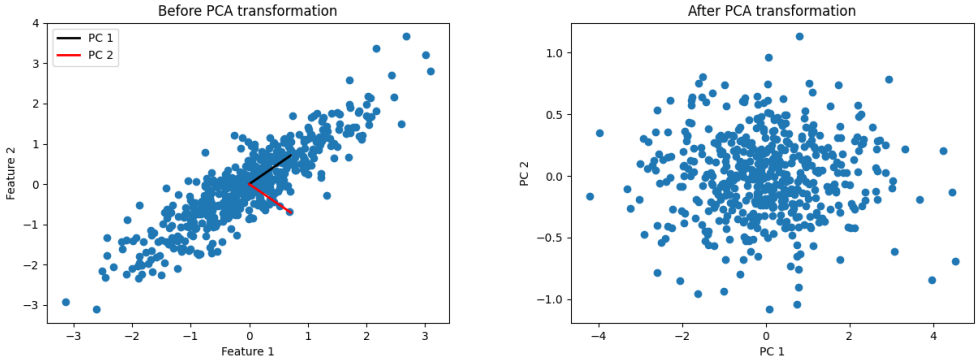
As with GMLVQ we need to apply this after every learning step.

3.2 Dimensionality Reduction

We look at two different methods for dimensionality reduction. We start by looking at the Principal Component Analysis (PCA). Then we look at the Λ-transformation we developed for our specific application. To make the distinction between elements before and after the transformation, we add an asterisk to any element after the transformation, $\cdot^*$.

3.2.1 Principal Component Analysis

PCA is a method often used to reduce the dimensionality of a data set. The original ideas behind PCA can be found in a paper by Pearson in 1901 [13]. The modern name and formulation can be attributed to Hotelling [14].
The main idea behind PCA is to find the linear combinations of features with the highest variances. These combinations of features, called Principal Components (PC), are calculated such that they capture the highest variant while also being uncorrelated to the other principal components. The first component is the component with the biggest variance. The second component captures the biggest amount of remaining variance while being orthogonal to the first one. This process repeats until a predefined ending criterium is reached. An example of this can be seen in Figure 1. Here, a dataset generated by a multivariate normal distribution is transformed using PCA. In figure 1a, we see the two features of the original data set. In addition to this, the direction of the first two principal components is show. Figure 1b shows the distribution after applying the PCA.



(a) X, the data set before PCA transformation (b) X^*, the data set after PCA transformation

Figure 1: PCA transformation of a dataset generated by a multivariate normal.

There are multiple ways to calculate the principal components. We used the singular value decomposition to get our PCs [15]. The singular value decomposition separates the data matrix X into three separate matrices:

$$X = USV^\top. \tag{12}$$

Our data set X is a $P \times N$ matrix. U is a $P \times P$ matrix, S is a $P \times N$ matrix and V is a $N \times N$ matrix. V contains all the linear combinations of features that make up the PCs in the columns. The diagonal of S contains the part of variance that is explained by each PC. To reduce the number of features, we transform the data only using the PCs that explain the biggest part of the variation. To do this V is shortened to V_r by removing all columns with small explained variance. The transformed data set can be calculated using

$$X^* = XV_r. \tag{13}$$

3.2.2 Λ-transformation

The PCA transformation prioritized variation in the data set to determine which features are important so keep. However, there is little to guarantee us that features with high variation are also important for classification. Ideally, we would like to select the linear combination of features that are relevant to our classification. After training a GMLVQ classifier, the information of the relevant linear combinations is stored in the relevance matrix, Λ. The eigenvectors of Λ form an orthogonal set of features whose relevance is equal to the eigenvalues of the eigenvectors [1, 12]. Thus, the eigenvectors of Λ would make for a good basis for a transformation to reduce the number of features.
To obtain Λ we need to train a classifier on the data set. Afterwards, we can then

use the Λ to reduce the number of features. If we only needed to train on the data set once, this would be useless as the calculation for the reduction is the same as just training on the full data set. However, in our case, this is less of an issue. We are training a classifier for each combination of feature and fixed feature value (see section 3.3). The extra computational cost of a single training process is small compared to the large benefit we gain from reducing the number of features in our data set. When using this transformation, which we will refer to as the Λ-transformation, we need to change our methodology a bit. We need to train once without fixing the relevance of any feature to be able to find Λ for the transformation. The transformation matrix T can be found be looking at the canonical form of Λ, Λ_C, where the eigenvectors of Λ are the columns sorted by their eigenvalues. We reduce the number of columns by removing the eigenvectors of Λ_C with a low eigenvalue to make our transformation matrix T. T is a $N \times M$ matrix where $M \leq N$. The transformation from the original data set and vice versa can be calculated using

$$X^* = X \cdot T, \tag{14}$$

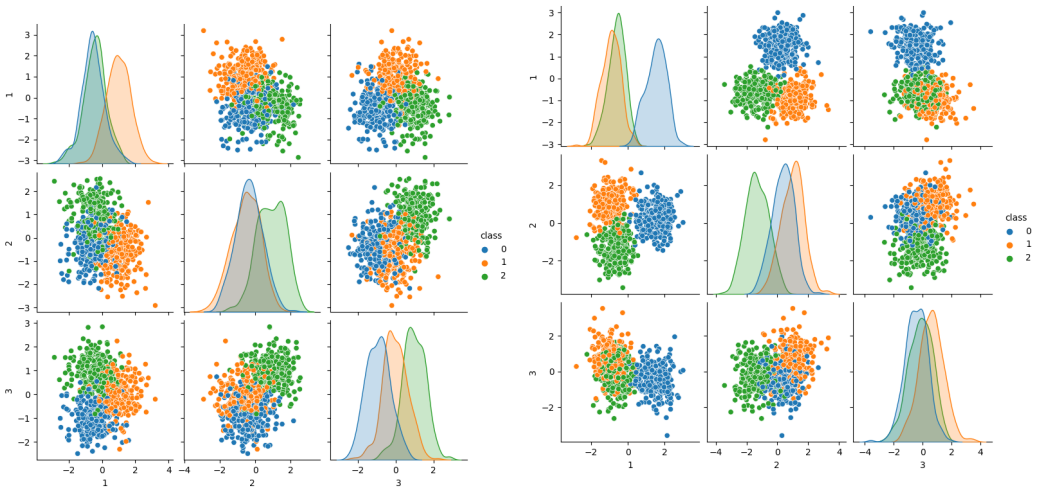
$$X = X^* \cdot T^\top. \tag{15}$$

This same process can be applied to the relevance matrix. Unlike our data set X, Λ is a square matrix where both the axis need to be transformed. To do this we use

$$\Lambda^* = T \cdot \Lambda \cdot T^\top, \tag{16}$$

$$\Lambda = T^\top \cdot \Lambda^* \cdot T. \tag{17}$$

To see the effect of the Λ-transformation we can plot some features before and after the transformation. In figure 2a we see the first 3 features of the original data set that contains 5 features. More information on this data set can be found in section 5.1.2. We can see an individual features alone are not enough as we see a lot of overlap on the diagonal. The first feature has a significant overlap between class 0 and 2 and the second feature has a lot of overlap between class 0 and 1. On the off-diagonal we see scatter plots of 2 features. In all the plots of the original features we see a significant overlap between the classes. In the second figure, 2b, we see the transformed features. As Λ_C was sorted by their eigenvalues we expect the first features to be the most significant. On the diagonal there is still some overlap visible. These features alone would not be able to classify very accurately. On the off-diagonal we can already see considerably improved separation between the classes, especially between feature 1 and 2. This is exactly what we want from the transformation. The first few features have become the most important to distinguish the classes.



(a) X_1, X_2, X_3 of the original data set

(b) X_1^*, X_2^*, X_3^* The 3 features of the transformed data set

Figure 2: A data set before and after Λ-transformation. The first three eigenvectors of Λ have eigenvalues of 0.49, 0.22 and 0.11

3.3 Relevance bounds

Whether we use dimensionality reduction or not, we need to compute the relevance bounds using all the results. To see how this is done we take a look at the error curve. An error curve shows the error for a single fixed feature with different fixed feature values. To get different relevances values for the curve we use the constrained GMLVQ. An example of an error curve can be seen in figure 3. Here we see a graph for the accuracy of a single feature. Each dot indicated a single run with a fixed relevance on the x-axis and the accuracy on the y-axis. As the fixed relevance value increases it reduces the relevance of other features as they still add up to 1. In the extreme cases we have a fixed relevance of 0 which is equal to removing the feature and a fixed relevance of 1 which considers only the feature whose value was fixed. Each run below the horizontal line is within a predefined margin of error. From the figure we would set the minimum relevance to 0.08 and the maximum to 0.28. This process is repeated for each feature.

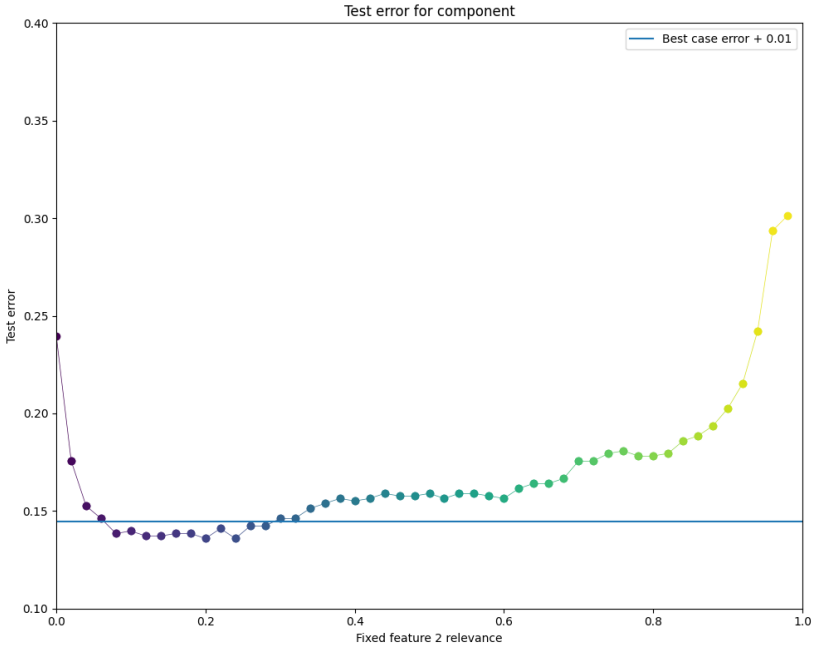
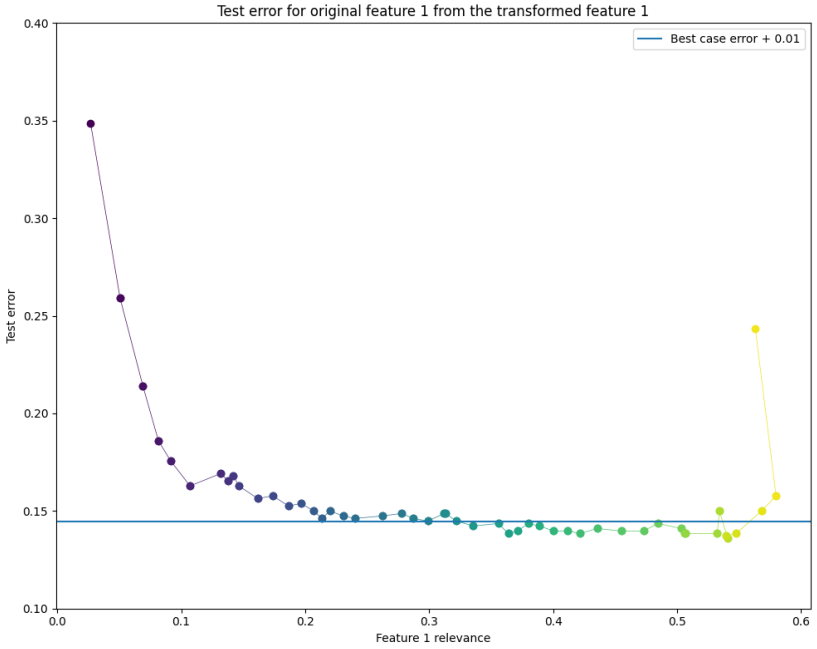


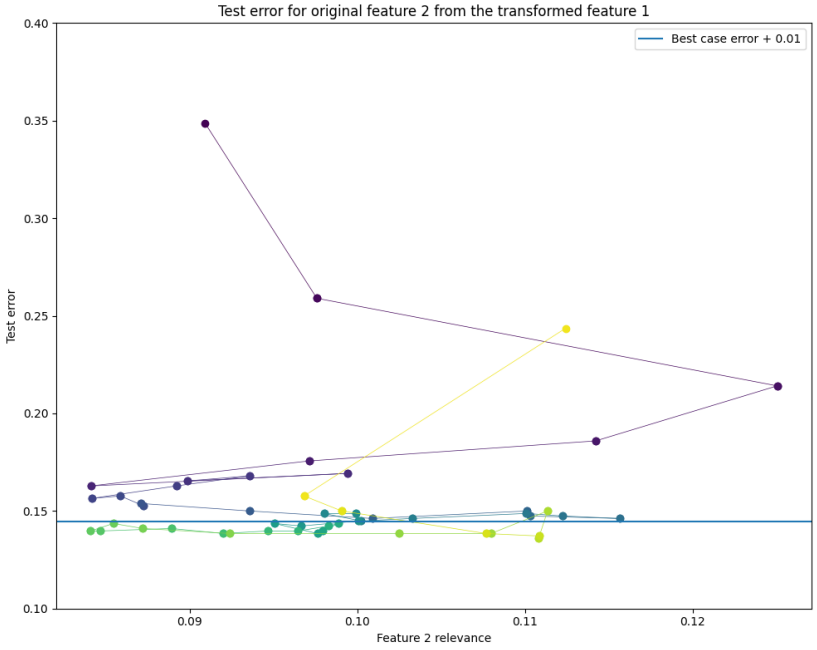
Figure 3: Example of an error curve where the second feature, X_2 has its relevance fixed.

When using dimensionality reduction, this becomes slightly more complicated. Once we have our relevance matrices for each fixed feature we need to go from the relevance of the fixed feature back to the original space. For this, we can use equation 16. Because we only have full control over the fixed feature we do not get sequential values on the x-axis going from 0 to 1 (See figure 4). In the error graphs the colour indicates the fixed relevance value going from 0 (Purple) to 1 (Yellow). The lines connecting the points are for visual aide. In figure 4a we see a feature that is well represented by X_1^*. The line does not start at 0 as other features in the transformed space can also contribute to the relevance of X_1. The relevance increases to 0.6 as it does not folly represent $X_!$. In figure 4b we see a feature that is not well represented by X_1^*. Here we see a narrow range of values for X_2 going from 0.08 to 0.13. As X_2 is not well represented by X_1^* the x-axis is not increasing as the fixed feature value increases. For each original feature, we get an error curve from each of the features in the dimensionally reduced space. All of their error curves are considered when determining the minimum and maximum relevance of the features. For each original feature we look at all their error curves and determine each minimum and maximum. We then take the lowest minimum and highest maximum relevance for that feature. For a single error margin we can show the error bounds quite simple with a bar graph as seen in figure 5a. We can also put more information into the graph. If we increase the error margin,

the range of valid relevances grows as well. This means we can see different error margins as growing relevance ranges. Because the bounds only grow larger we can overlay them without losing any information. This can be seen in 5b, where we can see some additional context. Doing this reduces the chance we take wrong conclusions simply due to an unfortunate choice of error margins.

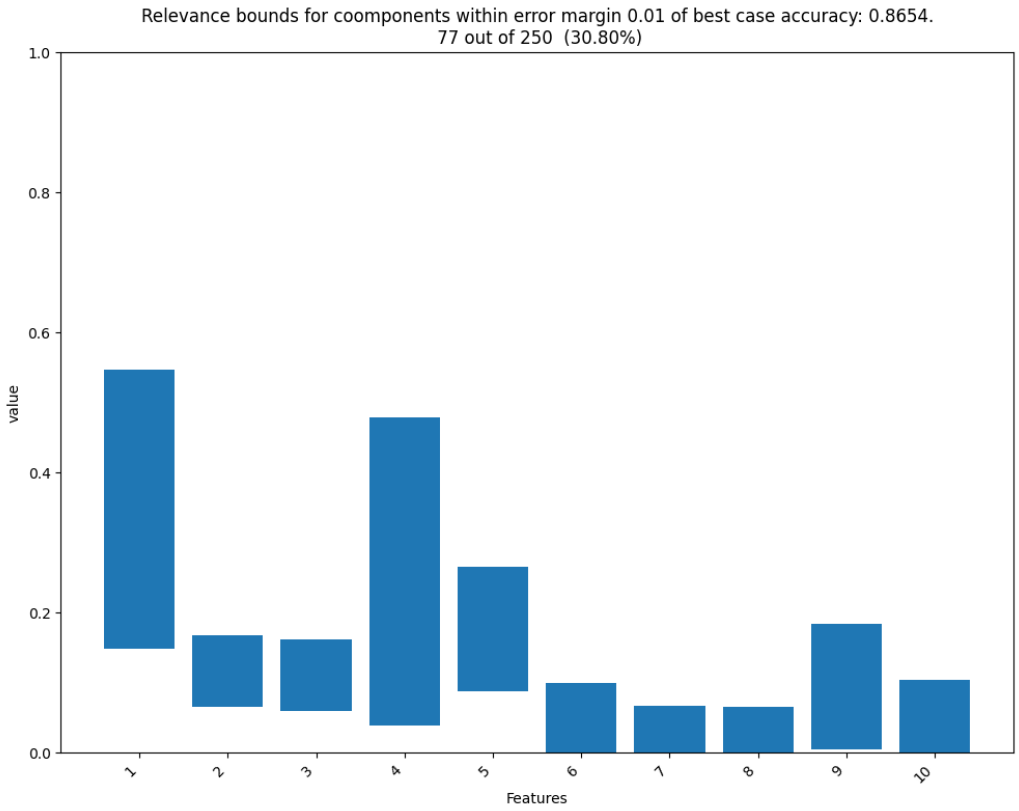


(a) Error curve of X_1, which is well represented by X_1^*.

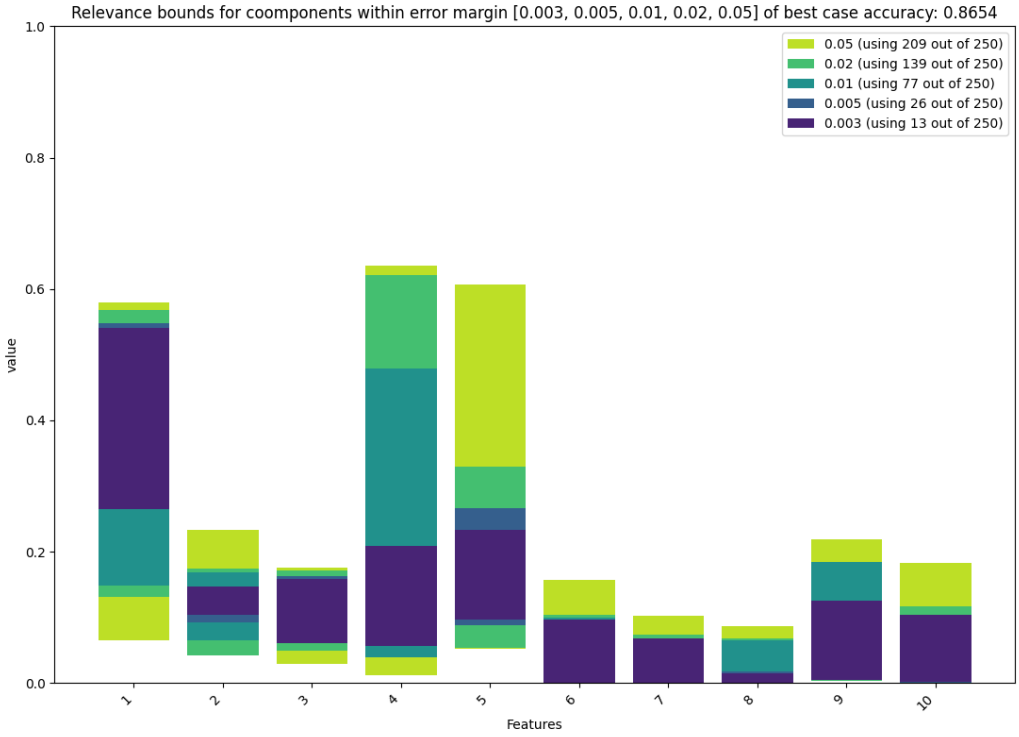


(b) Error curve of X_2, which is badly represented by X_1^*.

Figure 4: Error curves derived from constraining X_1^* in the transformed space.



(a) Relevance bounds with a 0.01 error margin.



(b) Relevance bounds with multiple error margins.

Figure 5: Relevance bounds in the transformed space.

3.4 Optimization method

We replace gradient descent with a newer variant. This variant uses the averaging of previous steps in the gradient descent to reduce the issues of oscillating convergence [16]. The methods start the same as gradient descent. After a set number of steps it starts averaging the latest results. This average is then compared to a potential new step by looking at the objective function for both. If the average scores better than the ordinary gradient descent step, it is chosen instead, and the step size of the gradient descent is reduced. This causes the steps to become smaller if it is stuck oscillating and break the oscillating patterns.

3.5 Null space correction

Correlated features can have a very significant on the relevances in GMLVQ. The correlated features can cause ambiguous relevances. To see this, we take a look at an extreme example. Consider a data set X with two features that are entirely identical, $X_1 = X_2$. Also consider that for all prototypes we have $\forall j, \omega_{j,1} = \omega_{j,2}$. In other words, the first and second component of a prototype are the same for each prototype. If X_1 and X_2 are completely irrelevant, then our relevance matrix should represent this as a relevance of 0. For $\Lambda = \Omega^\top \Omega$ this would mean

$$\Lambda_{1,1} = \sum_i \Omega_{i,1}^2 = 0, \tag{18}$$

$$\Lambda_{2,2} = \sum_i \Omega_{i,2}^2 = 0. \tag{19}$$

However, relevance matrices are not unique. We can generate an $\tilde{\Omega}$ which makes the same classifications as Ω but has non-zero relevances for X_1 and X_2. To do this we simply a row to Ω with c in the first column and $-c$ in the second column for any $c \in \mathbb{R}$,

$$\tilde{\Omega} = \Omega + \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c & -c & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{20}$$

Due to the first two features being exact copies of each other the classes given to the samples or prototypes do not change

$$\tilde{\Omega}\xi_i = \Omega\xi_i \quad \forall i, \tag{21}$$

$$\tilde{\Omega}\omega_j = \Omega\omega_j \quad \forall j. \tag{22}$$

If we look at the relevance matrix $\tilde{\Lambda} = \tilde{\Omega}^\top \tilde{\Omega}$ we can calculate the relevance of the first two features.

$$\tilde{\Lambda}_{1,1} = \Lambda_{1,1} + c^2, \tag{23}$$

$$\tilde{\Lambda}_{2,2} = \Lambda_{2,2} + c^2. \tag{24}$$

If the original Ω has a zero relevance, there are other relevance matrices that can achieve the same results with different relevances. These additions of unnecessary influences due to correlated features will be referred to as ambiguous relevances. We want to eliminate the influence of these ambiguous relevances. In section 2.2 we mentioned the concept of a null space. These ambiguous relevances we have seen are caused by combinations of features that are in the null space. There is a method to limit reduce the effect of the null space [9]. The features in the null space can be found by looking at the covariance matrix $XX^\top$. The eigenvectors of the covariance matrix with an eigenvalue of 0 are the combinations of features that comprise the null space. For $XX^\top$ we have the eigenvectors c_j with corresponding eigenvalues λ_j where they are sorted by their eigenvalues. If there is a null space then there will be a number of eigenvalues that are zero

$$\lambda_1 \geq \lambda_2 \cdots \geq \lambda_J = 0 = \lambda_{J+1} \cdots = \lambda_N. \tag{25}$$

We can construct a matrix, Ψ, which is used to correct the influence of the null space

$$\Psi = I - \sum_{i=J+1}^{N} c_j \cdot c_j^\top. \tag{26}$$

With this matrix we can remove the influence of the null space on the relevance matrix as follows

$$\hat{\Omega} = \Omega \cdot \Psi. \tag{27}$$

Here $\hat{\Omega}$ does not have any contribution from the null space but still retains the same classification as Ω. There are different ways to apply this. You could apply this technique every iteration while training, do it every P steps or only do it after the training. We have opted to apply this method every iteration. It has not been sufficiently explored whether doing it less often significantly impacts performance.

In practice, the eigenvalues are rarely exactly 0. Instead of just selecting the eigenvalues that are 0, we look at the all eigenvalues below a certain threshold. While these eigenvectors are not in the null space of X they do have a very small influence. For this threshold we use the contribution to the total eigenvalues. We

consider c_j to be in the null space if

$$\frac{\lambda_j}{\sum_i^N \lambda_j} \leq t \text{ where } t = 10^{-4}. \tag{28}$$

With this threshold we are able to reduce the influence of ambiguous relevances.

4 Implementation

The implementation of this method is done in python. We use an existing library as a base. The SKLVQ library [17] already has implementations for several LVQ variants including GMLVQ. It is scikit-learn [18] compatible and is made to be expandable. Due to the modular nature of this library it is easy to implement the constrained GMLVQ.

4.1 Changes to sklvq

For our methods we need to implement two additions. We need to implement the ability to constrain the relevance matrix. For this, two variables are added to the model. To indicate whether the constrained GMLVQ will be used a flag is added. To keep track of what needs to be constrained, a tuple is added as well. This tuple contains the index of the feature and the fixed relevance value. These values will then be used in an alternative function for normalizing the Ω matrix.

```
1 def _constrained_normalise_omega(omega: np.ndarray,
   ↪ relevance_fixed_value: tuple) -> None:
2     selector = [x for x in range(omega.shape[1]) if x !=
       ↪ relevance_fixed_value[0]]
3     if relevance_fixed_value[1] == 0:
4         omega[:,relevance_fixed_value[0]] = 0
5         np.divide(omega, np.sqrt(np.einsum("ji, ji", omega,
           ↪ omega)), out=omega)
6     else:
7         np.multiply(omega,
           ↪ np.sqrt((1-relevance_fixed_value[1])/np.einsum("ij,
           ↪ ij", omega[:,selector], omega[:,selector])),
           ↪ out=omega)
8         np.multiply(omega[:,relevance_fixed_value[0]],
           ↪ np.sqrt(relevance_fixed_value[1]/np.einsum("i, i",
           ↪ omega[:,relevance_fixed_value[0]],
           ↪ omega[:,relevance_fixed_value[0]])),out =
           ↪ omega[:,relevance_fixed_value[0]])
```

If the relevance is not fixed then the normalization of relevances happens as described in section 3.1.4. If the flag for constraining the relevance matrix is set, we will enter the alternative normalization function. This is split in two branches to prevent division by zero. In the first branch we set the feature relevance of our fixed relevance to 0. Then we normalize the matrix. For a non-zero fixed relevance this process becomes a bit more involved. For fixed feature f and fixed

feature value Y we use

$$\Omega_{ij}(t+1) = \Omega_{ij}(t) \cdot ((1-Y)/\sum_{i,j\neq f}(\Omega_{ij}(t))^2)^{1/2}, \tag{29}$$

$$\Omega_{if}(t+1) = \Omega_{iX}(t+1) \cdot (Y/\sum_{i}(\Omega_{if}(t+1))^2)^{1/2} \tag{30}$$

to normalize the matrix. We normalize the matrix such that all the features except f are normalized such that the summed squared is $1-Y$. After this, we normalize feature f such that the summed square is Y.
The second change is a small one. We need to be able to add a null-space correction matrix. As before, we add a flag to indicate if it is used. We also add a variable for the correction matrix itself to the model. The application is done by multiplying Ω with our correction matrix.

```
1 def _correct_null_space(omega: np.ndarray, correction_matrix:
   ↪ np.ndarray) -> None:
2     np.matmul(omega,correction_matrix,out=omega)
```

5 Experimental Setup

For our experimentation, we rely on four different data sets. Each of these data sets is increasing in complexity. The first three data sets are artificially generated. The fourth data set is an astronomy data set concerning merger and non-merger galaxies. We also specify the parameters and metrics used for our experiments.

5.1 Data sets

In this section, we go over the data sets used in the experiments. For each data set, we show how the data set is obtained and what we expect from it. We also mention if any parameters are used to generate the data.

5.1.1 Simple data set

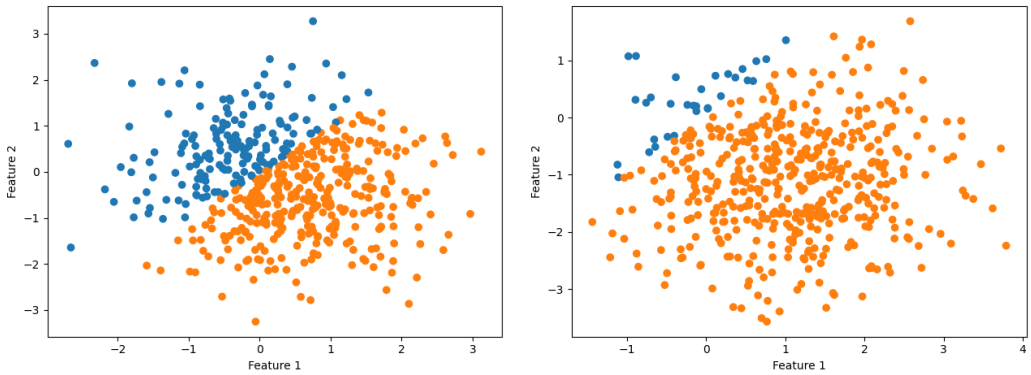
The first data set we use has 4 features. The class is based on the first two features. The first two features are normally distributed. For the normal distribution, we choose a parameter value c. X_1 is normally distributed with a mean of c and a standard deviation of 1. X_2 has the same standard deviation but a mean of $-c$. These two features are used to determine the class using

$$\forall \xi \in X : c(\xi) = \text{sign}(X_1 - X_2).$$

The other two features add some complexity to the data set. X_3 is a copy of X_1 with some Gaussian noise added to it. This results in X_3 to be highly correlated to X_1. The final feature X_4 is a noise feature. This is done with a simple Gaussian noise which is not related to the classification in any way. For a $c, d \in \mathbb{R}$:

$$\begin{aligned} X_1 &\sim \mathbf{N}(c, 1), \\ X_2 &\sim \mathbf{N}(-c, 1), \\ X_3 &\sim X_1 + \mathbf{N}(0, d), \\ X_4 &\sim \mathbf{N}(0, 1). \end{aligned}$$

Here c determined how X_1 and X_2 are situated and thus how big both classes are. d determines how much noise is added to X_3. Examples of the first two features can be seen in figure 6.



(a) Simple data set X_1 and X_2 with $c = 0.3$ (b) Simple data set X_1 and X_2 with $c = 1$

Figure 6: First two features of the simple data set

The theoretical best case scenario would be a relevance of 0.5 for X_1 and X_2, as those two feature are the determining features for the classes. However, X_3 is a feature that is highly correlated to X_1. If the influence of noise, d is small enough, the relevance of X_3 may take the same role and share the relevance with X_1. This data set is small and simple. Because of this simplicity, a high relevance for the noise may be permissible to still gain a well performing classifier. So it is still possible that X_4 hold on to some relevance.

5.1.2 Sklearn blobs

Our second data set is generated using the MAKE_BLOBS function in sklearn [18]. This function generated a number of cluster centres in a box. Around these centres the data is generated from an isotropic Gaussian distribution (Covariance matrix is $\sigma^2 I$). This function has several parameters we can configure. These parameters are listed below with their default value that we used unless stated otherwise.

- Number of samples (Default: 800)
- Number of features (Default: 5)
- Number of classes (Default: 3)
- Standard deviation σ (Default: 0.2)
- Size of the box where the centres lie (Default: Hypercube with length 1)

This configuration gives us a data set that is less trivial than the simple data set. In figure **??** we can see an example of a data set generated with this method.

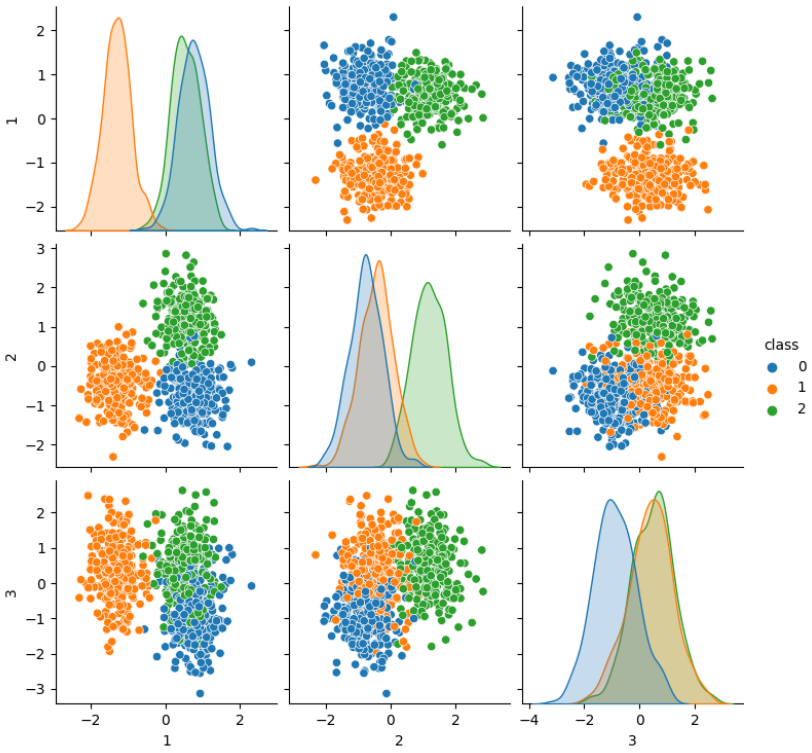


Figure 7: Pair plot of the X_1, X_2 and X_3 off a data set generated from sklearn blobs.

In figure 7 we can see the distribution of the individual features on the diagonal. We can clearly see there is overlap between the classes. For this reason, a single feature is not enough to correctly classify all examples. In the plots in the off-diagonal, we see that the blobs are already slightly separated.
To further complicate this data set, we also add noise features to this data set. The noise features added to this data set are all independent Gaussian noise. By default, we also add 5 noise features. With these features added, we have a total of 10 features.
Theoretically, the relevances we expect for this data set are quite simple. Each of the features used to generate the data set are about equally relevant. The actual optimal values differ depending on the positioning of the blobs in respect to each other. Theoretically, the noise feature should hold no relevance. In practice, this might differ a bit. Noise features may get a small relevance if their influence stays small enough to not disrupt the classification. There is also a chance that, by pure luck, the noise helps slightly in classification.

5.1.3 Blob variations

There are two variations we use. The first variant reshapes each blob such that they are not isotropic. This is done by generating a random transformation matrix for each class and transforming all data points using their respective matrix. This causes each blob to have a different shape and puts more emphasis on the interaction between features. In figure 8 we see the same data set as before but with the blobs transformed. We can see that the main axis of the blobs shifted to combinations of features.

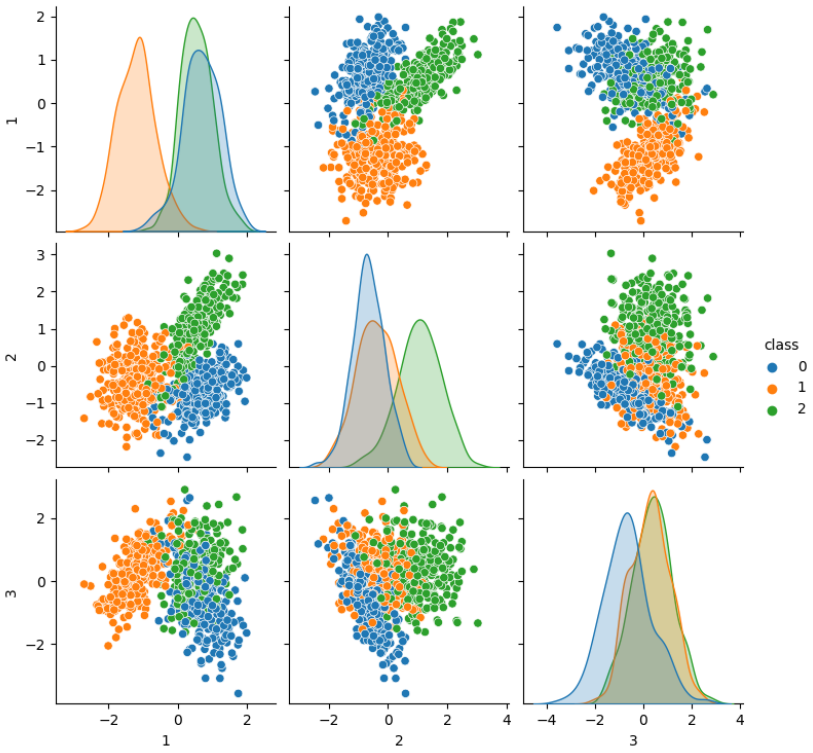


Figure 8: Pair plot of the first three features off a data set generated from sklearn blobs that are transformed.

The second variation adds copies of the first feature to the data set. These copies can be added with or without noise. This variation is used when we want to test the effect of strongly correlated features.

5.1.4 KiDS-GAMA

Our final data set is a real world data set. This data set holds information for classifying merger and non-merger galaxies. The data set is a combination of two imaging surveys, KiDS [19] and GAMA [20]. The two data sets have previously been combine in a single data set which has a label to distinguish mergers from non-mergers [21]. The features of the data set are generated using statmorph [22]. Statmorph is able to generate a large set of morphological data from images. Before we can use this data, we need to filter some data from it.
We start by removing some features from the data set. Several features in the data set are related to the position of the galaxy in the image. The position in the image should not matter for the classification of the data set. The statmorph documentation also suggests considering the removal of these features. For this reason, we remove the following features:

- nx_stamp: Number of pixels in the 'postage stamp' along the x direction.
- ny_stamp: Number of pixels in the 'postage stamp' along the y direction.
- sky_mean: Mean background value. Equal to -99.0 when there is no skybox.
- sky_median: Median background value. Equal to -99.0 when there is no skybox.
- sky_sigma: Standard deviation of the background. Equal to -99.0 when there is no skybox.
- xmax_stamp: The maximum x position of the 'postage stamp'.
- xmin_stamp: The minimum x position of the 'postage stamp'.
- xc_centroid: The x-coordinate of the centroid, relative to the original image.
- yc_centroid: The y-coordinate of the centroid, relative to the original image.
- ymax_stamp: The maximum y position of the 'postage stamp'.
- ymin_stamp: The minimum y position of the 'postage stamp'.

In addition to these feature we also need to remove two asymmetry features: asymmetry, outer_asymmetry. The calculation of these features uses the information of the merger label. We also need to remove any data points that have wrong or no information. The first step is using the flags from statmorph. Both 'flag' and 'flag_sersic' can be used to determine if either the measurement or the sersic calculation had problems. We remove all data points that have issues here.

Then we also remove all data points with 'NA' values. This brings our data set from 36753 to 3577 data points. From here we remove the outliers. We use interquartile range (IQR) to remove the outliers from our data. This leaves us with a data set with 2400 data points and 37 features. Of these 2400 data points, 962 are mergers and 1438 are non-merger.

5.2 Experimental Configurations

For GMLVQ the following settings are used by default. If different settings are used for an experiment, it will be explicitly stated.

- 90 gradient steps.
- Z-score normalization.
- 1 initial prototype step size.
- 2 initial matrix step size.
- 5 waypoints stored during waypoint gradient descent.
- Initialized prototypes by class conditional mean.
- Fixed relevance values run from 0.00 to 0.98 with 0.02 between each point.
- 1 prototype per class.

We also use 10-fold cross validation for calculating the Λ-transformation and for calculating the relevance bounds. For the computation of relevances we average the relevance matrix resulting from each fold. For unbalanced data sets we use undersampling of the majority classes to balance the classes.

5.3 Performance Metrics

For measuring performance, two different metrics are used. The first metric used is the test accuracy/error. After testing on the training set, the resulting classifier is used on the test set. The accuracy is then calculated by dividing the number of correct classifications by the total number of samples is the test set. This is then repeated for every fold and averaged. The error is simply 1 minus the accuracy.
The second metric is used for 2 class problems. The performance of 2 class problems can be measured the Area Under the Receiver Operating Characteristics (AUROC) [23]. The receiver operating characteristic curve (ROC-curve) is the curve that plots the True Positive Rate (TPT) against the False Positive Rate (FPR) for differing values of the decision function. An example of a ROC curve

can be seen in Figure 9. Here we see a ROC-curve and a diagonal line. The diagonal line is the ROC-curve that corresponds to random guessing.

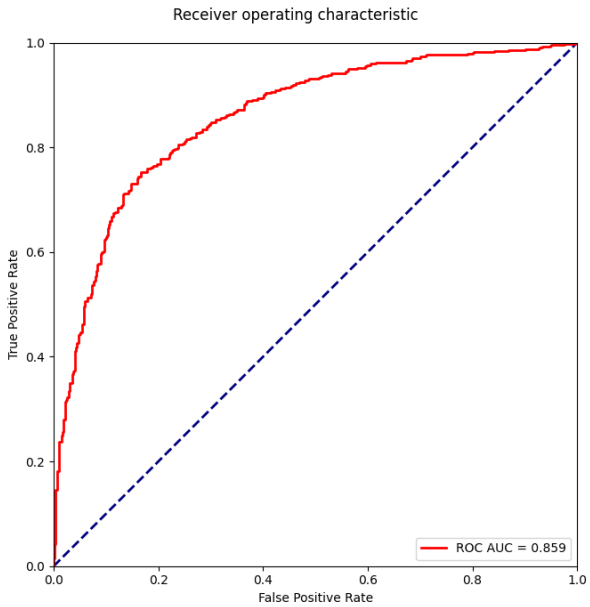


Figure 9: Example of an ROC-curve.

The AUROC is the area underneath the ROC-curve. A classifier with a higher AUROC would be a better classifier, as it means a lower FPR and higher TPR. The best case scenario is an AUROC of 1. The AUROC of random guessing is 0.5. To calculate the AUROC while using cross validation we require a process called threshold averaging which is described in [23]. This method can be used to average multiple ROC-curves. To achieve this average you look at a set number of threshold values. For each of these thresholds you look at all the individual ROC-curves. You then average out the TPR and FPR of the value of the ROC-curves with a threshold closest to the current threshold value.

6 Bounds without dimensionality reduction

In this section, we look at the relevance bounds we get without reducing the dimensionality of the data sets. For each data set we discuss the error bounds.

6.1 Simple data set

First we look at the relevance bounds for the simple data set. In figure 10 we see the relevance bounds for different error margins for the accuracy.

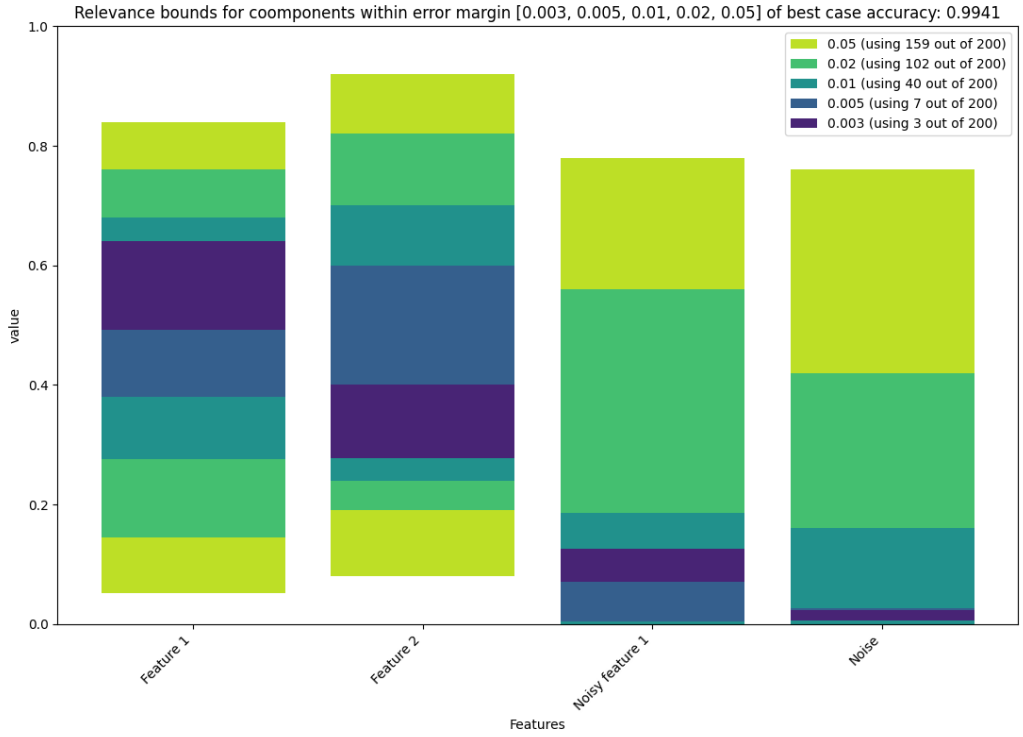


Figure 10: Relevance bounds of the full simple data set.

The intervals for the smaller error and quite thin. This is due to the low number test that are within this error bound. Looking at the legend of figure 10 we see there are only 3 (1.5%) are within 0.3% accuracy and 7 (3.5%) are within 0.5%. We see the bounds of feature 1 and 2 are most relevant, which is what we expect. When looking at the 1% error margin we see that the ranges of valid relevances are roughly 0.3-0.7.
When looking at the minimal relevance of the noisy copy and the noise feature, we see that both start at 0. This tells us that the relevance of these features can be set to 0 without significantly impacting the performance. For the noise feature

this makes sense. Not including the noise feature should not impact performance. When the noisy copy or the noise feature have a high relevance, we see a decrease in accuracy. This tells us the classification is relatively fragile. If the noise gains some relevancy, it immediately causes some additional misclassification. This can be explained by the closeness of the 2 classes. Looking at the data set in figure 6 we see that there are a lot of points close to the boundary. Even if the boundaries shift slightly, it already causes a significant number of misclassifications.

6.2 Blob data set

The blob data set has its classes more separated. If we look at the data set (Figure 7) we see some clear separation between the classes. The classes overlap near the boundary, which makes a 100% classification accuracy near impossible to achieve. The prototypes for this data set. We see the classes are not completely separated. In figure 11 we see the relevance bounds of this data set.

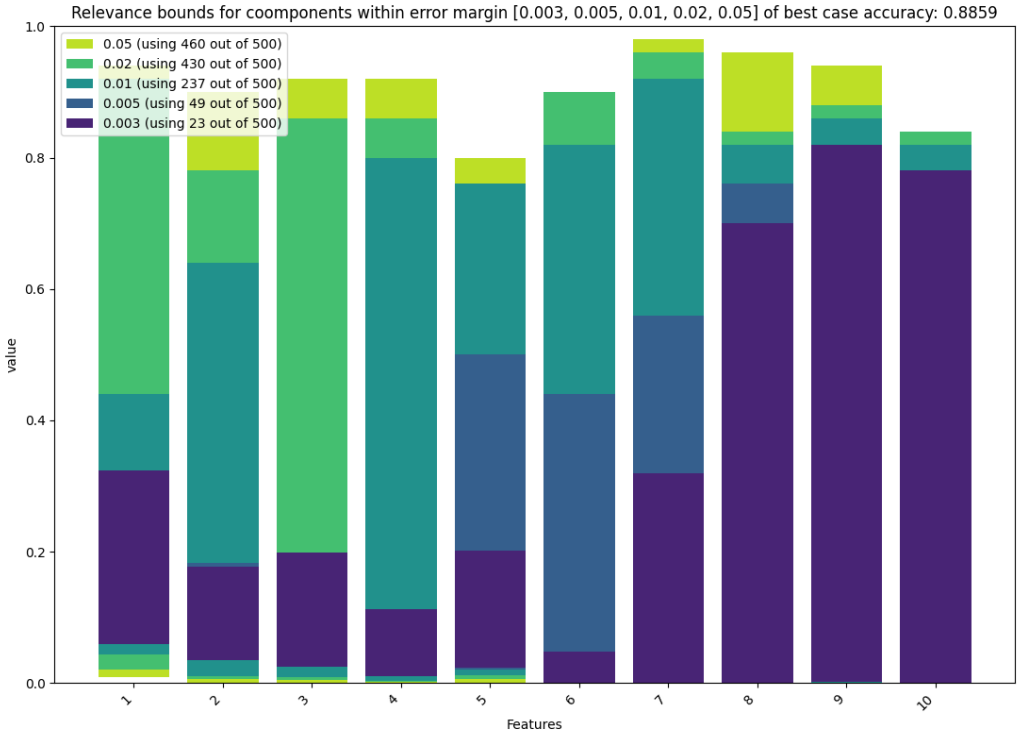
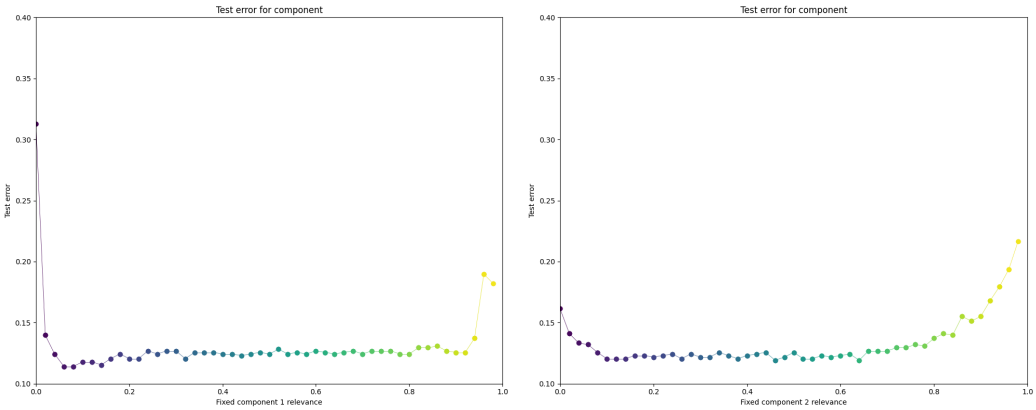


Figure 11: Relevance bounds of the blob data set.

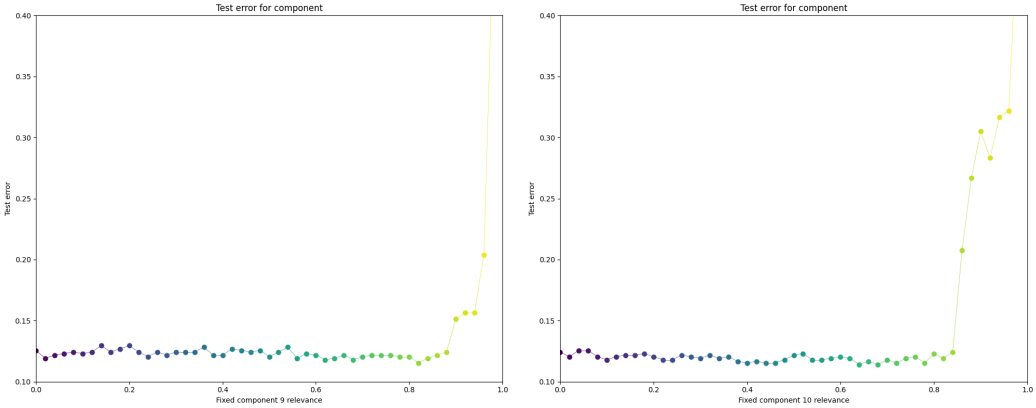
In these relevance bounds we see different behaviour. The first thing we see here is that the minimum relevances are significantly lower, even for the features

that we expected to be relevant. An interesting effect can be seen in the noise, where most of the interval is able to perform well. To explain this, we need to look at the involved error curves. Below are 2 Figures 12,13 showing 2 relevant and 2 noise features.



(a) Error curve of feature 1 (b) Error curve of feature 2

Figure 12: Error curves for the first 2 relevant features, 1 and 2



(a) Error curve of feature 9 (b) Error curve of feature 9

Figure 13: Error curves for 2 irrelevant features, 9 and 10

Here we see the error curves are almost entirely flat. For the relevant features, we see mild dips for the lower values, increasing only once the relevance is forced to high values. The irrelevant features are mostly flat. Most variance can be explained with errors due to randomized effects. There are some conclusions we can make from these graphs. Firstly, of all relevant features, it seems like feature 4

is the least useful. It has the lowest minimal relevance and performance decreases once it becomes more relevant. Noise feature 6 seems, by pure luck, to be the most harmful noise feature. In general, this problem is resilient to the noise features. Performance barely drops even if the relevances of the noise becomes very high.

6.3 Transformed blob data set

The transformed blob data set is more complicated than the normal blob data set. Each blob has a different shape. This means that the important features for each class may differ. The bounds we find after training are shown in figure 14.

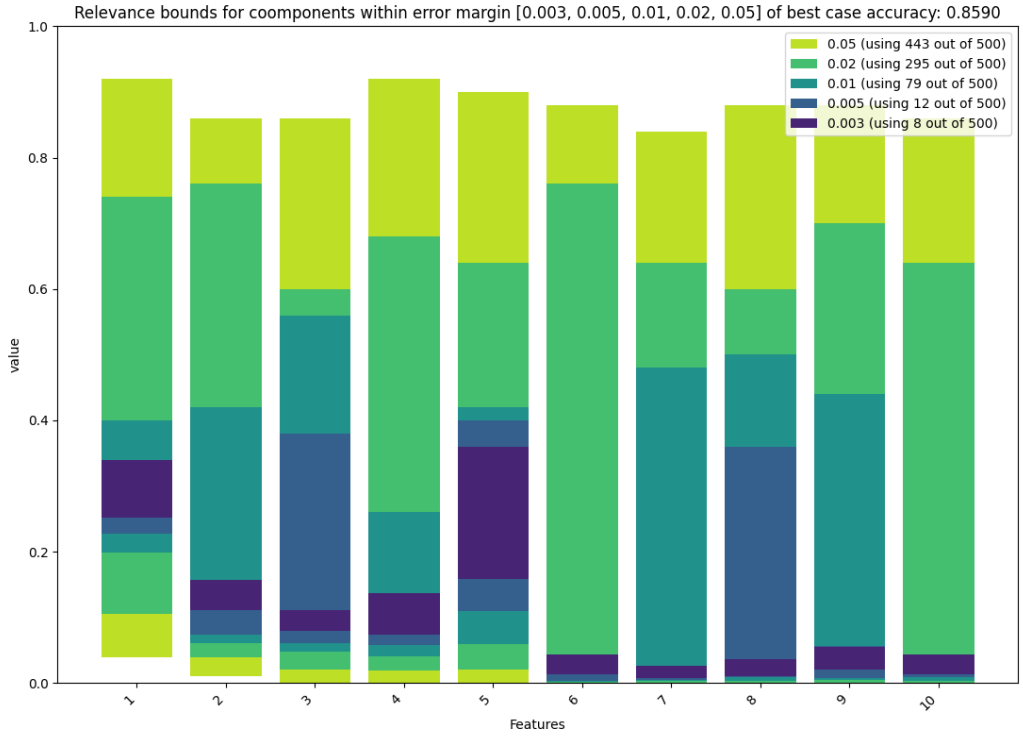


Figure 14: Relevance bounds of the transformed blob data set.

The 5 features that were used to generated blobs all have a higher minimal relevance than the 5 noise features. Feature 1 and 5 seems to have the most impact on the classification. We also see the error bounds for the noise features are very different. While minor relevances in the noise are still tolerable, performance quickly drops when the noise features are forced to be relevant.

6.4 KiDS-GAMA data set

With the KiDS-GAMA data set, the main disadvantage of this method becomes very apparent. As the method brute forces all different combinations of features and fixed values. This significantly increases the number of trainings that have to be done if the number of features increase. The total number of trainings that have to be done are

$$\text{Features} \cdot \text{Number of fixed relevance values} \tag{31}$$

For the blob data set this is $10 \cdot 50 = 500$ total trainings. The KiDS-GAMA data set increases this values to $37 \cdot 50 = 1850$, which is already 3.7 times as high. GMLVQ itself is also affected by the dimensionality. Both the increased number of features and the increased number of fixed values cause the time required to increase. This causes the computational time to become too high for this data set. For this reason, we investigate techniques to reduce the number of features we need to train before we can find the relevance bounds.

7 Comparing dimensionality reduction

For dimensionality reduction, two methods are compared. PCA is compared to the Λ-transformation by looking at their performance using different numbers of features in the reduced data set.

7.1 Principal Component Analysis

We are comparing the performance of GMLVQ classifier with different number of PCA components. To do this, we measure the performance of the classifier for different number of PCA components used. We first look at the simple data set. For this data set we should be able to reduce the data set down to 2 features. In table 1 we see the results of the tests. We see the best performance is achieved when using all 4 PCs. When lowering it down to 2 PCs the accuracy also takes a significant hit.

components	4	3	2
acc (%), PCA	99	96	92

Table 1: The accuracy when changing the number of PCA components used during training of the simple data set.

The blob data set has 5 clearly relevant features and 5 noise features. We would expect to still see a decent performance with 5 features. In table 2 we can see the accuracies.

components	10	9	8	7	6	5	4	3	2
acc (%), PCA	88	87	88	88	83	82	83	83	80

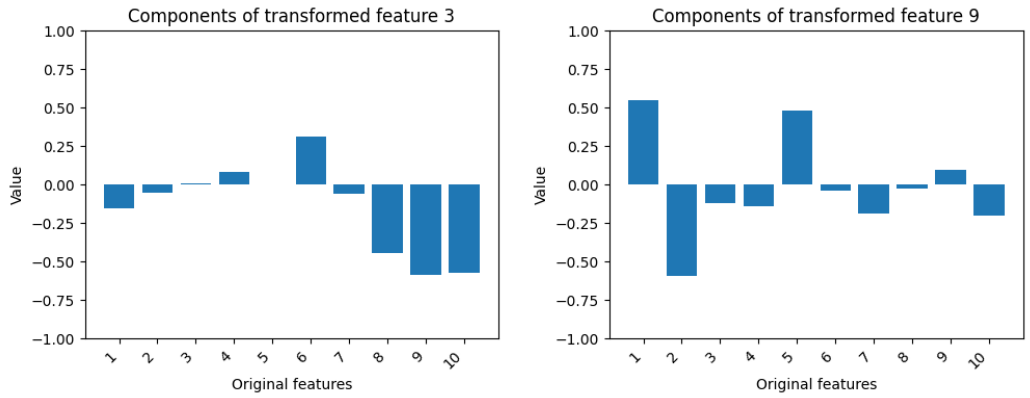
Table 2: The accuracy when changing the number of PCA components used during training of the blob data set.

For these two data sets the accuracy after the PCA drops, but the drop-off is not very high. However, if we take a look at one of the Blob variants this changes. If we take the transformed blobs we see a more significant impact as seen in table 3. We see here that certain PCs contain important information. We see a drastic drop between 8 and 9. A similar decrease can be seen between components 3 and 4. This implies that PC 4 and 9 seem to hold significant information for classification that is not held in the features with higher explained variance.

components	10	9	8	7	6	5	4	3	2
acc (%), PCA	85	83	70	70	71	70	70	60	59

Table 3: The accuracy when using the PCA components with highest explained variance during training of the transformed blob data set.

To show this is not the optimal way to choose our feature we can consider the following. If this method results in the best feature to use, we should not be able to find other combinations of features that perform better. To do this we look at 3 features. In table 3 we see this gave us an accuracy of 60%. As previously explained, feature 4 and 9 seem important. In addition to this, we add feature 1 to get our three features. If we train strictly on these features we get an accuracy of 77%. This significantly outperforms using the 8 features with the highest explained variance. This shows the choice of PCs is not optimal. When we look at the linear transformations of these PCs in figure 15, we see a feature with a high explained variance but irrelevant information or vice versa. All other PCs can be found in Appendix A.



(a) PC with high explained variance but little relevant information

(b) PC with low explained variance but contains relevant information

Figure 15: Example of two different PCs

7.2 Λ-transformation

We can apply the same method for the Λ-transformation. By training once beforehand we can get Λ. We use it to reduce the number of features in the same way as we did with PCA. For this, we use the exact same data sets. The results of this for the simple data set can be found in table 4. Here we see that the performance stays quite high. Staying near perfect even when reduced to 2 components.

components	4	3	2
acc (%), Λ	99	99	98

Table 4: The accuracy when changing the number of Λ-components used during training of the simple data set.

Now we look at the blob data set. PCA kept a decent performance when lowering the number of features. In table 5 we can see the accuracies for the Λ-transformation. Here we are starting to see a clear difference.

components	10	9	8	7	6	5	4	3	2
acc (%), Λ	88	88	88	88	87	88	88	89	89

Table 5: The accuracy when changing the number of Λ-components used during training of the blob data set.

Our third data set, the transformed blobs, have a very significant change in performance. In table 6 we see performance is maintained even when significantly lowering the number of components. Unlike PCA, where the 9th component was very relevant, the Λ-transformation find only 3 components that are very relevant.

components	10	9	8	7	6	5	4	3	2
acc (%), Λ	84	85	85	85	86	86	86	86	80

Table 6: The accuracy when changing the number of Λ-components used during training of the transformed blob data set.

These test confirm our intuition. When we create a transformation for the data set using the information gained from a prior training, we are able to reduce the number of dimensions significantly better than we could when using PCA. From now on, we use the Λ-transformation as our method to reduce dimensionality.

8 Properties of the Λ-transformation

In the previous section we have seen the performance of the Λ-transformation in comparison to the PCA transformation. In this section we look further into effects of the Λ-transformation on the calculation of the relevance bounds.

8.1 Λ matrix after transformation

One of the major differences of the Λ-transformation when comparing it to PCA is that the Λ-components are chosen in such a way that they are the features that are relevant. As these components are the components we got from training we think of them as the optimal features to consider. If we use entire Λ_c to perform our transformation, we see something interesting happen which confirms this idea.

When we perform the Λ-transformation of the transformed-blob data set and train on the X^* we see the relevance matrix turn into an almost diagon matrix (figure 16). As the Λ-components are all orthogonal and all original information is still there, we expect there to be very little interplay between the features. On the diagonal of this matrix are the eigenvalues that are associated with each eigenvector.

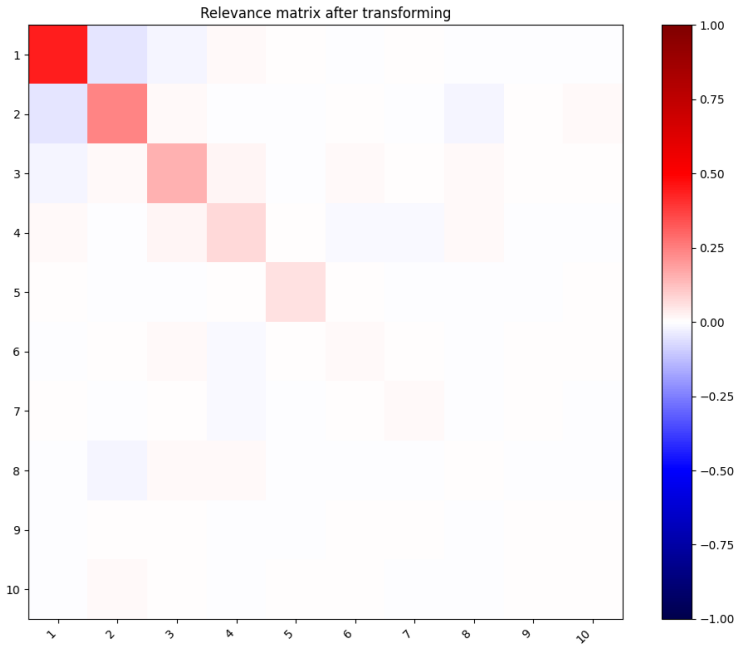


Figure 16: Relevance matrix after training the transformed data set.

We can repeat this process using fewer features. When we reduce the number

of features to the first 5 Λ-components we see more values appear on the off-diagonal (Figure 17). While the main diagonal is still the most relevant, the effects of the features on the main diagonal are compensated by the other features.

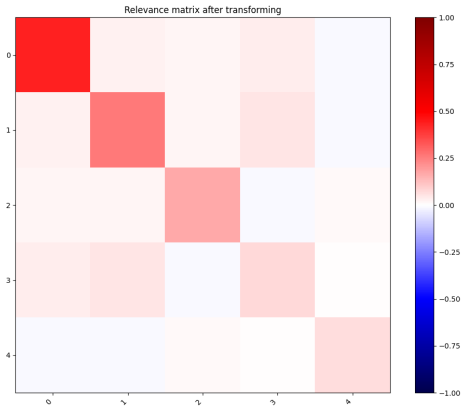


Figure 17: Relevance matrix after training the transformed data set with 5 features.

These matrices can be transformed back into the original space. Ideally, these matrices are near identical to the Λ matrix that was obtained when training on the original data set without any dimensionality reduction (Figure 18). This matrix is very similar to the matrices obtained when inverting the Λ-transformations with different number of features as seen in Figure 19.

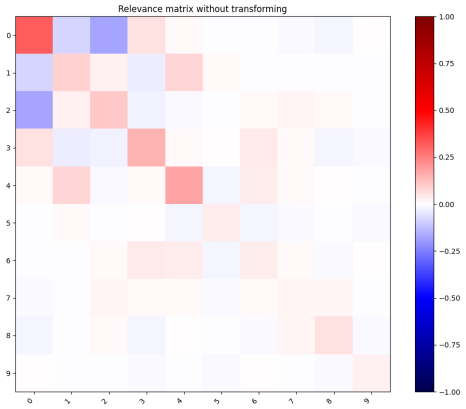
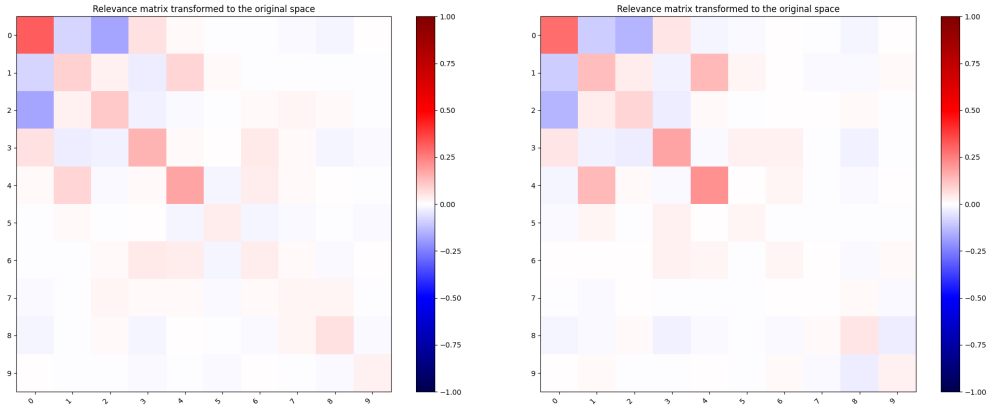


Figure 18: Relevance matrix after training the transformed-blob data set.



(a) Λ^* transformed back into the original space of Λ-transformation (10 features).

(b) Λ^* transformed back into the original space of Λ-transformation (5 features).

Figure 19: Relevance matrix with different number of Λ-components for the transformed-blob data set.

We can see that even with a reduced number of features our final result will be the same relevance matrix. This tells us that the method is effective at reducing the dimensionality without significantly changing the results. Using this new transformation we can also calculate the relevance bounds. To do this we take the resulting relevance matrices we get from training after the transformation and apply the inverse transformation to get the original relevance bounds which can be seen in figure 20.

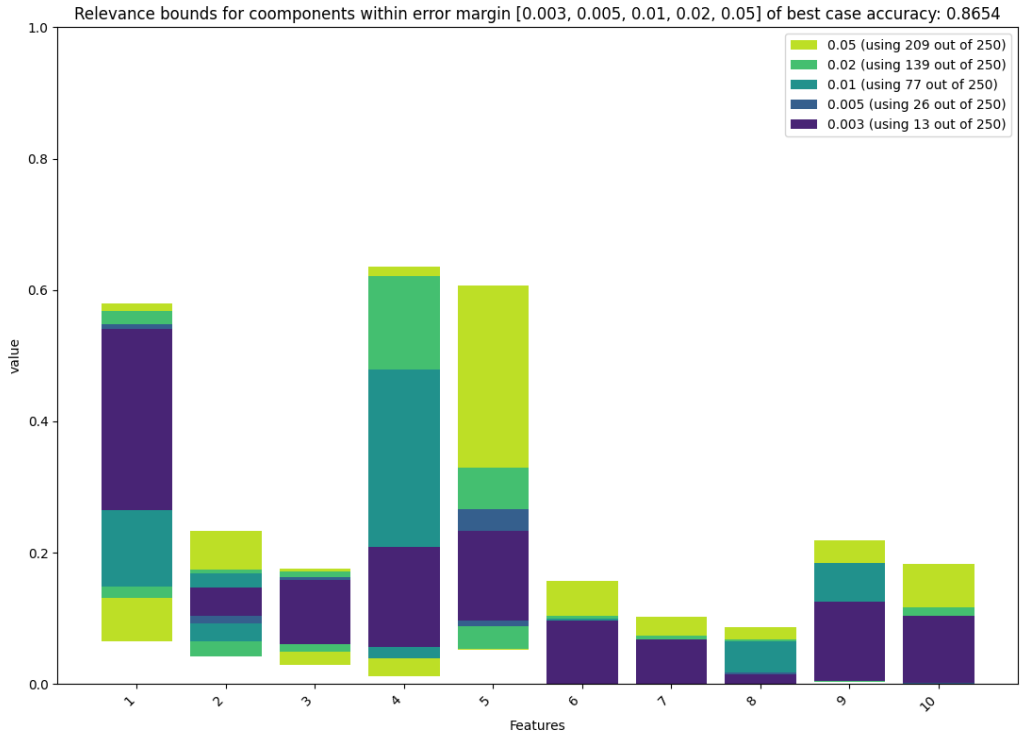


Figure 20: Relevance bounds of the transformed-blob data set using 5 Λ-components.

8.2 Eigenvalues and eigenvectors

The choice of features in the Λ-transformation is dependent on the eigenvalues. The eigenvalues correspond to the relevance of the feature created using the associated eigenvector. In table 7 we see the accuracies of the transformed-blob data set and the eigenvalues.

components	1	2	3	4	5	6	7	8	9	10
acc (%)	-	80	86	86	86	86	85	85	85	84
eigenvalues	0.428	0.238	0.169	0.097	0.056	0.007	0.003	0.002	0.001	0
Σeigenvalues	0.428	0.665	0.835	0.932	0.987	0.994	0.997	0.999	1	1

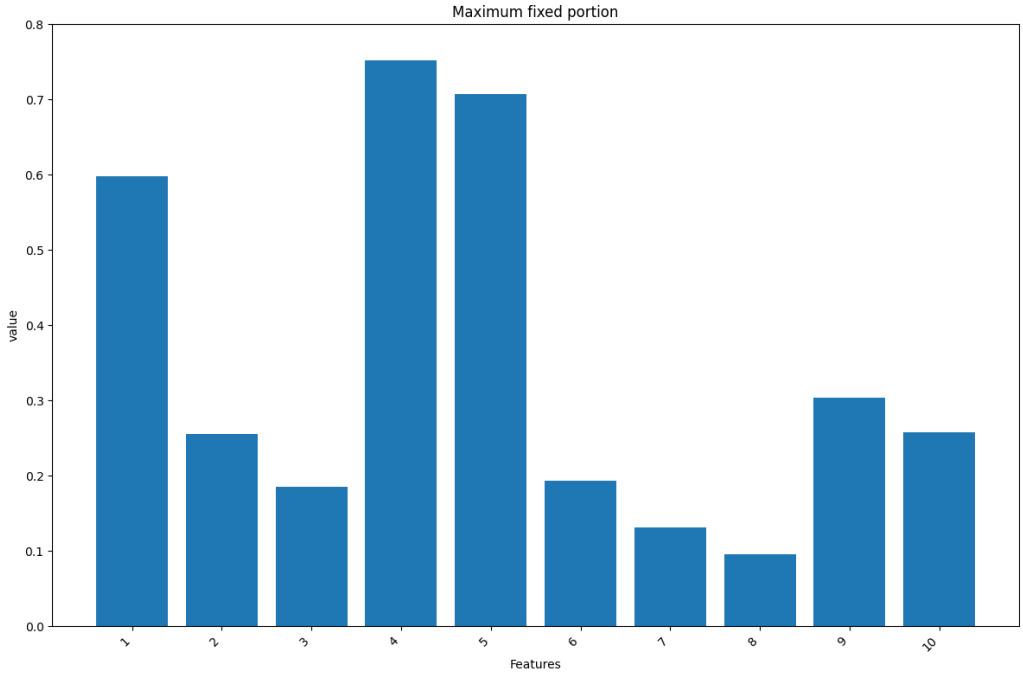
Table 7: The accuracy when changing the number of Λ-components used during training of the transformed blob data set with corresponding eigenvalues

Here we see the total eigenvalue needed for a good performance. We see that once the sum of eigenvalues exceeded 0.8 we are able to get a decent accuracy.

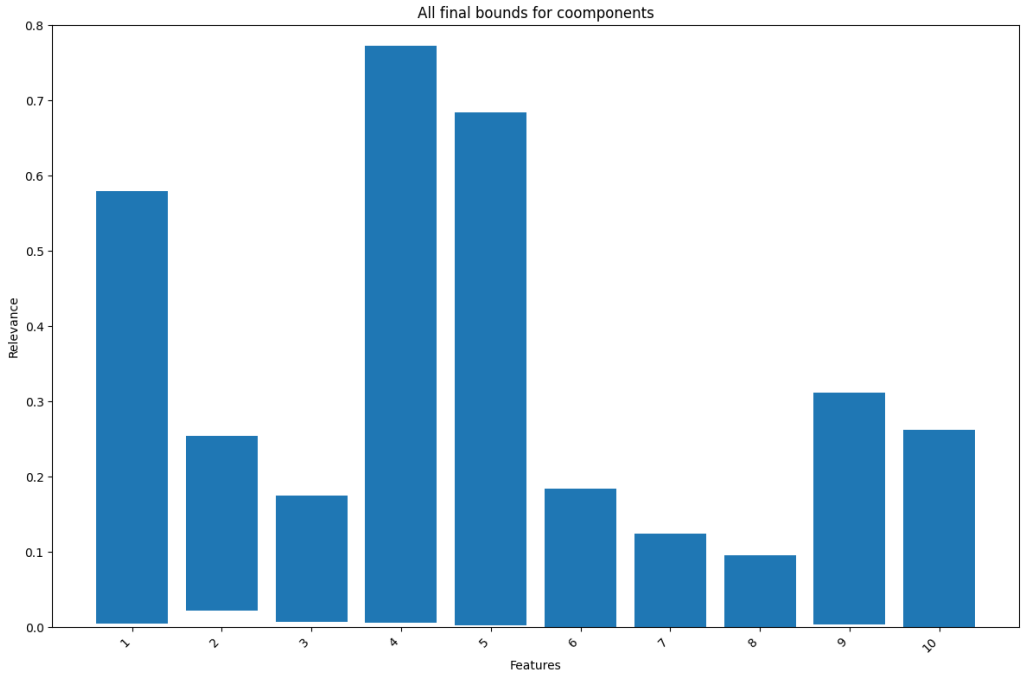
We must consider that accuracy is not the only thing we are looking for. Our goal is to find the relevance bounds.

8.2.1 Fixed space

We are not looking for just the best classifier, but at many configurations that have differing performance. For this reason, it is important to consider enough features which we can constrain. If we do not choose enough Λ-components then there will not be enough diversity among the features when we fix the values. In addition to this, if there are no Λ-components that significantly represent a feature from the original space, then it will never be fixed to a higher relevance. This means that, unlike the method without dimensionality reduction, not every original feature has their relevance fixed to a large range of values. We can use two different graphs to show the values that have been seen during the training. First, we can graph the maximum value of that each feature will be set to with a fixed relevance. To calculate this we simply take the maximal value for each of the original features in the used Λ-components. The second graph we can view the range of values we have attained after training. We can compare this to the relevance bounds to see if we have a significant space where no good classifiers could be trained.
Below are these graphs for the transformed-blob data set (Figure 21). When we compare these to figure 20 we see that there is a significant space that is covered by the fixed relevances which is outside the relevance bounds.



(a) The maximum value each original feature is fixed to.



(b) All resulting relevances from the all tests.

Figure 21: Transformed-blob data set with 5 Λ-components.

This means that our choice of number of features has a significant effect on the outcome. More features gives us a wider range of relevances that are explored but comes at a great computational cost. To make this decision there are two main methods. The first method is to discard ant Λ-components with an eigenvalue below a certain threshold. The second method is to add Λ-components until the sum of their eigenvalues exceeds a threshold. In our experiments, using the second methods with a threshold of 0.95 both reduces our data sets enough to make this method computationally viable but leave enough features to explore a significant amount of the relevances.

8.3 Correlated features

If we want to explore a large range of values for the relevance of a feature we need that feature to be represented significantly in a Λ-component. One of the properties of data sets that can complicate this is correlation. Correlated features tend to go together in the Λ-matrix. For example, imagine a feature with a relevance of 0.5. If we take that same data set and add 4 copies of that feature to it, they will not all have a relevance of 0.5 as the total relevance would exceed 1. The relevances of those features are roughly 0.1. The Λ-components that include one of those features often also include the copies. This causes each individual feature to be less represented in the Λ-component. Because of this, they are also be fixed to a smaller range of values.

8.4 Null space correction

In section 3.5 we have seen an additional effect correlated features can have. The null space correction can correct these false relevances. To see this we take a look at the blob data set. To this data set we add 5 copies of the first feature with a small noise added some Gaussian noise, $\mathbf{N}(0, 0.02)$. We can train on this data set and see the relevances with and without null space correction.

Feature	Original	$Copy_1$	$Copy_2$	$Copy_3$	$Copy_4$	$Copy_5$
Without correction	0.012	0.027	0.021	0.009	0.015	0.017
With correction	0.011	0.011	0.012	0.012	0.011	0.010

Table 8: Relevances when training with and without null space correction

In table 8 we see the relevances change quite a bit. Without the correction the relevances of the features span a range between 0.009 and 0.027. In contrast, when applying the null space correction the relevances become very close to each other,

as we would expect. We can see the cause of this when looking at the eigenvalues of $XX^\top$ and the correction matrix.

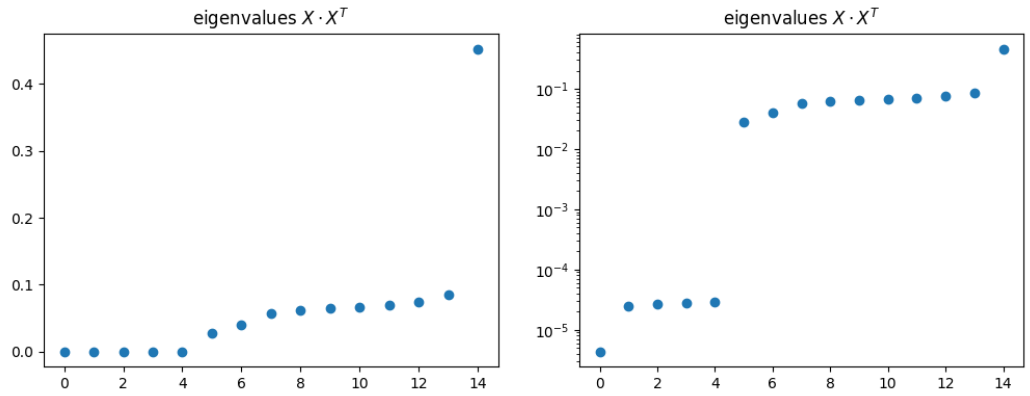


Figure 22: Eigenvalues of $XX^\top$ used to determine the null space.

In figure 22 we see the eigenvalues of $XX^\top$ divided by the total eigenvalues. We see a clear difference between the first 5 and the others. 5 eigenvalues are significantly lower then the rest. These 5 are small enough to consider inside the null space. With these can compute the null space correction matrix (Figure 23). Here we can clearly see how the null space is corrected. We see a 1 for each feature which are not copied and roughly $1/6$ for the copied features. This means X_1 and their copies get averaged at each step of the training.

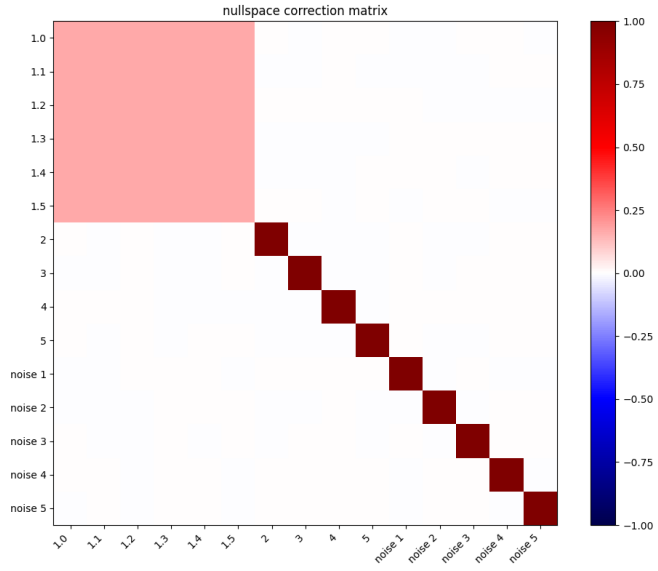


Figure 23: Null space correction matrix of the blob data set with added copies of the first feature.

9 Experiments after Λ-transformation

9.1 Simple data set

We look at the simple data set. As expected, this data set is mostly represented by one Λ-feature that shows the relation between the first two features. This feature has most of the relevance (0.94). The results can be seen in figure 24.

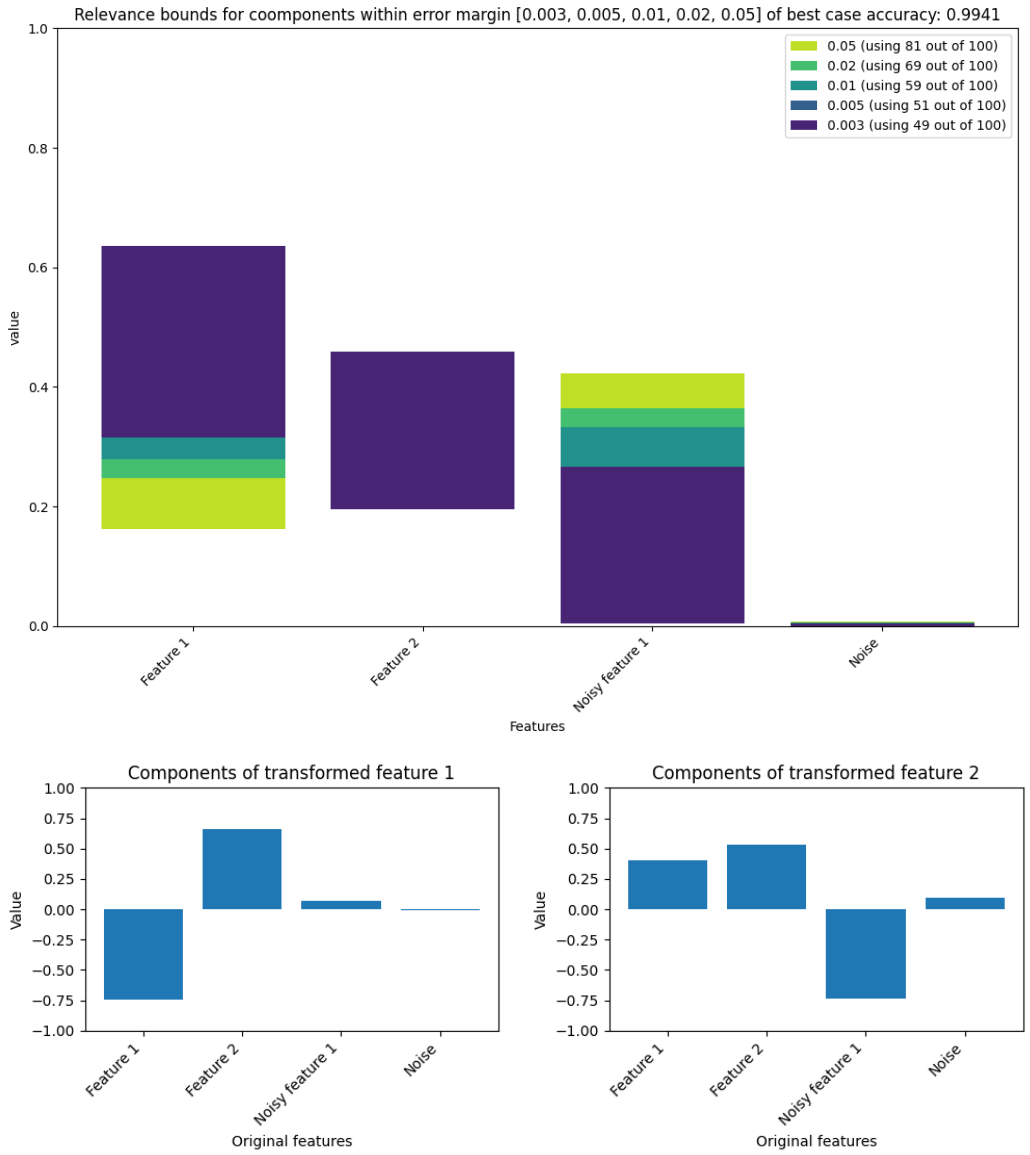


Figure 24: The relevance bounds and the first two Λ-component

We see the effect of a small number of Λ-components. There are just 2 components used here. Because of this, there is not enough variation possible. The noise feature is almost completely ignored as it is not represented in the Λ-components.

9.2 Transformed Blob data set

We have already seen the results of the transformed blob data set in figure 20. Here we see some differences with the original relevance bounds in figure 14. We see a slight increase for the best accuracy. This can be explained by two factors. These results have gone through two separate trainings. We train to compute the Λ-transformation and then we train in the transformed space. In total, it has had more iterations to train. In addition to this, the second training is done in a smaller space. This makes the space to traverse much smaller and can cause it to converge more quickly. In addition to the accuracy, we also see wider optimal bounds (dark blue) but smaller bounds overall. The wider optimal bounds are mostly due to the first few Λ-components which have a wider range of optimal values.

9.3 KiDS-GAMA data set

With these new tools at our disposal, we can start experimenting on the KiDS-GAMA data set. As mentioned previously, it is not feasible to calculate the relevance bounds of the full data set. This means we are using the Λ-transformation to reduce the dimensionality of the data set. The first three features of the data set after the transformation can be seen in Figure 25.

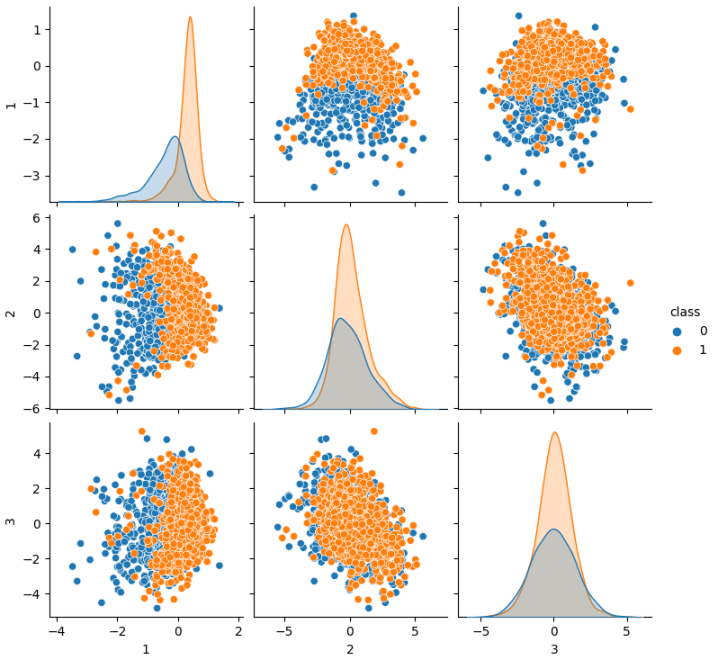
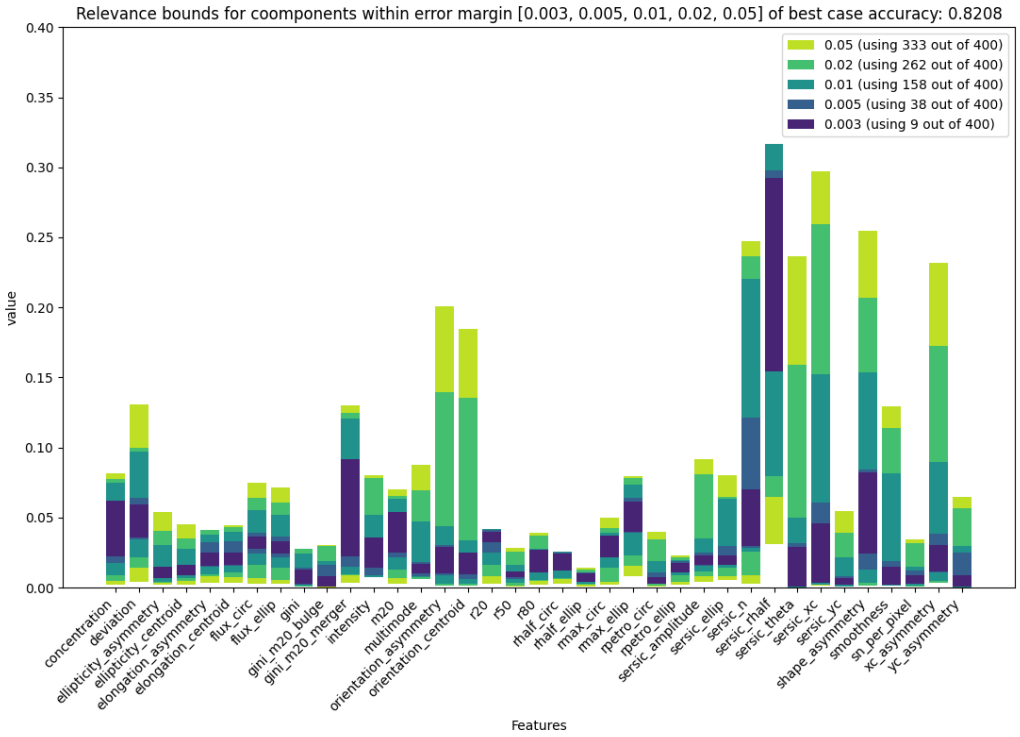
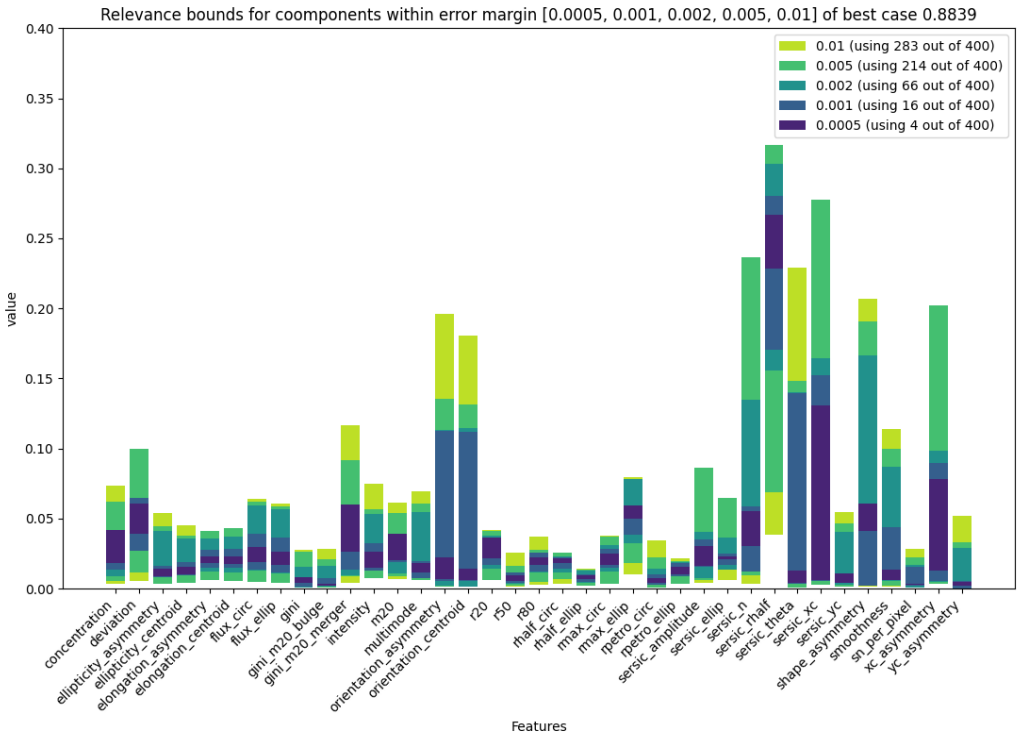


Figure 25: features, X_1^*, X_2^* and X_3^* of the KiDS-GAMA set after the Λ-transformation.

Using a sum of eigenvalues of 0.95 we reduce our data set from 37 components to 8 components. The eigenvalues of these components are: 0.715, 0.097, 0.055, 0.042, 0.022, 0.016, 0.006 and 0.006. Using the Λ-transformation, we can create the relevance bounds for using the accuracy or the AUROC as metric. These can be seen in figure 26a and 26b respectively.



(a) Relevance bound of the KiDS-GAMA data set using the accuracy.



(b) Relevance bound of the KiDS-GAMA data set using AUROC.

Figure 26: KiDS-GAMA data set with 8 Λ-components.

If we look at the eigenvalues one thing really stands out. There is one eigenvalue that is significantly higher than the rest. This implies that the data set can be reduced to a very small number of features while still retaining a high accuracy. When we reduce the number of features to 3 we see this effect. With 3 features we are still able to achieve an 82% accuracy. For the reasons mentioned in section 8.2.1 we still use 8 features.

9.4 Correlations within KiDS-GAMA data set

Just like some of our artificial data sets, the KiDS-GAMA data set also has a significant amount of correlations between features.

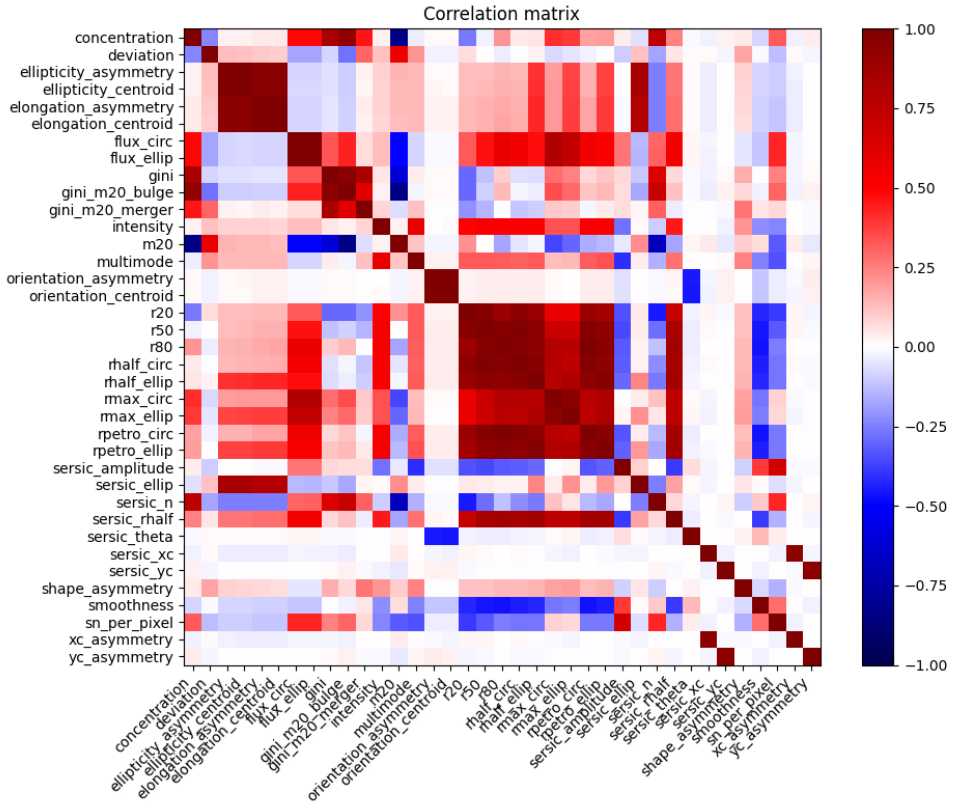


Figure 27: Correlation matrix

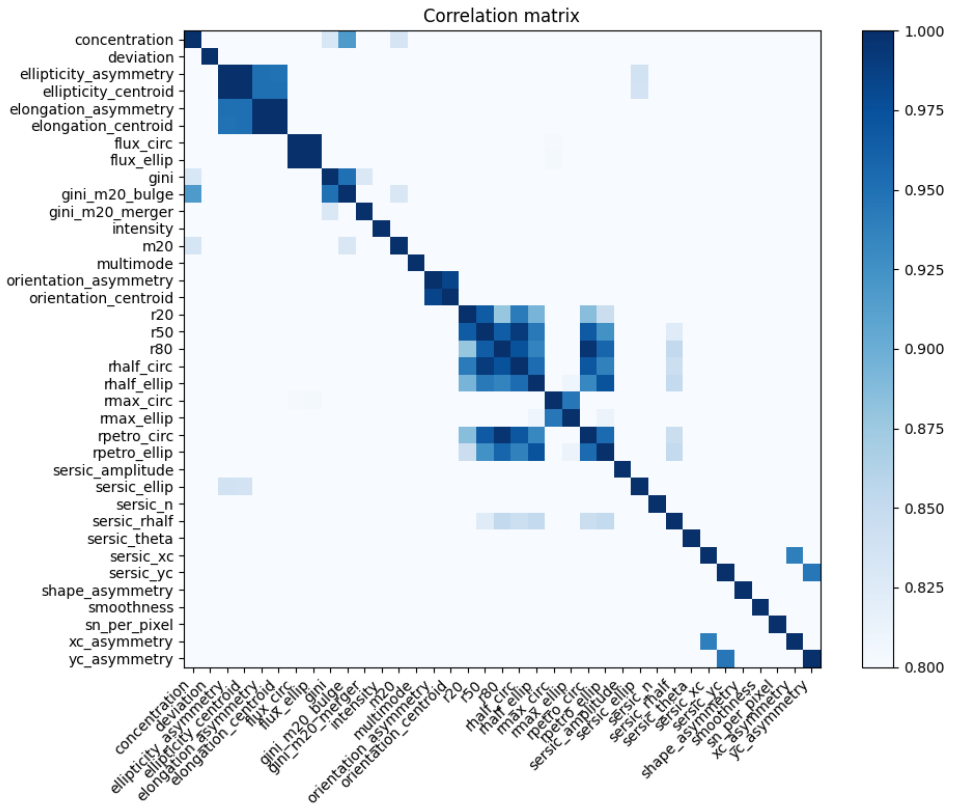


Figure 28: Absolute value of the correlation matrix > 0.8

In figure 27 and 28 we can see the different blocks of correlated features. We see a few groupings with features that are highly correlated to each other. We would expect these to influence our process. To see the effect this has on the resulting relevance bounds we look at the first block of 4 features which involve the ellipticity and elongation. These features are highly correlated. We compared the relevance bounds of the normal method to training on the same data set, but with 3 of the 4 features removed. In figure 29 we see a big difference. Removing the correlated features give us the knowledge that even with the elongation_centroid feature relevance at 0.10 we can still get respectable results. When using the full data set this configuration was never attained as the Λ component which has the elongation_centroid in it also had the other 3. The only situations where this can happen is when all 4 features have their relevance fixed to a high value or if a Λ component is added which can differentiate them. Those Λ components have a very low eigenvalue, and adding them would increase the computational effort. This effect is less pronounced for the lower bounds, which is the strongest indicator of e necessary feature.

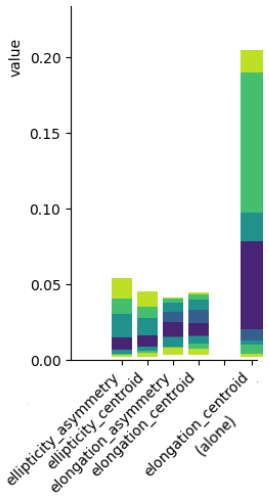
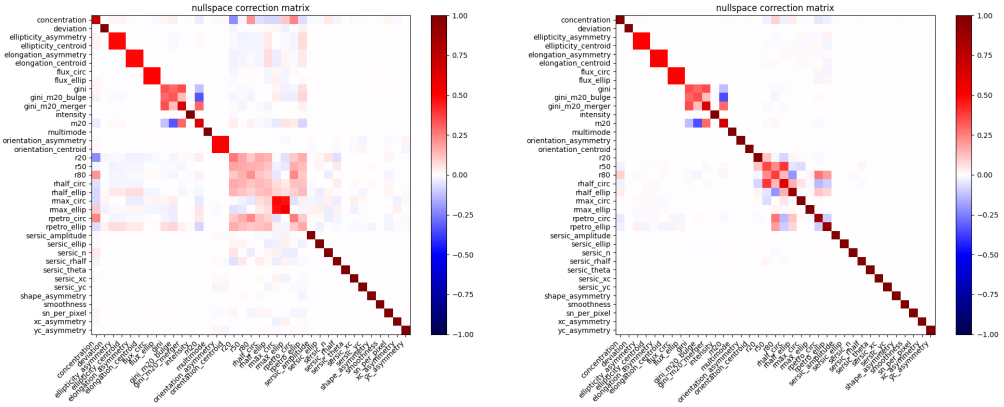


Figure 29: Effect of correlated features in the data set

We can use null space reduction to reduce the influence of the correlated features. For the null space correction matrix, we consider three thresholds: $10^{-4}, 10^{-3}, 10^{-2}$. Our first test shows that $t = 10^{-2}$ reduces the accuracy to 0.75. This means this threshold is too loose and considers relevant data is in the null space. For the remaining thresholds, we can see the correction matrices in figure 30.



(a) Null space correction matrix for $t = 10^{-3}$. (b) Null space correction matrix for $t = 10^{-4}$.

Figure 30: Null space correction matrix for the KiDS-GAMA dataset.

These correction matrices have 2 interesting qualities. Firstly, there seem to be several blocks of 2 features that are very related to each other. The second are 2 bigger groups, one being 4 features and the other being 7 features, which seem to be in the null space. The grouping of 2 features were already very visible

in the bounds we have previously seen and the feature names themselves. In the relevance bounds we see little difference. For $t = 10^{-4}$ we can see the relevance bounds in figure 31. The main effect seems to be an equalization of the features in the null space, which is expected. The relevant features also have a slightly increased range.

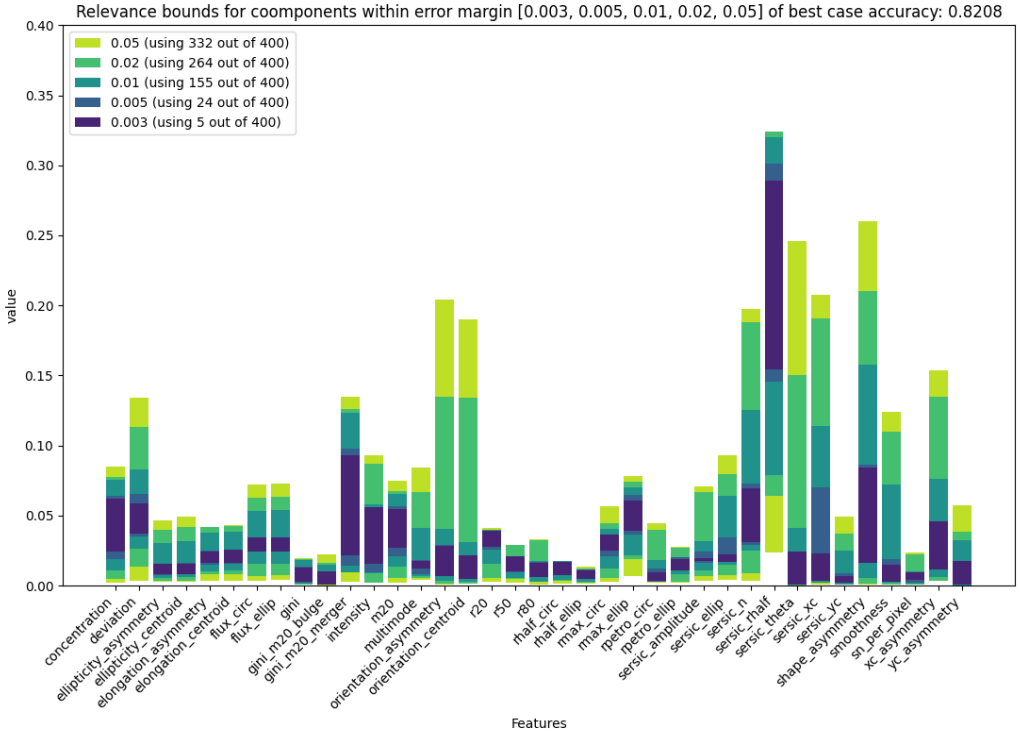


Figure 31: Relevance bounds using the null space correction with $t = 10^{-4}$

9.5 Removing sersic_rhalf

In section 5.1.4 we have given a basic overview of which features we chose to omit from the data set. The data set still hold a feature which may be troublesome. Sersic_rhalf is a feature that holds a high relevance. The measurement of Sersic_rhalf may be inaccurate for mergers. For this reason, we want to look at the same data set without Sersic_rhalf. The relevance bounds obtained can be seen in figure 32. The biggest difference can be seen in max_ellip. In our previous experiments this feature was quite relevant but did not stand out. After removing sersic_rhalf the relevance of max_ellip seems to have jumped up significantly, becoming the feature with the lowest minimum relevance. In other words, this feature has to be included in at least 0.05 relevance to get a within 1% of the best

case. The other features that seem to increase are all the other features that were correlated to sersic_rhalf.

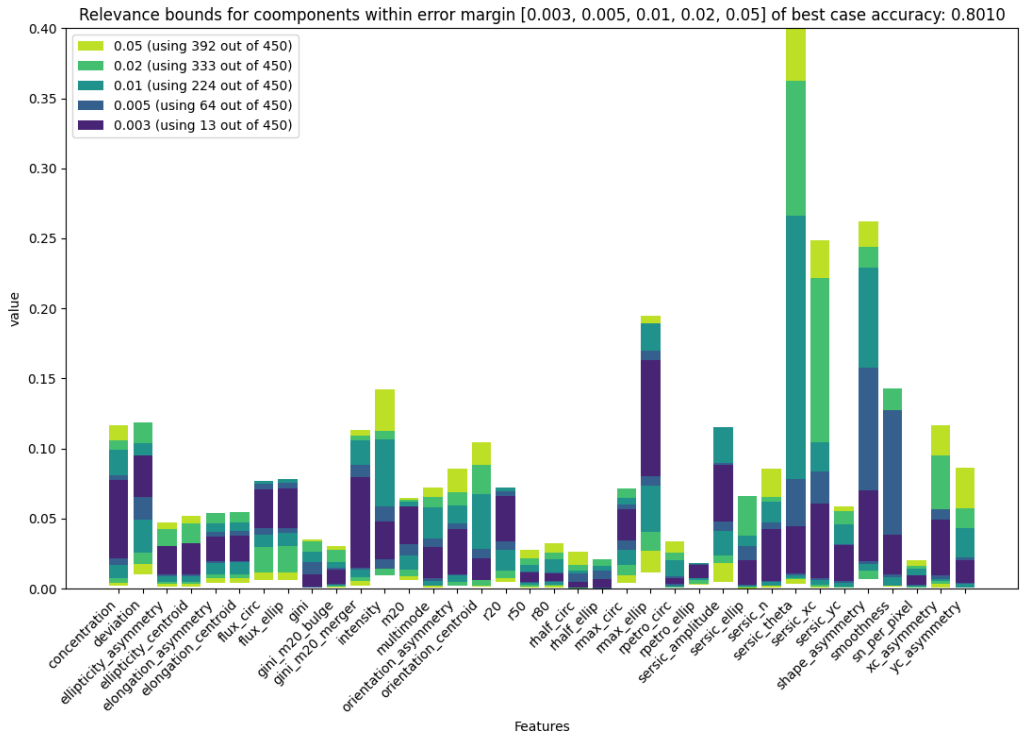


Figure 32: Relevance bounds for null space correction with $t = 10^{-4}$ without sersic_rhalf

10 Conclusion

In this thesis we have looked at a use case for the fixed relevance. We used fixed relevances to calculate relevance bounds in GMLVQ. With these relevance bounds we were able to gain additional insight into the data sets which could not be done using just GMLVQ. We have also investigated the main disadvantage of our method, the computational time. To reduce the computational time, we used two main methods to reduce the dimensionality. We compared PCA to the Λ-transformation for several data sets and shown the Λ-transformation to be highly efficient in reducing the number of features. Using this transformation, we were able to reduce the number of features of our mock data sets.
For the Λ-transformation we have determined the main disadvantages: irrelevant features and correlations. We have looked at null space reduction. Which is able to regulate some of the problems that stem from correlated features but was not able to fully eliminate the problems. Finally, we applied the method to a real-world data set. This data set was too big to use without a form of dimensionality reduction.
In this data set, we were able to significantly reduce the number of features. The data set went from 37 dimensions to 8 or 9 depending on the null space reduction.

10.1 Future Work

While the results shown are promising, the method currently has a number of problems that will need to be investigated. One of the primary hurdles for this method is the existence of correlated features. Currently, the only mitigating factor for correlated features is in the null space reduction. Both method used to reduce the dimensions have a tendency to group all correlated features together in the post-transformation features. We think there is a lot to gain from eliminating correlated features beforehand.
Currently, the method uses a grid search for our fixed feature values. While this is very exhaustive, it is a significant waste of resources. Instead of using a static grid of values for the fixed relevance we think there is much to gain by using a dynamic range of relevances based on the fixed relevance values that you have already tested. Using a dynamic range we can reduce the number of tests done for fixed relevances outside our margin of error.
With the Λ-transformation we can not constrain the relevance of the original feature as it may be represented by multiple Λ-components. One way this may be mitigated is by constraining multiple features at the same time. It may be possible to more strongly control the relevance of the original features if multiple relevant Λ-components are constrained at the same time.

Bibliography

[1] P. Schneider, M. Biehl, and B. Hammer, "Adaptive Relevance Matrices in Learning Vector Quantization," *Neural Comput.*, vol. 21, no. 12, pp. 3532–3561, 2009.

[2] B. Frenay, D. Hofmann, A. Schulz, M. Biehl, and B. Hammer, "Valid interpretation of feature relevance for linear data mappings," in *Computational Intelligence and Data Mining (CIDM), 2014 IEEE Symposium on*, pp. 149–156, IEEE (The Institute of Electrical and Electronics Engineers), Dec. 2014.

[3] A. Schulz, B. Mokbel, M. Biehl, and B. Hammer, "Inferring Feature Relevances From Metric Learning," in *2015 IEEE Symposium Series on Computational Intelligence*, pp. 1599–1606, IEEE, 2015.

[4] T. Kohonen, *Learning Vector Quantization for Pattern Recognition*. Report TKK-F-A, Helsinki University of Technology, 1986.

[5] M. Biehl, A. Ghosh, and B. Hammer, "Learning vector quantization: The dynamics of winner-takes-all algorithms," *Neurocomputing*, vol. 69, no. 7, pp. 660–670, 2006. New Issues in Neurocomputing: 13th European Symposium on Artificial Neural Networks.

[6] A. Sato and K. Yamada, "Generalized Learning Vector Quantization," in *Advances in Neural Information Processing Systems 8, NIPS, Denver, CO, USA, November 27-30, 1995* (D. S. Touretzky, M. Mozer, and M. E. Hasselmo, eds.), pp. 423–429, MIT Press, 1995.

[7] T. Bojer, B. Hammer, D. Schunk, and K. T. von Toschanowitz, "Relevance determination in Learning Vector Quantization," in *ESANN 2001, 9th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 25-27, 2001, Proceedings*, pp. 271–276, 2001.

[8] B. Hammer and T. Villmann, "Generalized Relevance Learning Vector Quantization," *Neural Networks*, vol. 15, no. 8-9, pp. 1059–1068, 2002.

[9] M. Strickert, B. Hammer, T. Villmann, and M. Biehl, "Regularization and improved interpretation of linear data mappings and adaptive distance measures," in *2013 IEEE symposium on computational intelligence and data mining (CIDM)*, pp. 10–17, 2013.

[10] T. Kohonen, "Improved versions of learning vector quantization," in *1990 IJCNN International Joint Conference on Neural Networks*, pp. 545–550 vol.1.

[11] B. Hammer, F.-M. Schleif, and T. Villmann, "On the generalization ability of prototype-based classifiers with local relevance determination," tech. rep., Clausthal University of Technology, 2005.

[12] K. Bunte, P. Schneider, B. Hammer, F.-M. Schleif, T. Villmann, and M. Biehl, "Limited rank matrix learning - discriminative dimension reduction and visualization," *Neural Networks*, pp. 159–173, 2012.

[13] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," Nov. 1901.

[14] H. Hotelling, "Analysis of a complex of statistical variables into principal components.," *Journal of educational psychology*, vol. 24, no. 6, p. 417, 1933.

[15] V. Klema and A. Laub, "The singular value decomposition: Its computation and some applications," *IEEE Transactions on Automatic Control*, vol. 25, no. 2, pp. 164–176, 1980.

[16] G. Papari, K. Bunte, and M. Biehl, "Waypoint averaging and step size control in learning by gradient descent," *Machine Learning Reports*, vol. 6, p. 16, 2011.

[17] R. van Veen, M. Biehl, and G.-J. de Vries, "sklvq: Scikit learning vector quantization," *Journal of Machine Learning Research*, vol. 22, no. 231, pp. 1–6, 2021.

[18] 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*, vol. 12, pp. 2825–2830, 2011.

[19] Kuijken, K., Heymans, C., Dvornik, A., Hildebrandt, H., de Jong, J. T. A., Wright, A. H., Erben, T., Bilicki, M., Giblin, B., Shan, H.-Y., Getman, F., Grado, A., Hoekstra, H., Miller, L., Napolitano, N., Paolilo, M., Radovich, M., Schneider, P., Sutherland, W., Tewes, M., Tortora, C., Valentijn, E. A., and Verdoes Kleijn, G. A., "The fourth data release of the kilo-degree survey: ugri imaging and nine-band optical-ir photometry over 1000 square degrees," *A&A*, vol. 625, p. A2, 2019.

[20] S. P. Driver, P. Norberg, I. K. Baldry, S. P. Bamford, A. M. Hopkins, J. Liske, J. Loveday, J. A. Peacock, and G. T. (listed below), "GAMA: towards a physical understanding of galaxy formation," *Astronomy & Geophysics*, vol. 50, pp. 5.12–5.19, 10 2009.

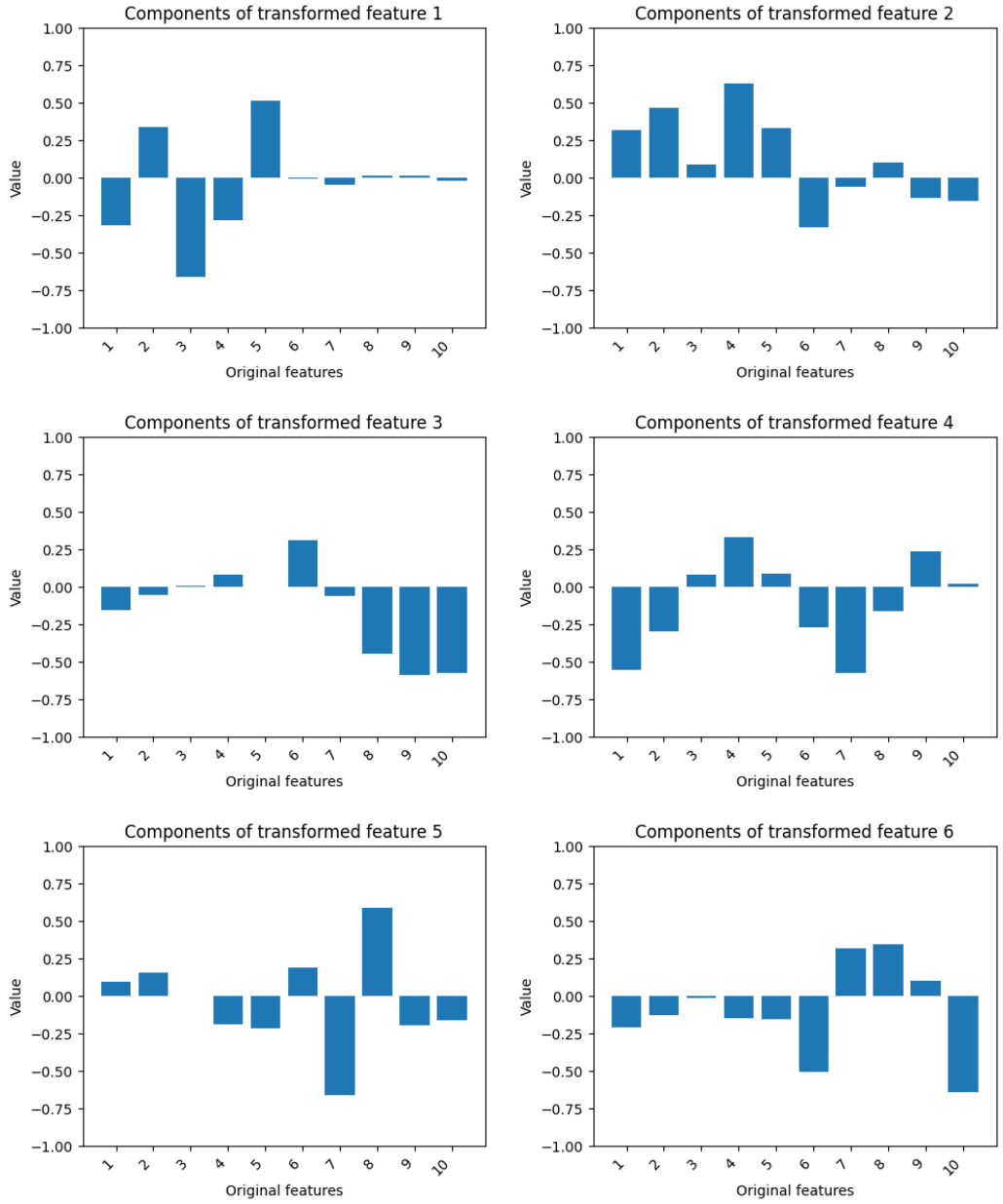
[21] W. J. Pearson, L. Wang, M. Alpaslan, I. Baldry, M. Bilicki, M. J. I. Brown, M. W. Grootes, B. W. Holwerda, T. D. Kitching, S. Kruk, and F. F. S. van der Tak, "Effect of galaxy mergers on star-formation rates," *aap*, vol. 631, p. A51, Nov. 2019.

[22] V. Rodriguez-Gomez, G. F. Snyder, J. M. Lotz, D. Nelson, A. Pillepich, V. Springel, S. Genel, R. Weinberger, S. Tacchella, R. Pakmor, P. Torrey, F. Marinacci, M. Vogelsberger, L. Hernquist, and D. A. Thilker, "The optical morphologies of galaxies in the illustristng simulation: a comparison to pan-starrs observations," *mnras*, vol. 483, no. 3, pp. 4140–4159, 2019.

[23] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006. ROC Analysis in Pattern Recognition.

Appendix

A Principle Components of transformed blob data set



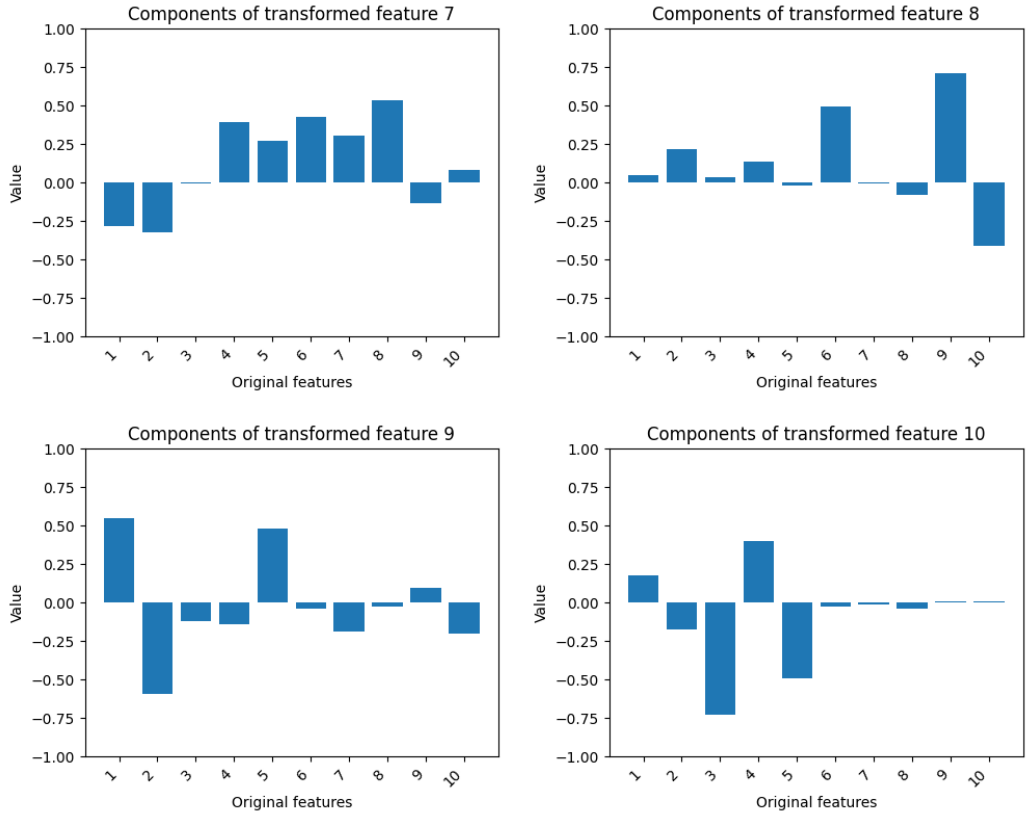


Figure 33: All principal components of the transformed blob data set