STUDENT'S T DISTRIBUTIONS IN LEARNING VECTOR QUANTIZATION

ANDO EMERENCIA

Abstract. The aim of this research was to find out how well Robust Soft Learning Vector Quantization (RSLVQ) based on Student's t distributions would perform in comparison to conventional Gaussian-based RSLVQ. Gaussian-based RSLVQ often incurs a great deal of information loss, using relatively few points to determine its decision boundaries. With their heavy-tailed property, Student's t distributions would take more points into account, at the same variance. For our research, we derived the update rule for Student's t-based RSLVQ, used this in a Matlab implementation of the algorithm, and tested its performance against Gaussian-based RSLVQ on several datasets. Furthermore, we approached the problem of "where prototypes converge to" theoretically, which gave insights as to which parameters influence prototype position and when poor performance is to be expected. Proving to be more robust, Student's t-based RSLVQ was able to slightly outperform Gaussian-based RSLVQ on a real-life dataset. Testset accuracies showed greater fluctuations for Gaussian-based RSLVQ. Of course, Student's t-based RSLVQ has disadvantages as well, one of them being poor performance in case the value for meta-parameter v (the degrees of freedom) was not chosen carefully. Our findings show that while best-case performance of Gaussian-based RSLVQ may very well be nearly optimal, worst-case (and thus average) performance can be improved by using a heavy-tailed distribution such as the Student's t. However, the measured differences on the real-life dataset were small, and further testing on other real-life datasets is needed to determine if Student's t-based RSLVQ is a consistent, significant improvement over Gaussian-based RSLVQ.

1. Introduction

Learning Vector Quantization (LVQ) is a supervised classification algorithm, originally proposed by Kohonen [Koho86][Koho90]. LVQ efficiently represents data of different classes using one or more prototypes – which have the same dimensions as samples in the dataset. During training, the algorithm iterates over a given set of labeled samples, which causes continuous updates to the prototypes. After training, unlabeled samples can be classified by nearest prototype. LVQ gives intuitive results and accuracies comparable to those achieved by computationally more expensive schemes.

Over the years, many LVQ adaptations and variants have been introduced. In this paper we use a statistical formalization of LVQ, called Robust Soft LVQ (RSLVQ) [Seo03][Schn09]. The motivation behind RSLVQ was to create an LVQ variant whose cost function could be derived mathematically, rather than being some intuitive heuristic. As such, the RSLVQ update rule aims to maximize the relative likelihood that the dataset is classified correctly, which is achieved through on-line gradient ascent (see e.g. [Duda01]) applied to the prototypes. Furthermore, unlike schemes such as LVQ 2.1 [Koho90], RSLVQ does not require a window rule to ensure numerical stability as this is done implicitly by the model.

Date: February–August 2009, Institute for Mathematics and Computing Science, University of Groningen.

Conventional RSLVQ models the dataset using a mixture of Gaussian distributions positioned at the prototypes. However, such a representation may not always be optimal. The tails of the Gaussian distribution go to zero relatively fast (exponential decay), so when e.g. classes are spread further apart or have a limited amount of points, updates depend only on a few outliers and prototypes start moving erratically.

Therefore, our idea is to use a distribution with heavier tails and find out if we get better results. In this paper we consider one such distribution: the Student's t distribution [Weis09]. Because our main goal is to determine the effect of using a heavy-tailed distribution rather than to attain high accuracies or intuitive prototypes, we use the most basic variant of RSLVQ, i.e. without any relevance learning.

The rest of this paper is organized as follows: in section 2 we derive the update rules for Gaussian-based and Student's t-based RSLVQ, which we then further analyze in section 3. Section 4 presents the results of testing the RSLVQ schemes on various datasets, and in section 5 we discuss our findings and conclude the paper. Section 6 suggests topics for future work.

2. Model

We identify the two RSLVQ variants used in this paper, which are named after the distribution used in their update rule. We call these RSLVQ variants the *Gaussian method* (or: Gaussian-based RSLVQ) and the *Student's t method* (or: Student's t-based RSLVQ). We refer to their update rules as the *Gaussian update rule* and the *Student's t update rule*.

In this section we first derive the update rules for both methods and then list the dataset assumptions that we use in our analysis (section 3).

2.1. Gaussian Method.

Since we limit ourselves to the application of RSLVQ without relevance learning, the distance measure we use is simply the Euclidean distance. The conditional densities $p(\boldsymbol{\xi}|j)$ of RSLVQ can be chosen to have the normalized exponential form [Schn09]:

$$p(\boldsymbol{\xi}|j) = K(j) \cdot \exp f(\boldsymbol{\xi}, \boldsymbol{\omega}_j, \sigma_j^2)$$

where $K(j)$ is a normalization factor, $\boldsymbol{\xi}$ is a sample in the dataset, $\boldsymbol{\omega}_j$ is a prototype, and σ_j^2 is the variance used in the update rule – where we assume that all prototypes use the same variance. The $f(\dots)$ term denotes the distance measure.

So to put an arbitrary probability density function $q(\dots)$ in this form, we simply set $K(j) = 1$, and:

$$f(\boldsymbol{\xi}, \boldsymbol{\omega}, \sigma^2) = \log q(\dots) \tag{1}$$

We are interested in the derivative of equation (1) with respect to $\boldsymbol{\omega}$, as that is what is used in the rule to update prototypes [Schn09].

In the basic case that the probability density of a class is described by a mixture of N-dimensional Gaussian distributions, we have that $K(j) = (2\pi\sigma_j^2)^{(-N/2)}$ and that:

$$f_{\text{gaus}}(\boldsymbol{\xi}, \boldsymbol{\omega}, \sigma^2) = -\frac{(\boldsymbol{\xi} - \boldsymbol{\omega})^T(\boldsymbol{\xi} - \boldsymbol{\omega})}{2\sigma^2}, \quad \text{so that } \frac{\partial f_{\text{gaus}}}{\partial \boldsymbol{\omega}} = \frac{1}{\sigma^2}(\boldsymbol{\xi} - \boldsymbol{\omega})$$

2.2. **Student's t Method.**
The Student's t distribution for $v = 1$ (degrees of freedom) is equal to the standard Cauchy distribution, and for $v = \infty$ it is equal to the standard Gaussian distribution; for any other v, we get something in between [Weis09].

The pdf of the Student's t distribution is given by:

$$q_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}, v) = \frac{\Gamma(\frac{v+1}{2})}{\sqrt{v\pi}\Gamma(\frac{v}{2})} \left(1 + \frac{(\boldsymbol{\xi} - \boldsymbol{\omega})^T(\boldsymbol{\xi} - \boldsymbol{\omega})}{v}\right)^{-\left(\frac{v+1}{2}\right)} \tag{2}$$

where Γ is the Gamma function.

So we have:

$$\begin{aligned} f_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}, v) &= \log q_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}, v) \\ &= \log \left[\frac{\Gamma(\frac{v+1}{2})}{\sqrt{v\pi}\Gamma(\frac{v}{2})} \left(1 + \frac{(\boldsymbol{\xi} - \boldsymbol{\omega})^T(\boldsymbol{\xi} - \boldsymbol{\omega})}{v}\right)^{-\left(\frac{v+1}{2}\right)} \right] \\ &= \log \Gamma(\frac{v+1}{2}) - \log \left[\sqrt{v\pi}\Gamma(\frac{v}{2})\right] - \left(\frac{v+1}{2}\right) \cdot \log \left[1 + \frac{(\boldsymbol{\xi} - \boldsymbol{\omega})^T(\boldsymbol{\xi} - \boldsymbol{\omega})}{v}\right] \end{aligned}$$

Differentiating with respect to $\boldsymbol{\omega}$ gives [Schn09]:

$$\frac{\partial f_{\text{stud}}}{\partial \boldsymbol{\omega}} = \frac{(v+1)(\boldsymbol{\xi} - \boldsymbol{\omega})}{v + (\boldsymbol{\xi} - \boldsymbol{\omega})^T(\boldsymbol{\xi} - \boldsymbol{\omega})} \tag{3}$$

Substituting equation (3) in the update rule for the prototypes gives:

$$\Delta\boldsymbol{\omega}_j = \frac{\alpha_1 \cdot (v+1)}{v + (\boldsymbol{\xi} - \boldsymbol{\omega}_j)^T(\boldsymbol{\xi} - \boldsymbol{\omega}_j)} \begin{cases} (P_y(j|\boldsymbol{\xi}) - P(j|\boldsymbol{\xi}))(\boldsymbol{\xi} - \boldsymbol{\omega}_j), & c(\boldsymbol{\omega}_j) = y, \\ -P(j|\boldsymbol{\xi})(\boldsymbol{\xi} - \boldsymbol{\omega}_j), & c(\boldsymbol{\omega}_j) \neq y \end{cases}$$

We would like to remark that this formula can be derived in another way. If we were to drop the requirement for the conditional densities to have the normalized exponential form and simply set $p(\boldsymbol{\xi}|j) = q_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}_j, v)$, then after working out all the modified update formulas as is done in [Schn09], we end up with exactly the same update rule. To see why this is true, realize that $\frac{\partial q_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}, v)}{\partial \boldsymbol{\omega}} = \frac{\partial f_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}, v)}{\partial \boldsymbol{\omega}} \cdot q_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}, v)$, and use this in deriving the update formulas.

It should be noted that the values of $P_y(j|\boldsymbol{\xi})$ and $P(j|\boldsymbol{\xi})$ have different values compared to when using Gaussian distributions, since f is used in their definitions [Schn09]:

$$P_y(j|\boldsymbol{\xi}) = \frac{P(j)K(j) \exp f_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}_j, v)}{\sum_{i:c(\boldsymbol{\omega}_i)=y} P(i)K(i) \exp f_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}_i, v)} \tag{4}$$

$$P(j|\boldsymbol{\xi}) = \frac{P(j)K(j) \exp f_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}_j, v)}{\sum_i P(i)K(i) \exp f_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}_i, v)} \tag{5}$$

The $P(i \text{ or } j)$ here is the prior probability of a prototype. Note that if we assume that all prototypes have the same prior probability, the terms cancel each other out in both equations. Furthermore, the $K(i \text{ or } j)$ is a normalization factor, which in our case is always 1 since we do not need normalization as we chose f_{stud} to be the log of a pdf (which is normalized by definition). As a third remark, please note

that we defined $f_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}, v) = \log q_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}, v)$. So using all this, we get:

$$P_y(j|\boldsymbol{\xi}) = \frac{q_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}_j, v)}{\sum_{i:c(\boldsymbol{\omega}_i)=y} q_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}_i, v)}$$

$$P(j|\boldsymbol{\xi}) = \frac{q_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}_j, v)}{\sum_{i} q_{\text{stud}}(\boldsymbol{\xi}, \boldsymbol{\omega}_i, v)}$$

where $q_{\text{stud}}(\dots)$ is the Student's t pdf as defined in equation (2).

2.3. **Update Rules.**

We first give the update rule for the Gaussian distribution [Schn09]:

$$\Delta\boldsymbol{\omega}_j = \frac{\alpha_1}{\sigma^2} \begin{cases} (P_y(j|\boldsymbol{\xi}) - P(j|\boldsymbol{\xi}))(\boldsymbol{\xi} - \boldsymbol{\omega}_j), & c(\boldsymbol{\omega}_j) = y, \\ -P(j|\boldsymbol{\xi})(\boldsymbol{\xi} - \boldsymbol{\omega}_j), & c(\boldsymbol{\omega}_j) \neq y \end{cases} \tag{6}$$

Our previously derived update rule for Student's t distributions is as follows:

$$\Delta\boldsymbol{\omega}_j = \frac{\alpha_2 \cdot (v+1)}{v + (\boldsymbol{\xi} - \boldsymbol{\omega}_j)^T(\boldsymbol{\xi} - \boldsymbol{\omega}_j)} \begin{cases} (P_y(j|\boldsymbol{\xi}) - P(j|\boldsymbol{\xi}))(\boldsymbol{\xi} - \boldsymbol{\omega}_j), & c(\boldsymbol{\omega}_j) = y, \\ -P(j|\boldsymbol{\xi})(\boldsymbol{\xi} - \boldsymbol{\omega}_j), & c(\boldsymbol{\omega}_j) \neq y \end{cases} \tag{7}$$

where v denotes the degrees of freedom used in the update rule.

While the exact values of the $P_y(j|\boldsymbol{\xi})$ and $P(j|\boldsymbol{\xi})$ terms in the above two formulas differ depending on whether the Student's t or the Gaussian pdf is used, they follow the same form (derived in the previous section):

$$P_y(j|\boldsymbol{\xi}) = \frac{\text{pdf}(\boldsymbol{\xi}, \boldsymbol{\omega}_j, \dots)}{\sum_{i:c(\boldsymbol{\omega}_i)=y} \text{pdf}(\boldsymbol{\xi}, \boldsymbol{\omega}_i, \dots)} \tag{8}$$

$$P(j|\boldsymbol{\xi}) = \frac{\text{pdf}(\boldsymbol{\xi}, \boldsymbol{\omega}_j, \dots)}{\sum_{i} \text{pdf}(\boldsymbol{\xi}, \boldsymbol{\omega}_i, \dots)} \tag{9}$$

where $\text{pdf}(\dots)$ is the probability density function (so either Student's or Gaussian). The pdf's distance measure is the Euclidean distance, that is the root mean square of $(\boldsymbol{\xi} - \boldsymbol{\omega})$. We note here that for every test in this paper we assume one prototype per class, so the P_y parts (equation (8)) are always 1.

2.4. **Dataset Assumptions.**

For our initial analysis we assume a simple dataset. The dataset is two-dimensional and contains two classes. Exactly half of the samples in the dataset is labeled as class 0, and the other half as class 1. The classes in the dataset are distributed as two-dimensional Gaussians, with class variance σ_d^2 in each dimension. So note that while σ^2 denotes the variance used in the update rule, σ_d^2 denotes the variance used in the dataset. The class means are located at $[\pm D, 0]$. Thus, D is the absolute distance from the class means to the origin.

For our initial analysis we use $\sigma_d^2 = 3.0$ and $D = \sqrt{3}$, $2\sqrt{3}$, $3\sqrt{3}$ or $4\sqrt{3}$. We let the RSLVQ methods use one prototype per class.

The datasets we used for calculations contained 1024 samples per class, although for display purposes some plots show only 384 samples per class.

If we refer to the two classes as *class 0* and *class 1*, then class 0 is the class with class mean $[-D, 0]$ and class 1 is the class with class mean $[D, 0]$.

3. Analysis

In this section we will first break the update rules down and visualize the normalized probability density and update strengths per position. We then formulate the problem of determining final prototype positions (i.e. where the prototypes of the RSLVQ methods end up), and approach this problem both analytically and numerically.

In the following, we may refer to the first dimension of the prototypes as the x-dimension, and to the second as the y-dimension. Furthermore, "the prototype" will always refer to the left prototype, and "the other prototype" refers to the right prototype. Black points are datapoints belonging to the class of the left prototype, and white points are datapoints belonging to the class of the right prototype. Furthermore, note that the figures show the unweighted forces, i.e. they do not depend on the dataset, only on the position of the prototypes. These are shown in magenta, and chosen to coincide with the class means of the dataset portrayed.

3.1. **Visualizing The Update Rules.**

In RSLVQ, samples are classified by nearest prototype. So the difference between the RSLVQ methods comes down to how and where the prototypes move during training. In this section we visualize the forces that act on the prototypes.

3.1.1. *Normalized Probability Density.*

First, we want to measure the effect that the different distributions have on the

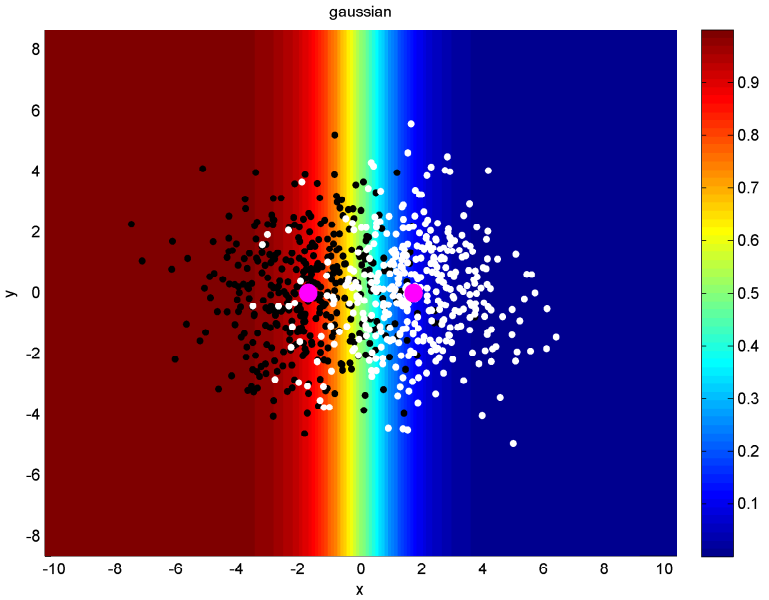

Figure 1. The force $P(j|\boldsymbol{\xi})$ (see equation (9)) of the Gaussian update rule, for the prototype on the left side. Meta-parameter σ^2 (the variance) is set to 3.0. The dataset shown is an instance of the dataset described in section 2.4 with $D = \sqrt{3}$, and serves to illustrate the scale.

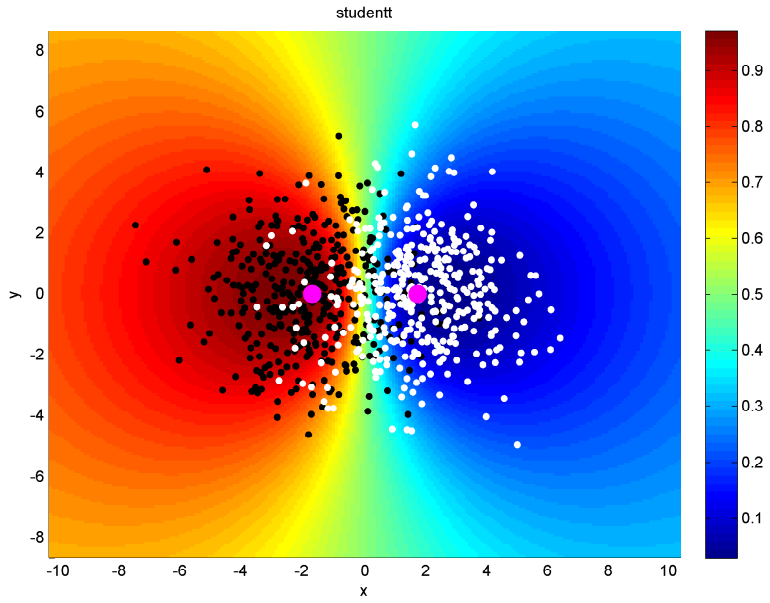

FIGURE 2. The force $P(j|\boldsymbol{\xi})$ (see equation (9)) of the Student's t update rule, for the prototype on the left side. Meta-parameter v (degrees of freedom) is set to 3. The dataset shown is an instance of the dataset described in section 2.4 with $D = \sqrt{3}$, and serves to illustrate the scale.

algorithm by measuring the effect they have on the update rule. Looking at equations (6) and (7), we recall that the P_y parts (equation (8)) are always 1. This means a large part of the differences between the methods stems from the term $P(j|\boldsymbol{\xi})$ (equation (9)), the relative ratio of the probabilities at a certain position (or: the *normalized probability density*).

We have visualized this force in Fig. 1 for the Gaussian method and in Fig. 2 for the Student's t method. We note that these figures do not depend on the dataset, only on the position of the prototypes. These are shown in magenta, and chosen to coincide with the class means of the dataset portrayed.

To start with Fig. 1, what this figure shows is that for the left prototype, black points closer to the right prototype and white points closer to the left prototype have the largest impact on its position. Points lying in the overlap in the middle affect it as well. Generalizing for both prototypes, we can conclude that points closer to the 'wrong' distribution have the largest impact on the prototypes, but the effect of the other points should not be overlooked. Of course more points come into play if we increase the variance, as this increases the width of the center band where partial forces act on datapoints.

Now we look at Fig. 2, which shows the same dataset, but displays the force ($P(j|\boldsymbol{\xi})$) according to the Student's t distribution. There are some striking differences. First off, the overlap area near the middle is wider, especially towards the top and bottom (it narrows when increasing v). Secondly, we notice that because of the heavy-tailed property of the Student's t distribution, nowhere in the figure are the forces exactly 1 or 0 (minimum is 0.0286 and maximum is 0.9714). This means that all points

in the dataset influence the position of the prototypes. For this particular dataset it means that for example outliers from the left prototype can not only move this prototype to the right side, but also to the left side (because the $P(j|\boldsymbol{\xi})$ force is not exactly 1 there). It should be noted that if the distance between the prototypes is increased, more extreme values are reached (i.e. closer to zero or one), and the figure will look a bit more like the Gaussian one. The same happens when we increase v.

We also see that towards the left and right edge of the figure, the force does not remain constant but starts to decrease/increase. Initial tests show that these forces reach 0.5 at both sides in infinity, as long as v is not infinitely large. This is because the force is in fact a ratio of two probability distributions at the prototypes. With increasing distance from a point to the prototypes, the distance between the two prototypes becomes relatively smaller. This does not happen for the Gaussian distribution because it goes to zero too fast.

The fact that the Gaussian $P(j|\boldsymbol{\xi})$ force is invariant to changes in the y-coordinate means that the ratio of the pdfs at different distances from a point is the same. This does not hold for the Student's t pdf, where for the same x-coordinate, the force decreases with increasing y-distance to the mean. This means that compared to the Gaussian, the decrease of the Student's t pdf is slower at larger distances/faster at smaller distances. This is in accordance with what we know from the one-dimensional shape of the pdfs.

3.1.2. *Update Strengths As Function Of Sample Position.*
Of course, the term $P(j|\boldsymbol{\xi})$ is an important factor in understanding how the update rules work, but the update rules include other factors that differ per distribution as well. So to compare the effects of the Gaussian and Student's t distributions on the update rule, we need to visualize the update rules as a whole, i.e. the actual increments/decrements to the prototypes from a datapoint at a certain position. We have calculated these for both the Gaussian (Fig. 3) and the Student's t distribution (Fig. 4).

Now we will analyze Fig. 3. The left upper image tells us that only black points with $x \geq 0$ influence the prototype. In fact, those points pull the prototype to the right (the plot corresponds to the x-dimension and shows positive values). The more further to the right a black point is found, the stronger it will pull the prototype to the right. We can see that for this dataset, quite a few black points actually influence the x-dimension of the prototype. The upper right image shows how black points update the y-value of the prototype. Note that there are positive and negative updates, e.g. a black point with $y > 0, x > 0$ pulls the prototype up, while a black point with $y < 0, x > 0$ pulls it down. The lower two images correspond to how the white points update the prototype. In the left lower image we can see that a white point with $x_{\text{prototype}} < x < 0$ pushes the prototype to the left, while a white point with $x < x_{\text{prototype}}$ pushes the prototype to the right.

Interestingly, the strength of the updates increases with the distance to the prototype; this is due to the $(\boldsymbol{\xi} - \boldsymbol{\omega}_j)$ term in the update rule. The result is that the direction of the update is the mean direction of all points taken into account. While this is often desired it is also questionable, since most points considered will already be outliers, and the largest outliers will have the largest influence. The lower right image shows that for white points with $x < 0$, points above the prototype push it down, and points below the prototype push it up.

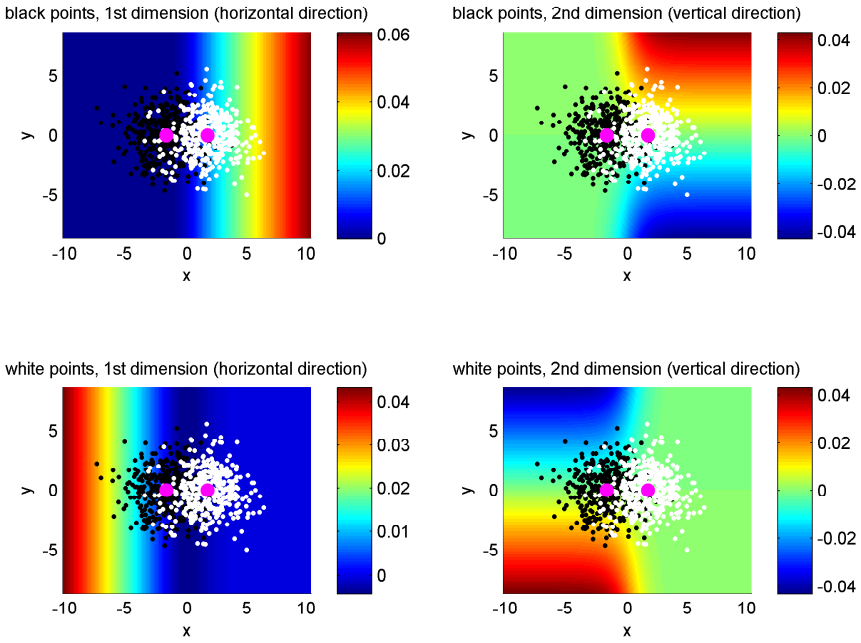

FIGURE 3. The images show only forces that update the left prototype (the one amidst the black points). Shown are the actual $\Delta\boldsymbol{\omega}$ values – positive values move the prototype up or to the right, negative values move it down or to the left. Forces were calculated using the Gaussian update rule, with meta-parameters $\alpha = 0.015$ and $\sigma^2 = 3.0$. The upper two images show the attractive forces, i.e. those that apply to datapoints from the same class (e.g. the black points). The bottom two images show the repulsive forces, i.e. those that apply to datapoints from the other class (e.g. the white points). Since the update of a datapoint to the prototype is actually a two-dimensional vector, we have two images per force: the images on the left correspond to the first dimension (horizontal distance), and the images on the right correspond to the second dimension (vertical distance).

We notice that in the left two images, the y-coordinate does not seem to affect the value at all. This is true even though the pdfs use the Euclidean distance over both dimensions. We note that the Gaussian update rule for the x-dimension has no dependencies on y, except for the ratio of pdfs; and as it turns out, this ratio does not change with varying y. We have included the proof for this claim in appendix A.

Most of these effects are within our expectations. Noteworthy is that in this Gaussian approach, outliers are equally important if not more important than other points. This is understandable for outliers that are close to class boundaries, but it also holds for outliers that are completely on the other side (i.e. white points lying in the far left of the lower left image). Also note that black points cannot move the prototype to the left, as the left upper image has no negative values.

We move on to analyze Fig. 4. In the left upper image we can see that black points that are near the other prototype have the largest influence on moving the

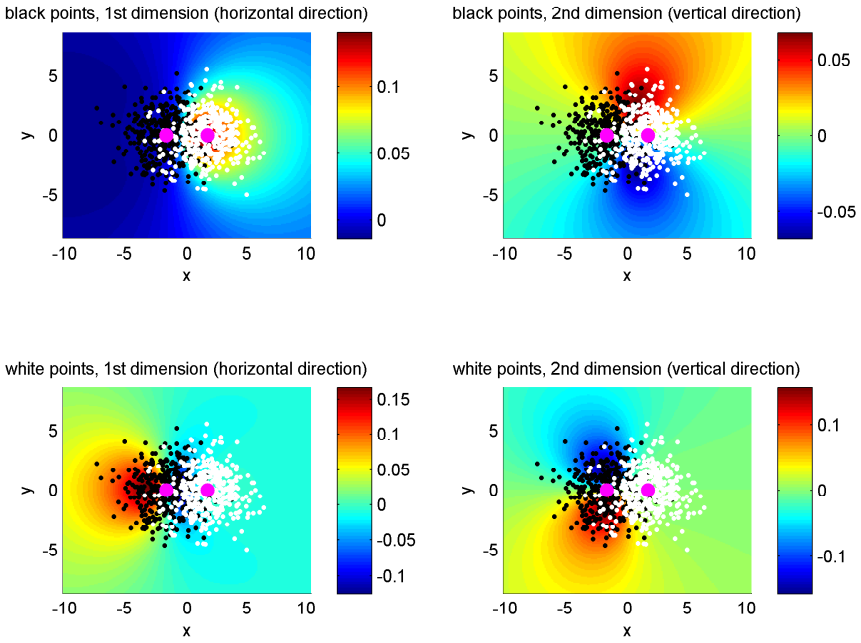

FIGURE 4. The images show only forces that update the left prototype (the one amidst the black points). Shown are the actual $\Delta\boldsymbol{\omega}$ values – positive values move the prototype up or to the right, negative values move it down or to the left. Forces were calculated using the Student's t update rule, with meta-parameter $\alpha = 0.15$ and $v = 3$. The upper two images show the attractive forces, i.e. those that apply to datapoints from the same class (e.g. the black points). The bottom two images show the repulsive forces, i.e. those that apply to datapoints from the other class (e.g. the white points). Since the update of a datapoint to the prototype is actually a two-dimensional vector, we have two images per force: the images on the left correspond to the first dimension (horizontal distance), and the images on the right correspond to the second dimension (vertical distance).

prototype to the right. Contrary to the Gaussian update rule, outliers lying further to the right/above/below have a lesser impact on the prototype's movement. Also note that there are small negative values to the left of the prototype, i.e. black points to the left of the prototype can pull it to the left. The upper right image shows that black points above the prototype pull it up, and black points below the prototype pull it down. Because the $P(\dots)$ force for Student's t has no extreme values, transitions around $x = 0$ are less strict. The left lower image shows that white points can push the prototype to the left or right, as long as they are relatively close to the prototype. The lower right image shows that white points can push the prototype up or down, as long as they are relatively close to the prototype.

The most striking difference between the Gaussian and Student's t update rules would be that for Student's t, both dimensions play an important role in all figures. For instance, values in the left figures differ for different y-values, where they do not for the Gaussian update rule. This is because the update rule of the Student's t

distribution includes the term $\frac{1}{(\boldsymbol{\xi}-\boldsymbol{\omega}_j)^T(\boldsymbol{\xi}-\boldsymbol{\omega}_j)}$. Ergo, forces decrease (in the absolute sense) with increasing Euclidean distance from the prototype.

The movement of Gaussian prototypes is predictable, e.g. if we have two overlapping clusters so that most points influence the prototypes, we know that the prototype will move towards the mean of those points, and thus approach the means of the classes. The Student's t update rule comes with no such guarantee, so it requires more refined calculations as to determine the final prototype positions. On the other hand, outliers in datasets have a lesser impact on the prototypes for the Student's t update rule, so it is more robust to e.g. possible noise or different instances of the same dataset. So where the Student's t method relies on points relatively close to the prototype, the Gaussian method puts most weight on outliers.

We note that all these differences start to fade when we increase v. As v increases, the Student's t distribution becomes more like the Gaussian distribution. For instance at $v = 100$ or higher, we hardly see any differences between the methods.

3.1.3. *Separation Between The Classes.*
We have seen in the previous section that there are certain differences between the Gaussian and Student's t update rules. When we increase the separation between the classes (D), these distinctions are amplified.

We refer to Figs. 5 and 6. These figures are like Figs. 3 and 4 from the previous section, except that the forces of the update rule are now weighted by the probability density distribution of their class. This allows us to see where the updates to the prototypes originate from. The images were calculated using $D = 4\sqrt{3}$, assuming prototypes at the class means.

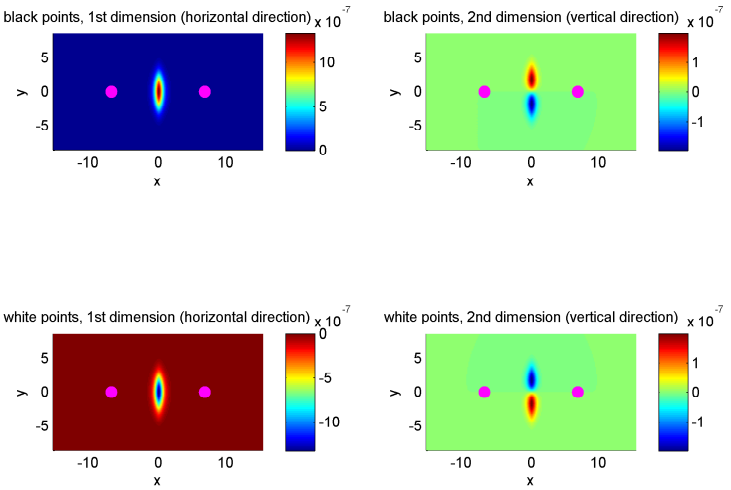

FIGURE 5. The forces of the Gaussian update rule, weighted by the probability density distributions of the dataset, assuming the prototypes (shown in magenta) are at the class means. Meta-parameters are $\sigma^2 = 3.0$, $\sigma_d^2 = 3.0$, $D = 4\sqrt{3}$.

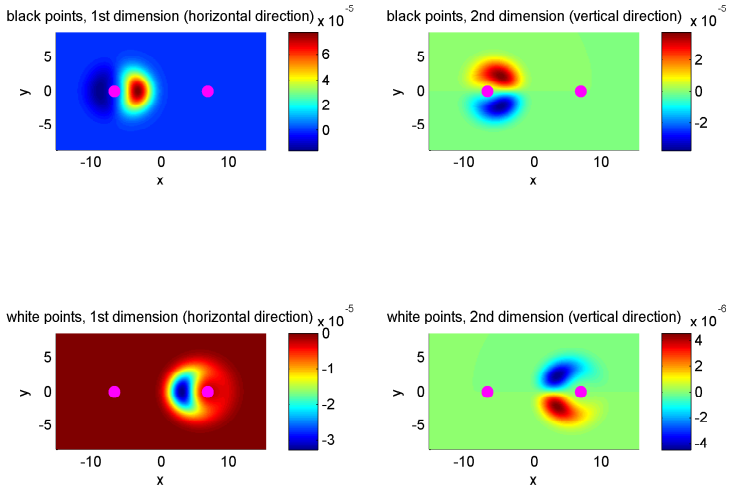

FIGURE 6. The forces of the Student's t update rule, weighted by the probability density distributions of the dataset, assuming the prototypes (shown in magenta) are at the class means. Meta-parameters are $v = 3$, $\sigma_d^2 = 3.0$, $D = 4\sqrt{3}$.

We can see in Fig. 5 that updates to the prototypes for the Gaussian method at $D = 4\sqrt{3}$ depend only on a very narrow strip of points near the middle, far away from the class means. On a real dataset, there are but a few outliers in these areas, if any. Thus, with greater separation between the classes, updates to the prototypes for the Gaussian method become increasingly dependent on outliers. We will see later that this causes convergence problems for the Gaussian method at higher D, as well as low accuracy (i.e. different instances of the same dataset having different outliers, resulting in greatly varying prototypes and decision boundaries).

For the Student's t method on the other hand (Fig. 6), we can see that instead, movement of the prototypes is directed by a large area of points closely surrounding the class means. This behavior can be traced back to the heavy-tailed property of the Student's t distribution. To see why, recall from Fig. 1 that some of the black points with $x < 0$ had $P(j|\boldsymbol{\xi}) = 1$ and were thus excluded, while the Student's t method did not exclude any points. We will later see that because at higher D the Student's t method uses a larger area, with more points and less outliers, it is more robust (smaller variance in testset accuracies) and its worst-case performance is better than that of the Gaussian method.

We note here that for this dataset we would need to use a very high variance (i.e. $\sigma^2 = 30.0$) for the Gaussian method to have the same weight on the tails as the Student's t method has using $v = 3$. However as we will later see, at this variance the prototypes of the Gaussian method convergence at about ten times further away from the origin than those of the Student's t method (which converge at the class means). This results in lower accuracy as higher learning rates are needed for runs to converge and because noise in the dataset has a larger influence when forces are weak.

3.2. **Finding Final Prototypes.**

We know from [Seo03] that in theory, RSLVQ prototypes always converge. To compare the behavior of the two RSLVQ methods, we want to know *where they converge to*, i.e. where the prototypes will end up after an infinite amount of iterations.

Let *final prototypes* denote the positions where the prototypes have converged to after an infinite amount of iterations.

To see where the prototypes end up, we can run the RSLVQ algorithm a number of times, but this is slow and inaccurate. So we will try to solve the problem of finding final prototypes without having to run the RSLVQ algorithm.

We start by making some assumptions. Since the class means of our dataset have the same y-coordinate, we are only interested in finding the x-coordinate of the final prototypes. We will simply assume that $y = 0$ for the final prototypes. Note however that we still have to include the y-dimension in our calculations, as it matters for the Student's t method.

We also know that given the symmetry of our dataset, both RSLVQ methods will end up with decision boundaries at $x = 0$. This allows us to restrict our attention to the left prototype only, like we did for the figures in section 3.1, assuming that the other prototype behaves in a mirrored way.

Now our theorem is the following: the final prototype is at that exact location where the sum of all the forces pulling it to the left equals the sum of all the forces pulling it to the right.

The motivation for this theorem is the realization that once the RSLVQ algorithm has reached prototype positions that fulfill this requirement, the prototypes will not change in future iterations, as there is no net force exerted on them. In other words, if the RSLVQ algorithm can reach prototype positions where this holds, then these are final prototypes, as the algorithm will converge at that position.

Since the classes in our dataset follow a Gaussian distribution, our theorem is equivalent to saying (referring to Figs. 3 and 4) that we multiply the forces of the update rule by their classwise data distributions.

More formally, we would say (for the left prototype) that we look for $\Delta_{\text{forces}} = 0$, where Δ_{forces} is defined as:

$$\begin{aligned}\Delta_{\text{forces}} = \int_m \int_n \text{gpdf}(\sqrt{(m+D)^2+n^2}, 0, \sigma_d) \cdot \Delta\boldsymbol{\omega}_{x,\, c(\boldsymbol{\omega})=y}([m\ n], [-\boldsymbol{\omega}_x\ 0]) + \\ \text{gpdf}(\sqrt{(m-D)^2+n^2}, 0, \sigma_d) \cdot \Delta\boldsymbol{\omega}_{x,\, c(\boldsymbol{\omega})\neq y}([m\ n], [-\boldsymbol{\omega}_x\ 0])\ dn\ dm\end{aligned} \tag{10}$$

where gpdf(...) is the Gaussian pdf that takes as parameters the (Euclidean) distance, the mean and the standard deviation; D is the absolute distance from the origin to the class means in the dataset; σ_d is the class standard deviation of the dataset (equal for both classes); $\Delta\boldsymbol{\omega}_{x,\, c(\boldsymbol{\omega})=y}(\dots)$ is the update in the x-dimension using the upper option in the update rule, and takes as parameters the datapoint ($\boldsymbol{\xi}$) and the prototype ($\boldsymbol{\omega}$). The prototype is given as $[-\boldsymbol{\omega}_x\ 0]$, where $\boldsymbol{\omega}_x$ is the absolute distance from the prototypes to the origin (i.e. if the other prototype is needed in the formula, $[\boldsymbol{\omega}_x\ 0]$ is used); $\Delta\boldsymbol{\omega}_{x,\, c(\boldsymbol{\omega})\neq y}(\dots)$ is the update in the x-dimension using the lower option in the update rule, and takes as parameters the datapoint ($\boldsymbol{\xi}$) and the prototype ($\boldsymbol{\omega}$). Note that since we only consider one prototype here, we left out the additional i or j subscript to $\boldsymbol{\omega}$ that was used in section 2.3. Variables m and n range over the x- and y-dimension, respectively.

Now to solve this formula we would be looking for the value $\boldsymbol{\omega}_x$ for which $\Delta_{\text{forces}} = 0$. Of course the solutions differ depending on whether we use the Gaussian or the Student's t method.

3.3. **Analytical Approach.**

In this section we try to find analytical solutions to the problem of finding $\boldsymbol{\omega}_x$ for which $\Delta_{\text{forces}} = 0$.

We start with the formula for Δ_{forces} from equation (10). This formula is of the form:

$$\Delta_{\text{forces}} = \int_m \int_n f(m,n)\, dn\, dm \tag{11}$$

where $f(m,n)$ is the middle part of the integrals, which we write as a function of m and n. Besides the variables m and n, the function $f(m,n)$ also depends on certain meta-parameters (i.e. σ, σ_d, $\boldsymbol{\omega}_x$, D, etc.), which can be treated as constants with respect to the integrations.

3.3.1. *Trivial Solution.*

In this section we give the analytical derivation of the solutions for the Gaussian method.

We are interested in finding solutions where $\Delta_{\text{forces}} = 0$ (see equation (11)). Now assume that there is a specific configuration of the aforementioned meta-parameters, such that $f(m,n) = 0$ for all m, n. Then, $\int_m \int_n f(m,n)\, dn\, dm = 0$ as well. We call this solution the *trivial solution*, and we will later see (section 3.4) that for the Gaussian method all possible final prototypes can be found this way.

This solution can then be found by setting $f(m,n) = 0$, treating m and n as any other variable in the formula and solving/reducing in such a way that it no longer depends on m or n.

Summarizing, our theorems are the following:

- Assuming that we have found a specific configuration of meta-parameters such that $f(m,n)$ is always zero, this is a valid solution for the $\Delta_{\text{forces}} = 0$ problem (although there may be other solutions).
- If we set $f(m,n) = 0$ and by using only simple solve and reduce steps we eventually end up with an equation that no longer depends on n or m (i.e. they no longer appear in the formula), we have proven the assumption that there is a specific configuration of meta-parameters such that $f(m,n)$ is always zero.

  Namely, we have an equation (specific configuration of meta-parameters) such that $f(m,n)$ is always 0 – since we started with this assumption and showed that $f(m,n)$ does not depend on m or n, as they could apparently be canceled out.
- If we cannot solve/reduce $f(m,n) = 0$ in such a way that m and n no longer occur in the equation then this either means that such a solution does not exist, or that our math was not good enough to find it – either way, we are unable to prove that such a solution ($f(m,n)$ always zero) exists, so we cannot derive a solution analytically (at least not in this way). We will later see that this is the case for the Student's t method.

So for $\int_m \int_n f(m,n)\ dn\ dm = 0$, the trivial solution is $f(m,n) = 0$. We will now derive the trivial solution for the Gaussian method, proving that it exists. Below, we start by setting $f(m,n) = 0$ and using simple solve/reduce steps until we end up with an equation that no longer depends on m or n.

We start by setting $f(m,n) = 0$, written out in full, this gives:

$$\begin{aligned}\text{gpdf}(\sqrt{(m+D)^2+n^2}, 0, \sigma_d) \cdot \Delta\boldsymbol{\omega}_{x,\ c(\boldsymbol{\omega})=y}([m\ n],[-\boldsymbol{\omega}_x\ 0]) + \\ \text{gpdf}(\sqrt{(m-D)^2+n^2}, 0, \sigma_d) \cdot \Delta\boldsymbol{\omega}_{x,\ c(\boldsymbol{\omega})\neq y}([m\ n],[-\boldsymbol{\omega}_x\ 0]) = 0\end{aligned}$$

Rewriting gives:

$$\begin{aligned}\text{gpdf}(\sqrt{(m+D)^2+n^2}, 0, \sigma_d) \cdot \Delta\boldsymbol{\omega}_{x,\ c(\boldsymbol{\omega})=y}([m\ n],[-\boldsymbol{\omega}_x\ 0]) \ = \\ -\text{gpdf}(\sqrt{(m-D)^2+n^2}, 0, \sigma_d) \cdot \Delta\boldsymbol{\omega}_{x,\ c(\boldsymbol{\omega})\neq y}([m\ n],[-\boldsymbol{\omega}_x\ 0])\end{aligned}$$

Both options of the update rule share the factors $\frac{\alpha_1}{\sigma^2}$ and $(\boldsymbol{\xi} - \boldsymbol{\omega}_j)$, so we can leave those out. We also know that $P_y(j|\boldsymbol{\xi}) = 1$. This gives us:

$$\begin{aligned}\text{gpdf}(\sqrt{(m+D)^2+n^2}, 0, \sigma_d) \cdot (1 - P(j|\boldsymbol{\xi})) \ = \\ \text{gpdf}(\sqrt{(m-D)^2+n^2}, 0, \sigma_d) \cdot (P(j|\boldsymbol{\xi}))\end{aligned}$$

Writing out the expression for $P(j|\boldsymbol{\xi})$ gives:

$$\begin{aligned}\text{gpdf}(\sqrt{(m+D)^2+n^2}, 0, \sigma_d) \cdot \left(1 - \frac{\text{gpdf}(\sqrt{(m+\boldsymbol{\omega}_x)^2+n^2}, 0, \sigma)}{\text{gpdf}(\sqrt{(m+\boldsymbol{\omega}_x)^2+n^2}, 0, \sigma) + \text{gpdf}(\sqrt{(m-\boldsymbol{\omega}_x)^2+n^2}, 0, \sigma)}\right) = \\ \text{gpdf}(\sqrt{(m-D)^2+n^2}, 0, \sigma_d) \cdot \frac{\text{gpdf}(\sqrt{(m+\boldsymbol{\omega}_x)^2+n^2}, 0, \sigma)}{\text{gpdf}(\sqrt{(m+\boldsymbol{\omega}_x)^2+n^2}, 0, \sigma) + \text{gpdf}(\sqrt{(m-\boldsymbol{\omega}_x)^2+n^2}, 0, \sigma)}\end{aligned}$$

Now $P(j|\boldsymbol{\xi})$ is of the form $\frac{A}{A+B}$. So for $(1 - P(j|\boldsymbol{\xi}))$ we can write $\left(1 - \frac{A}{A+B}\right) = \frac{A+B-A}{A+B} = \frac{B}{A+B}$. This gives us:

$$\begin{aligned}\text{gpdf}(\sqrt{(m+D)^2+n^2}, 0, \sigma_d) \cdot \frac{\text{gpdf}(\sqrt{(m-\boldsymbol{\omega}_x)^2+n^2}, 0, \sigma)}{\text{gpdf}(\sqrt{(m+\boldsymbol{\omega}_x)^2+n^2}, 0, \sigma) + \text{gpdf}(\sqrt{(m-\boldsymbol{\omega}_x)^2+n^2}, 0, \sigma)} \ = \\ \text{gpdf}(\sqrt{(m-D)^2+n^2}, 0, \sigma_d) \cdot \frac{\text{gpdf}(\sqrt{(m+\boldsymbol{\omega}_x)^2+n^2}, 0, \sigma)}{\text{gpdf}(\sqrt{(m+\boldsymbol{\omega}_x)^2+n^2}, 0, \sigma) + \text{gpdf}(\sqrt{(m-\boldsymbol{\omega}_x)^2+n^2}, 0, \sigma)}\end{aligned}$$

Note we have equal denominators on both sides, which can be filtered out, giving:

$$\begin{aligned}\text{gpdf}(\sqrt{(m+D)^2+n^2}, 0, \sigma_d) \cdot \text{gpdf}(\sqrt{(m-\boldsymbol{\omega}_x)^2+n^2}, 0, \sigma) \ = \\ \text{gpdf}(\sqrt{(m-D)^2+n^2}, 0, \sigma_d) \cdot \text{gpdf}(\sqrt{(m+\boldsymbol{\omega}_x)^2+n^2}, 0, \sigma)\end{aligned}$$

Writing out the full formula for the gpdfs gives us:

$$\begin{aligned}\frac{1}{\sigma_d\sqrt{2\pi}} \exp\left(-\frac{(m+D)^2+n^2}{2\sigma_d^2}\right) \cdot \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(m-\boldsymbol{\omega}_x)^2+n^2}{2\sigma^2}\right) = \\ \frac{1}{\sigma_d\sqrt{2\pi}} \exp\left(-\frac{(m-D)^2+n^2}{2\sigma_d^2}\right) \cdot \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(m+\boldsymbol{\omega}_x)^2+n^2}{2\sigma^2}\right)\end{aligned}$$

The common terms on both sides cancel each other out, leaving:

$$\begin{aligned}\exp\left(-\frac{(m+D)^2+n^2}{2\sigma_d^2}\right) \cdot \exp\left(-\frac{(m-\boldsymbol{\omega}_x)^2+n^2}{2\sigma^2}\right) = \\ \exp\left(-\frac{(m-D)^2+n^2}{2\sigma_d^2}\right) \cdot \exp\left(-\frac{(m+\boldsymbol{\omega}_x)^2+n^2}{2\sigma^2}\right)\end{aligned}$$

We can take the two powers of e together on both sides:

$$\exp\left(-\frac{(m+D)^2+n^2}{2\sigma_d^2}-\frac{(m-\boldsymbol{\omega}_x)^2+n^2}{2\sigma^2}\right)= \exp\left(-\frac{(m-D)^2+n^2}{2\sigma_d^2}-\frac{(m+\boldsymbol{\omega}_x)^2+n^2}{2\sigma^2}\right)$$

And take the logarithm:

$$-\frac{(m+D)^2+n^2}{2\sigma_d^2}-\frac{(m-\boldsymbol{\omega}_x)^2+n^2}{2\sigma^2}= -\frac{(m-D)^2+n^2}{2\sigma_d^2}-\frac{(m+\boldsymbol{\omega}_x)^2+n^2}{2\sigma^2}$$

Notice how all the n^2 terms cancel each other out. Flipping signs and multiplying by 2 then gives us:

$$\frac{(m+D)^2}{\sigma_d^2}+\frac{(m-\boldsymbol{\omega}_x)^2}{\sigma^2} = \frac{(m-D)^2}{\sigma_d^2}+\frac{(m+\boldsymbol{\omega}_x)^2}{\sigma^2}$$

Writing out the numerators on both sides we get:

$$\frac{(m+D)^2}{\sigma_d^2}+\frac{m^2-2m\boldsymbol{\omega}_x+\boldsymbol{\omega}_x^2}{\sigma^2} = \frac{(m-D)^2}{\sigma_d^2}+\frac{m^2+2m\boldsymbol{\omega}_x+\boldsymbol{\omega}_x^2}{\sigma^2}$$

Subtracting the equal terms on both sides and rearranging gives:

$$\frac{(m+D)^2-(m-D)^2}{\sigma_d^2} = \frac{4m\boldsymbol{\omega}_x}{\sigma^2}$$

Rearranging gives:

$$\frac{\sigma^2((m+D)^2-(m-D)^2)}{4m\sigma_d^2} = \boldsymbol{\omega}_x$$

Writing out the expressions in parentheses gives:

$$\frac{\sigma^2(4mD)}{4m\sigma_d^2} = \boldsymbol{\omega}_x$$

Note that the term $4m$ is canceled out, and we get:

$$\frac{\sigma^2 D}{\sigma_d^2} = \boldsymbol{\omega}_x \tag{12}$$

This last equation is of the form we were trying to derive. Note how neither m nor n occur in it.

We note that we cannot derive such a relation for the Student's t method. Because the distributions of dataset and update rule are different, we cannot filter out enough common factors to get to the point where individual m and n values can be canceled out. As such, we cannot prove that it has a trivial solution. Note that the Student's t method still has nontrivial solutions, but we can only find these numerically, making it harder to determine in what way the (meta-)parameters influence the results.

Summarizing for the Gaussian method, we know that $\boldsymbol{\omega}_x$ scales with the variance and the distance between the means, and that it scales inversely with the variance of the classes in the dataset. This relation holds with respect to the dataset assumptions from section 2.4. It might not hold for more complex datasets of more classes and/or prototypes. Note that when $\sigma^2 = \sigma_d^2$, the final prototypes coincide with the class means of the dataset, for any D.

3.3.2. *Dataset Variance And The Gaussian Method.*
We have seen in our derived formula (12), that for the Gaussian method prototypes overlap exactly with the class means if the variance of the dataset and the variance of the update rule are chosen equal. In this section we explain how the variance of the dataset should be interpreted, and why the two variances are related.

The confusion stems from the fact that in the update rule of our integrations, we seem to be taking the Euclidean distance of two dimensions. The Euclidean distance of two normally distributed dimensions is no longer normally distributed; not even half-normally distributed. This poses the question: if the probability density of the update rule does not match its input, why should its variance match that of the dataset?

The short answer is that the Euclidean distance measure used in the update rule of our integrations is effectively taken over just one dimension, because the other dimension can be canceled out. And the Euclidean distance of a single, normally distributed dimension is half-normally distributed, thus it makes sense to choose a variance that matches the distribution of that dimension; which would be the variance of one of the dimensions in the dataset. Thus, the dataset should be distributed in such a way that each dimension follows an independent Gaussian distribution with variance equal to the variance used in the update rule.

The question that remains is why one dimension can be canceled out, which we will explain now. Recall the simple case of section 2.4: a dataset with two classes, distributed as two-dimensional Gaussians. Since the means of both classes always lie on the line with $y = 0$, our integrations assume the final prototype has $y = 0$ as well; thus reducing the problem of finding the final prototype within a certain region to a one-dimensional search. Furthermore, updates to the prototypes are vectors, they have an x- and a y-component. Our program only considers the x-component of these updates, since it already knows what the final y will be.

Key is now to realize that the updates of the Gaussian method in the x-dimension do not depend on y at all. This is visible in the images on the left of Fig. 3, and we have included the proof for this claim in appendix A. This appendix further states that the ratio of pdfs effectively uses the one-dimensional Euclidean distance in x. The Euclidean distance over one normally distributed dimension is still half-normally distributed, and can be approximated by a pdf with zero mean and the same variance as the distribution in its dimension. In other words, this tells us that with respect to our integrations, the optimal choice of variance to use in the update rule is the variance of the x-dimension in the dataset. Optimal in the sense that it forms the best match to the distribution that the pdf of the update rule expects.

Since we already knew from our analytical derivation that the variance of the dataset and the update rule are related, we conclude that the main result from this section was to find out why they are related, and how the variance of the dataset should be interpreted.

As a final remark we note that in practical implementations, datasets are often normalized per dimension (not per class). Since this normalization is done per dimension, the variance of each class is likely to differ per dimension. Since the update rule uses a single variance for the Euclidean distance over all dimensions, there is no longer a clear relation between class variance and dataset variance after normalization.

3.3.3. *Odd Function Solutions.*
We defined final prototypes as solutions $\boldsymbol{\omega}_x$ for which $\Delta_{\text{forces}} = 0$ (see equation (10)). Thus far we have only been able to solve this analytically for the trivial case where the attractive and repulsive forces for each point cancel each other out. While this gives us all solutions for the Gaussian method, we have not been able to find any analytical solutions for the Student's t method.

To uncover more solutions, a colleague suggested assuming that there exist odd function solutions – solutions such that the forces on either side of the final prototype cancel each other out – and using that assumption to find those solutions. For the sake of simplicity, we explain the following for the one-dimensional case.

Let $f(m)$ be the one-dimensional version of the $f(m, n)$ term from equation (11), i.e. it represents the weighted sum of repulsive and attractive forces from a certain point m acting on the left prototype. We start by assuming that an odd function solution exists. In order to find this solution, we take the following steps.

(1) First assume that $f()$ is an odd function around a pivot, point p_x. According to the odd function property, we now have that $f(-m + p_x)$ and $f(m + p_x)$ cancel each other out for any m, i.e. $f(-m + p_x) = -f(m + p_x)$.

(2) Note that the assumption that $f()$ is an odd function guarantees a solution to our $\Delta_{\text{forces}} = 0$ problem, because $\int_{m=-k+p_x}^{k+p_x} f(m)\ dm = 0$, for any k. This formula is true because it can be written as the sum of two integrations that cancel each other out.

(3) Assume that the sums of forces acting on the left final prototype of the odd function solution are pivoted around that prototype, i.e. let the position of the final prototype $\boldsymbol{\omega}_x$ be defined as the pivot of $f()$: $f(-m - \boldsymbol{\omega}_x) = -f(m - \boldsymbol{\omega}_x)$.

(4) Solve the previously derived equation of $f(-m - \boldsymbol{\omega}_x) = -f(m - \boldsymbol{\omega}_x)$ for $\boldsymbol{\omega}_x$, independent of m. The derived equation for $\boldsymbol{\omega}_x$ should be a solution to our $\Delta_{\text{forces}} = 0$ problem.

Unfortunately, this does not give us any solutions. When working out the formula we eventually got stuck, ending up with incompatible sums of exponentials so that further simplification was not possible.

This leaves us with the task of finding out which of our assumptions were wrong. We did this by visualizing the sums of weighted forces, to see what they actually look like.

Recall that we have already shown the unweighted forces acting on the left prototype in Figs. 3 and 4. These images are unweighted in the sense that they do not take the probability density distribution of the dataset into account. So to show the weighted forces, we multiply the forces of the black points from Figs. 3 and 4 with the pdf centered around the mean of the black points, and proceed likewise for the white forces and the pdf of the white points – i.e. we get figures like Figs. 5 and 6, but for different D.

The result is shown in Fig. 7 for the Gaussian method and in Fig. 8 for the Student's t method. It is important to note here that both images show the forces that act when the prototypes are at a stable solution. For the images this means that the sum of all points in the upper and lower images adds up to zero.

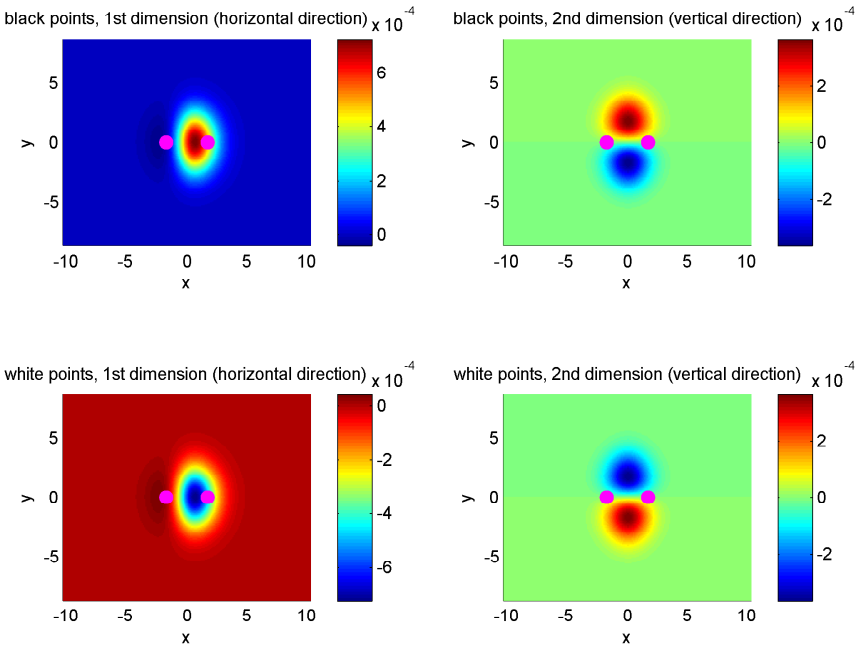

FIGURE 7. The forces of Fig. 3 (Gaussian update rule) weighted by the probability density distributions of the dataset, assuming the prototypes (shown in magenta) are at the first stable solution ($\boldsymbol{\omega}_x = \sqrt{3}$). Meta-parameters are $\sigma^2 = 3.0$, $\sigma_d^2 = 3.0$, $D = \sqrt{3}$.

Note how for the Gaussian image, adding the upper and lower images would result in zero everywhere (i.e. the resultant force at each point becomes zero). Recall from section 3.3.1 that this is a trivial solution.

Note also how the Student's t solution is not a trivial solution. Where the isoclines in the top image are dented to the right, the isoclines in the bottom image are dented to the left. The fact that this is a solution at all is because if you add up the values from each point of the upper and lower image, the resulting sum will be zero (but it is not necessarily zero at each point).

To view this in more detail, see Fig. 9 (Gaussian) and Fig. 10 (Student's t). These images show the resultant force of the horizontal dimension, calculated as the sum of the top left and bottom left images from the previous figures.

For the Gaussian method we can indeed see that this is about zero everywhere (bar random numerical precision noise), while for the Student's t distribution there are some significant deviations (which, however, add up to zero).

Now recall our odd function solution definition: $f(-m - \boldsymbol{\omega}_x) = -f(m - \boldsymbol{\omega}_x)$ for all m. It is in fact the two-dimensional version of this term $f()$ (the resultant force as a function of sample position) that is shown in Figs. 9 and 10. So using these images, we can now say a bit more with respect to odd function solutions.

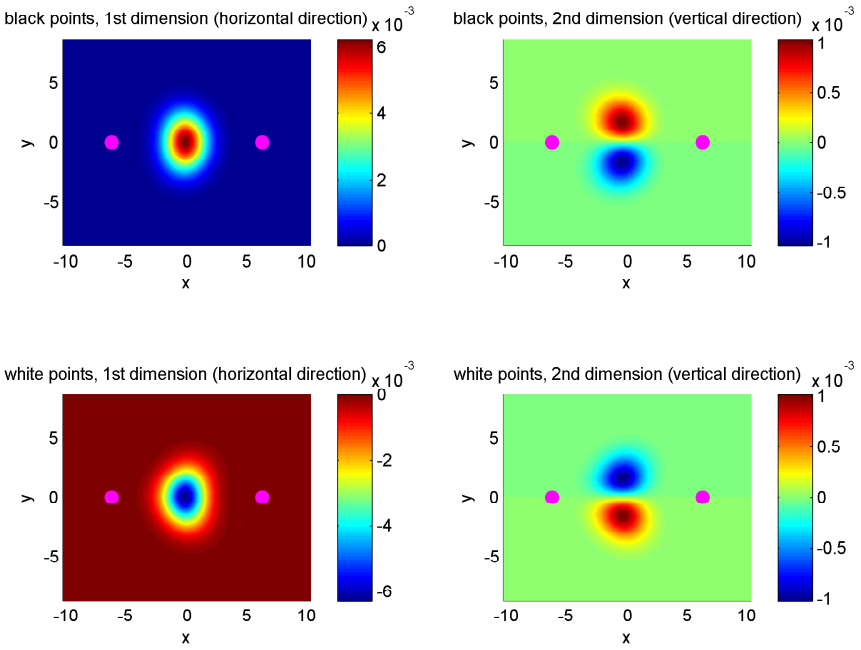

FIGURE 8. The forces of Fig. 4 (Student's t update rule) weighted by the probability density distributions of the dataset, assuming the prototypes (shown in magenta) are at the first stable solution ($\boldsymbol{\omega}_x = 6.306$). Meta-parameters are $v = 3$, $\sigma_d^2 = 3.0$, $D = \sqrt{3}$.

- Theoretically, the Gaussian solution is an odd function solution, because the resultant force is zero everywhere. In fact, every trivial solution is also an odd function solution.
- If the resultant force has significant deviations, and all of these occur within some finite range, then the pivotal point p_x has to lie exactly in the middle of these deviations. Furthermore, these deviations have to satisfy the odd function property, i.e. one side is the additive inverse of the other side.
- For the Student's t method, most significant deviations occur in-between the prototypes rather than centered around one. We will explain this for the white points using the lower left image of Fig. 4. Notice that to the right of the right prototype, the force is pretty much zero everywhere, and to the left of the left prototype, hardly any white points occur (the same holds for the black points in the other direction). Therefore, most significant weighted forces occur in-between the two prototypes.
- For the Student's t method, significant weighted forces indicate significant deviations. This is because we know that the Student's t method has no trivial solutions, so forces never cancel each other out in every point at the same time. We saw in Fig. 8 that this is because the forces never completely overlap, so that gives significant deviations.
- The previous three points imply that for the Student's t method, the significant deviations are not pivoted around the final prototypes, rather they

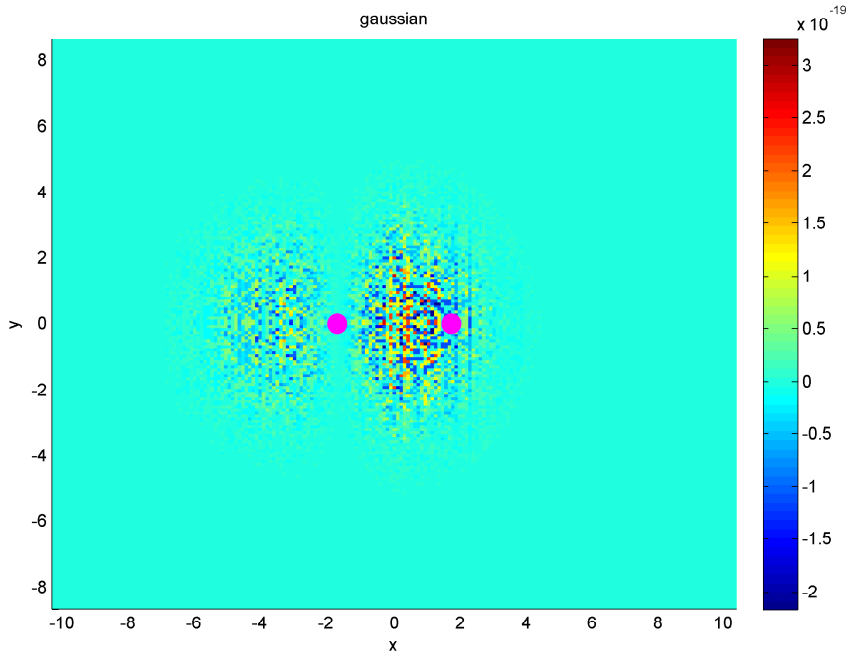

FIGURE 9. The resultant force of the horizontal dimension acting on the left prototype for the Gaussian update rule, calculated as the sum of the top left and bottom left images from Fig. 7. Note that it is basically zero everywhere.

occur in-between the two prototypes. So any odd function solution that assumes the final prototype as pivot would only give solutions close to $x = 0$, which we know is wrong.

So as it turns out, our assumption that for odd function solutions the resultant force would be pivoted around the final prototypes is wrong. This gives us reason to believe that a more interesting definition of an odd function solution would be to find a solution $\boldsymbol{\omega}_x$ for which $f()$ is an odd function around $p_x = 0$.

However, by dropping the assumption that odd function solutions are pivoted around $\boldsymbol{\omega}_x$, we lose any additional information about $\boldsymbol{\omega}_x$ that we gained from the odd function solution assumption. This means that finding odd function solutions is equally complicated as the original problem of finding nontrivial solutions.

We note here that the portrayed Student's t solution does not satisfy this definition of an odd function solution around $p_x = 0$ either. We refer to Fig. 11, which shows the forces of Fig. 10, summed per x so that the result is one-dimensional. This image shows that the left side peaks higher, which is negated by a small negative peak on the same side. Because of this asymmetry, the solution cannot be an odd function solution, no matter what value we choose as pivot.

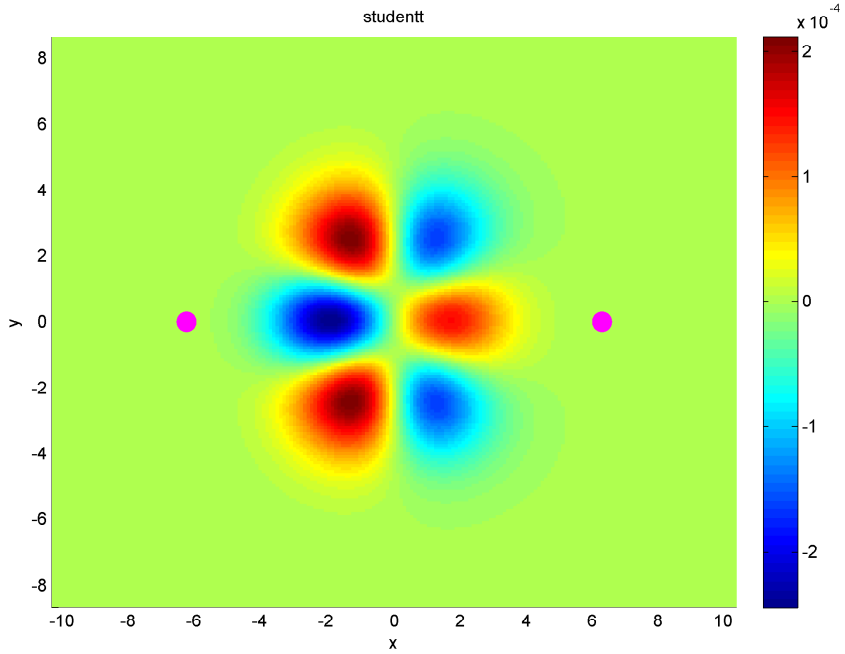

FIGURE 10. The resultant force of the horizontal dimension acting on the left prototype for the Student's t update rule, calculated as the sum of the top left and bottom left images from Fig. 8.

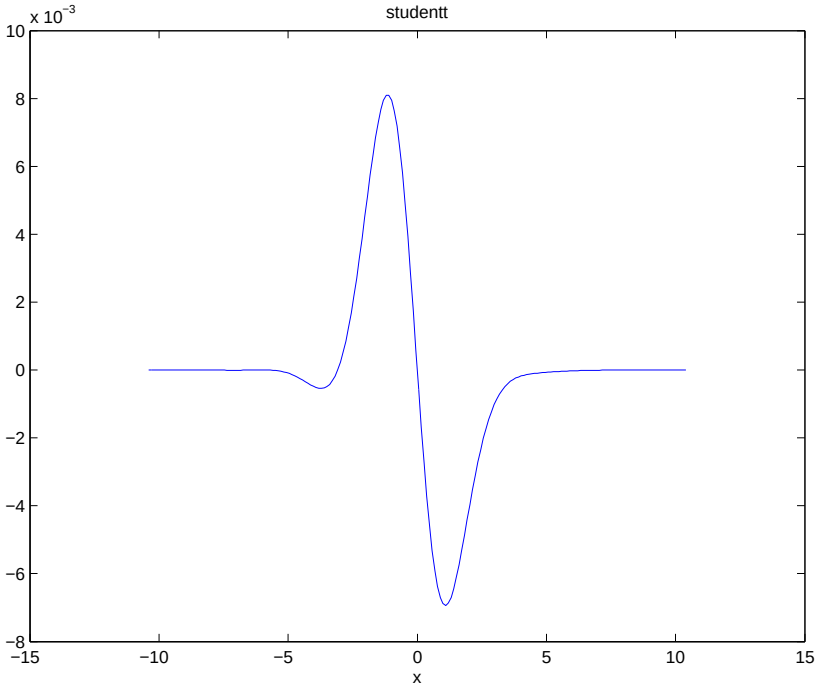

FIGURE 11. The resultant force of the Student's t solution from Fig. 10, summed per x so that the result is one-dimensional.

3.4. **Numerical Approach.**

In this section we discuss numerical solutions to the problem of finding $\boldsymbol{\omega}_x$ for which $\Delta_{\text{forces}} = 0$.

3.4.1. *Solutions By Numerical Integration.*

We modeled the final prototypes as the prototypes that are positioned in such a way that the forces on the prototypes exerted by points in a reasonable area around the prototypes sum up to zero. We furthermore reduced this to linear search by assuming that the prototypes lie on the line with $y = 0$. Approximate final prototype positions can then be found using numerical integration.

In our numerical integrations, we computed forces over a discrete domain. This domain is a sufficiently large and fine-grained rectangular raster of points around the class means. Each point in the domain is essentially treated twice, once as a point of class 0, and once as a point of class 1; where we consider only the forces that act on the prototype of class 0. Summing up all these forces then gives the resultant force acting on this prototype – which is basically an approximation of Δ_{forces}. This resultant force is either positive/negative (indicating a move to the right/left), or it is zero – in the case of final prototypes. By computing these resultant forces over a range of $\boldsymbol{\omega}_x$ values, we were able to find the approximate solutions listed in this section.

By "treating a point as a point of class x", we mean that the force from that point is weighted (multiplied) by a value that expresses the 'probability' that that point actually belongs to class x. This 'probability' is expressed by a probability distribution centered around the mean of class x. This probability distribution is of the type of dataset that is assumed.

The solutions we find through integrations in which the resultant force for each point is zero are exactly those that we found analytically. Recall that we could only find solutions for the Gaussian method in this way, and as it turns out, none of the numerical solutions for the Student's t method satisfy this condition. Numerical integrations show that the trivial solutions we found for the Gaussian method are the only solutions that exist, although we have no direct proof for this claim.

We will now look at the solutions we found numerically (using Matlab). The point of this is not only to find the location of the final prototypes, but also to learn how the meta-parameters of the Student's t method influence the results. Results are shown in Table 1 and Table 2.

$\boldsymbol{\omega}_x$	$D = \sqrt{3}$	$D = 2\sqrt{3}$	$D = 3\sqrt{3}$
$\sigma^2 = 1.0$	0.577	1.155	1.732
$\sigma^2 = 1.5$	0.866	1.732	2.598
$\sigma^2 = 3.0$	1.732	3.464	5.196
$\sigma^2 = 4.0$	2.309	4.619	6.928

TABLE 1. Final prototypes for the Gaussian method. The table shows $\boldsymbol{\omega}_x$ values for which $\Delta_{\text{forces}} = 0$, i.e. the absolute distance from the possible final prototypes to the mean. D is the absolute distance from the class means to the origin. σ^2 is the variance used in the update rule.

For the first distance ($D = \sqrt{3}$), recall that we have also shown the $P(\dots)$ forces (Fig. 1 and Fig. 2) and the effect of the update rule as a whole (Fig. 3 and Fig. 4).

$\boldsymbol{\omega}_x$	$D = \sqrt{3}$	$D = 2\sqrt{3}$	$D = 3\sqrt{3}$
$v = 1$	3.733	4.102	5.460
$v = 3$	6.306	4.073	5.128
$v = 10$	0.961, 3.951, 18.446	2.535, 3.553, 8.024	4.731
$v = 30$	0.721, 5.998, 53.107	1.455, 5.887, 25.597	2.314, 5.818, 15.849

TABLE 2. Final prototypes for the Student's t method. The table shows $\boldsymbol{\omega}_x$ values for which $\Delta_{\text{forces}} = 0$, i.e. the absolute distance from the possible final prototypes to the mean. D is the absolute distance from the class means to the origin. v is the degrees of freedom used in the update rule.

The different rows correspond to different values for meta-parameters (σ^2 for Gaussian and v for Student's t).

As should be clear from the formulas and was confirmed by testing, the α-term (see formulas (6) and (7)) has no influence on the position of the final prototypes, as it merely determines the speed of the learning process (i.e. the size of the adjustments). While the α-term has an amplifying effect on the sum of those adjustments, we are only interested in finding the point where that sum is zero, and α has no influence on that. We further note that the values calculated are ideal values and so the outcomes of an actual RSLVQ execution on a real dataset are likely to vary slightly from these values. But what is more important than the actual values is how these differ for the Gaussian and Student's t methods and what effect the meta-parameters have on them, which is what we will look at next.

If we look at the values in Table 1, we can see that the relation $\boldsymbol{\omega}_x = \frac{D \cdot \sigma^2}{\sigma_d^2}$, that we derived analytically in section 3.3.1 holds for our numerical integration results.

Table 2 shows quite different results. Apparently, the Student's t method has multiple values of $\boldsymbol{\omega}_x$ for which $\Delta_{\text{forces}} = 0$. This means that there may be multiple possible final prototypes. We will later (section 3.4.3) explain that only some of those prototypes are *stable* and actually matter.

3.4.2. *Update Strength As Function Of Prototype Position.*
To get a better view of Δ_{forces} for the Student's t distribution rather than just intersection points, we have visualized Δ_{forces} as a function of $\boldsymbol{\omega}_x$. This is shown in Fig. 12 for $D = \sqrt{3}$, in Fig. 13 for $D = 2\sqrt{3}$ and in Fig. 14 for $D = 3\sqrt{3}$. For comparison, we have also plotted Δ_{forces} when using the Gaussian method, in Figs. 15 to 17.

In the figures of the Student's t method we see that for all cases it holds that values start out negative and become eventually positive going towards infinity (for $v = 30$ in Fig. 12 this is not shown, but from Table 2 we can see that there is another intersection point at $\boldsymbol{\omega}_x = 53.107$). Assuming that this holds in general, it would mean that the number of possible final prototypes is always odd. We will see later in this paper (section 3.4.4), that there are either 1 or 3 possible final prototypes.

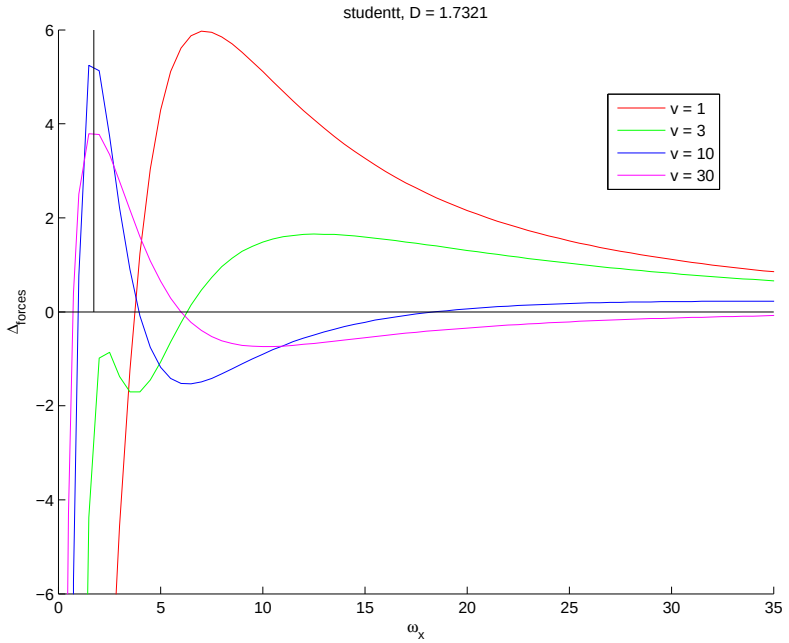

FIGURE 12. Shown is Δ_{forces} as a function of $\boldsymbol{\omega}_x$ for the Student's t method. Here we assume $D = \sqrt{3}$ with respect to the dataset. The possible final prototypes are the intersections with the horizontal axis (i.e. the $\boldsymbol{\omega}_x$ for which $\Delta_{\text{forces}} = 0$). The vertical line is drawn at $\boldsymbol{\omega}_x = D$.

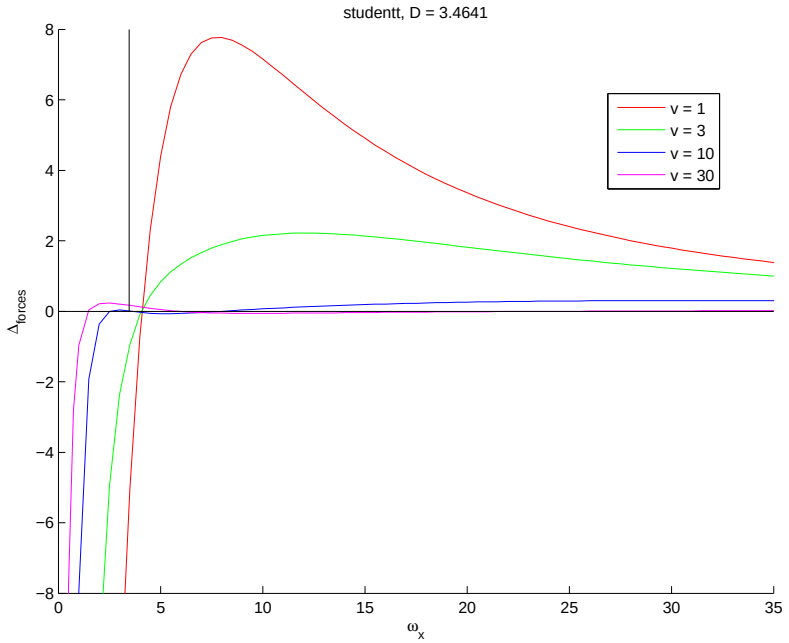

FIGURE 13. Shown is Δ_{forces} as a function of $\boldsymbol{\omega}_x$ for the Student's t method. Here we assume $D = 2\sqrt{3}$ with respect to the dataset. The possible final prototypes are the intersections with the horizontal axis (i.e. the $\boldsymbol{\omega}_x$ for which $\Delta_{\text{forces}} = 0$). The vertical line is drawn at $\boldsymbol{\omega}_x = D$.

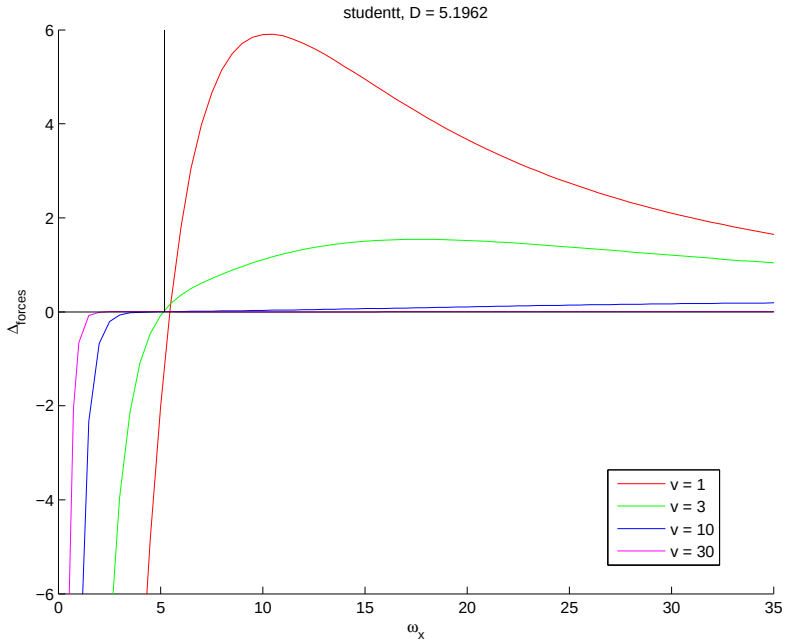

FIGURE 14. Shown is Δ_{forces} as a function of $\boldsymbol{\omega}_x$ for the Student's t method. Here we assume $D = 3\sqrt{3}$ with respect to the dataset. The possible final prototypes are the intersections with the horizontal axis (i.e. the $\boldsymbol{\omega}_x$ for which $\Delta_{\text{forces}} = 0$). The vertical line is drawn at $\boldsymbol{\omega}_x = D$.

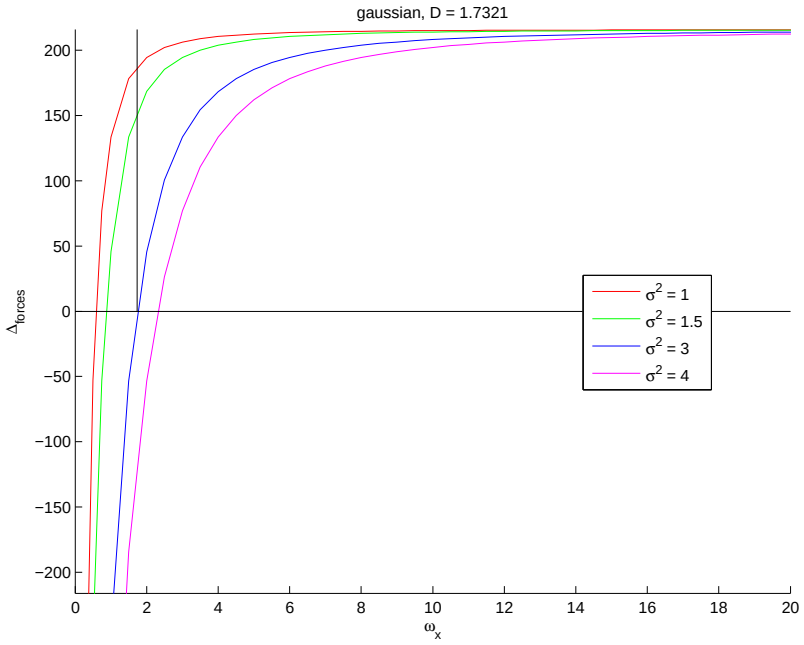

FIGURE 15. Shown is Δ_{forces} as a function of $\boldsymbol{\omega}_x$ for the Gaussian method. Here we assume $D = \sqrt{3}$ with respect to the dataset. The possible final prototypes are the intersections with the horizontal axis (i.e. the $\boldsymbol{\omega}_x$ for which $\Delta_{\text{forces}} = 0$). The vertical line is drawn at $\boldsymbol{\omega}_x = D$.

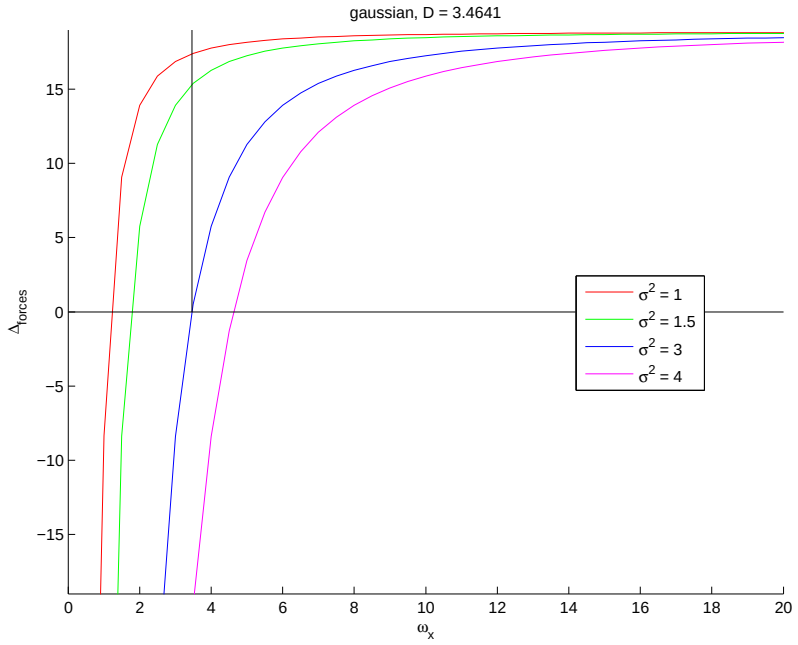

FIGURE 16. Shown is Δ_{forces} as a function of $\boldsymbol{\omega}_x$ for the Gaussian method. Here we assume $D = 2\sqrt{3}$ with respect to the dataset. The possible final prototypes are the intersections with the horizontal axis (i.e. the $\boldsymbol{\omega}_x$ for which $\Delta_{\text{forces}} = 0$). The vertical line is drawn at $\boldsymbol{\omega}_x = D$.

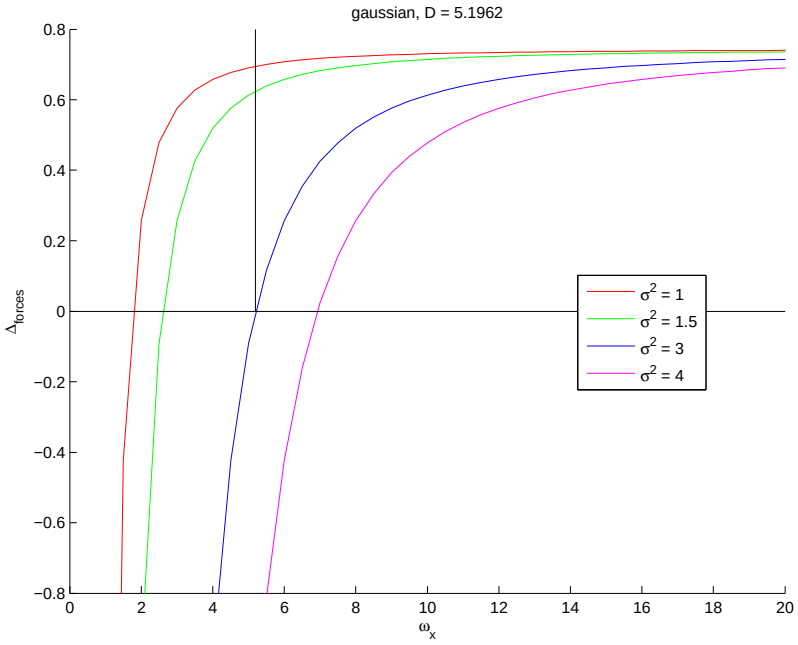

FIGURE 17. Shown is Δ_{forces} as a function of $\boldsymbol{\omega}_x$ for the Gaussian method. Here we assume $D = 3\sqrt{3}$ with respect to the dataset. The possible final prototypes are the intersections with the horizontal axis (i.e. the $\boldsymbol{\omega}_x$ for which $\Delta_{\text{forces}} = 0$). The vertical line is drawn at $\boldsymbol{\omega}_x = D$.

3.4.3. *Stable And Unstable Prototypes.*
It should be pointed out that the Δ_{forces} plots from the previous section actually exhibit two types of final prototypes. Recall that intersection with the x-axis in such a plot means that $\Delta_{\text{forces}} = 0$, meaning that if prototypes are positioned at that specific value of $\boldsymbol{\omega}_x$, forces pulling them to the left and right cancel each other out. Recall also how the Δ_{forces} graphs represent forces exerted on the left prototype only (as the right prototype is assumed to behave in a mirrored way). Then, negative Δ_{forces} values mean that the left prototype will move to the left, while positive Δ_{forces} values mean that the left prototype will move to the right.

We define two types of final prototypes, stable and unstable final prototypes. We define a stable prototype $\boldsymbol{\omega}_x^s$ in the following way:

$$\Delta_{\text{forces}}(\boldsymbol{\omega}_x^s) = 0 \quad \text{with} \begin{cases} \lim\limits_{\boldsymbol{\omega}_x \downarrow \boldsymbol{\omega}_x^s} \Delta_{\text{forces}}(\boldsymbol{\omega}_x) > 0 \\ \lim\limits_{\boldsymbol{\omega}_x \uparrow \boldsymbol{\omega}_x^s} \Delta_{\text{forces}}(\boldsymbol{\omega}_x) < 0 \end{cases}$$

Any other points for which $\Delta_{\text{forces}}(\boldsymbol{\omega}_x^u) = 0$ but do not satisfy the other two properties, we call unstable prototypes.

Stability is an important concept for the following reason. Consider an unstable prototype $\boldsymbol{\omega}_x^u$ (e.g. a line in a Δ_{forces} graph that was positive, crosses zero at $\boldsymbol{\omega}_x^u$ and then becomes negative). Directly to the left of $\boldsymbol{\omega}_x^u$, the line is positive. This means that if the RSLVQ algorithm was to approach $\boldsymbol{\omega}_x^u$ from the right (for the left prototype), then further updates would push it away from this final prototype. Also, if the left final prototype is approached from the left, we look at values to the right of $\boldsymbol{\omega}_x^u$ in the graph, these values are negative, so updates will always push the prototype to the left, again, away from the final prototype.

We conclude that when the RSLVQ algorithm is close enough to a stable prototype, updates will always move it in the direction of that stable prototype, causing the algorithm to converge to that prototype. We also conclude that it is safe to assume that implementations of the RSLVQ algorithm will never converge to unstable prototypes, since the algorithm works iteratively, and whenever the algorithm is near an unstable prototype, updates move further away from it.

We conclude that convergence of the RSLVQ algorithm, i.e. where the prototypes stabilize around a certain area, can only happen for stable final prototypes.

Recall how we saw in the previous section that in all integration plots, with increasing $\boldsymbol{\omega}_x$, Δ_{forces} always eventually became positive (sometimes approaching zero in infinity). This corresponds to infinity being either an unstable final prototype or no final prototype at all. So in theory, the RSLVQ algorithms will never have diverging prototypes. Although we will see later (section 4.1.4) that actual datasets can differ a bit from the integration results.

We also saw that all Δ_{forces} plots start out negative, so the first intersection is always a stable final prototype.

Recall that we have seen the Student's t method having either 1 or 3 final prototypes. For the case of having 3 possible final prototypes, only the first and last are stable. We will later see that in practice, the algorithm will always converge to the first, because it is closest to the starting position. For the Gaussian method, we always only have one possible final prototype, which is stable.

3.4.4. *Final Prototype Position As Function Of D.*

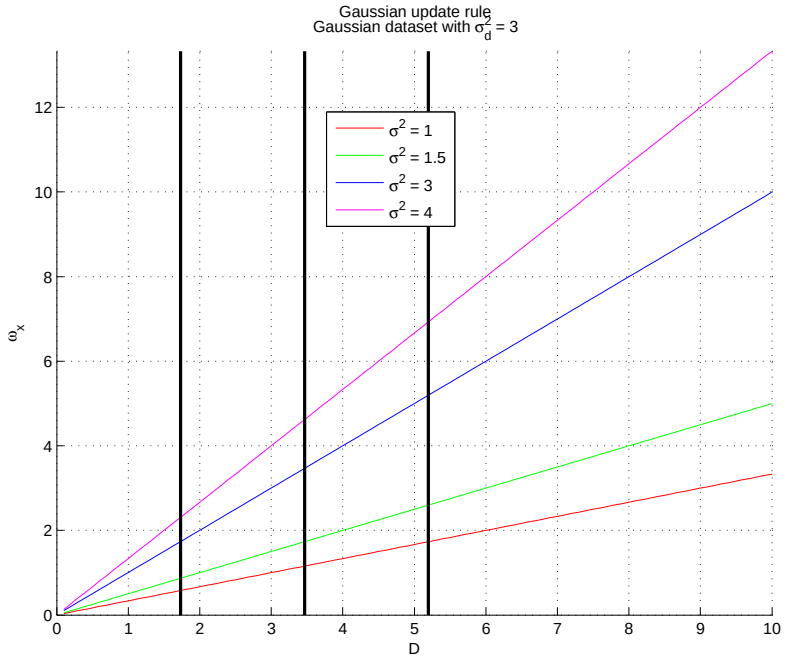

FIGURE 18. The position of the first stable prototype ($\boldsymbol{\omega}_x$) as a function of D, for the Gaussian method. The variance of the dataset was fixed at $\sigma_d^2 = 3$. From left to right, the three vertical lines indicate $D = \sqrt{3}, 2\sqrt{3}$ and $3\sqrt{3}$ respectively.

We present graphs that show the distance of the first stable final prototype, as a function of D. These graphs are shown in Fig. 18 for the Gaussian method, and Fig. 19 for the Student's t method.

These figures show the 'first' stable final prototype. This denotation is relevant only for the Student's t method, which as we know may have multiple stable final prototypes. In such cases, by 'first' prototype, we mean the prototype with the lowest value for $\boldsymbol{\omega}_x$. We also note that since all Δ_{forces} graphs start out negative, the first possible final prototype is always a stable one.

Looking at the plot for the Gaussian method (Fig. 18), there are no surprises. It is clear that the relation we derived earlier, $\frac{\sigma^2 D}{\sigma_d^2} = \boldsymbol{\omega}_x$ holds. This equation tells us that if we keep the two variances fixed, there is a linear relation between D and $\boldsymbol{\omega}_x$. The linearity of this relation is confirmed by the straight lines in the figure. Also note that when the variance of the update rule and the variance of the dataset are equal, then $D = \boldsymbol{\omega}_x$ (meaning that prototypes converge to the class means).

The plot for the Student's t method (Fig. 19) gives some new insights. The lines start out ascending, slightly faster than linear. At some point these lines suddenly stop, and the lowest prototype is subsequently found in a different segment, starting out high, then reaching a minimum and eventually becoming almost linear, asymptotically approaching $D = \boldsymbol{\omega}_x$.

These lines are made up of two segments, the first of which we call A, the second B. From integrations we know that the B segment of stable final prototypes did

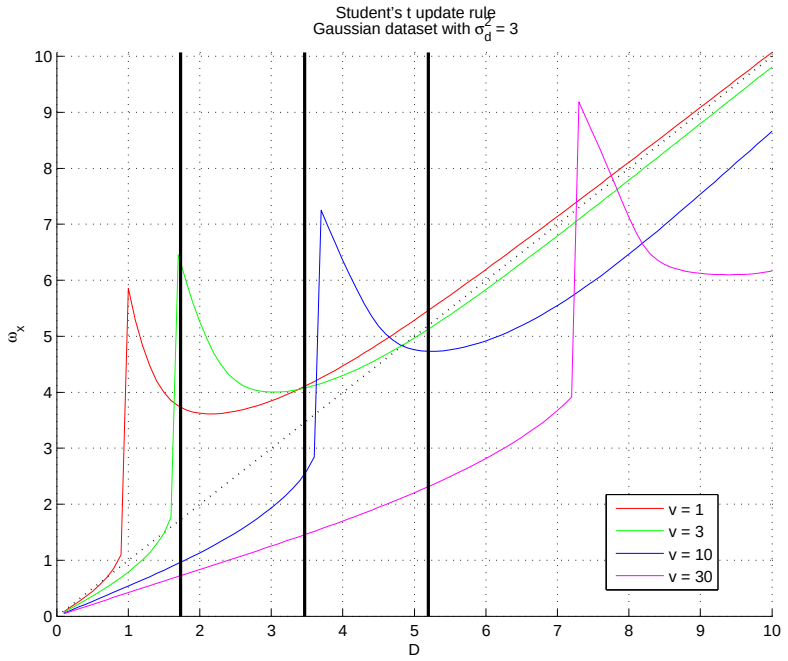

FIGURE 19. The position of the first stable prototype ($\boldsymbol{\omega}_x$) as a function of D, for the Student's t method. The variance of the dataset was fixed at $\sigma_d^2 = 3$. From left to right, the three vertical lines indicate $D = \sqrt{3}, 2\sqrt{3}$ and $3\sqrt{3}$ respectively. Note that the near vertical line segments do not represent actual values, but are interpolations spanning the jump discontinuities.

not simply appear at a specific point – it has been there all along, although at low values of D, its values are extremely high. This means that to the left of where B starts, the segment actually extends, and there are stable final prototypes as well, but they were never the first ones.

The segment A, which gives the first stable final prototypes, does disappear at some point. We think that this point is defined by the decreasing distance between segments A and B. Recall that for A and B to define stable prototypes, in between there has to be another intersection defining an unstable prototype. When the distance between A and B becomes too small, there simply is no room for three intersections, so the first segment, A, dissolves, and with it, one stable and one unstable final prototype.

Let us define the D_{jump} as the D where segment A ends and segment B begins. Note that D_{jump} depends on v and σ_d^2. We will see later (section 4.1.4) that the Student's t method on datasets with $D \approx D_{\text{jump}}$ can converge at the $\boldsymbol{\omega}_x$ of A, at the $\boldsymbol{\omega}_x$ of B or anywhere in between. This causes slow convergence and a large variance in final prototype position.

From Fig. 19 we can see that this happens for instance for the Student's t method with $v = 3$ on datasets with $D = \sqrt{3}$. This is further illustrated in Fig. 12, which we now know was taken at a D where the first peak for $v = 3$ just became negative.

Another such case shows how the second dataset ($D = 2\sqrt{3}$) was apparently taken right before segment A of the $v = 10$ line dissolved. We see this in Fig. 13, where we can just see the blue line dent below zero, while this is no longer the case at higher D (e.g. Fig. 14).

Looking a bit closer we notice how for segment A, the parameter v determines mostly the slope of the segment, higher v meaning less steep. While in segment B, all segments become more or less equally steep, and the v parameter determines the initial distance from the $D = \boldsymbol{\omega}_x$ line, higher v leading to greater distance from this line. We think that all v eventually approach the $D = \boldsymbol{\omega}_x$ line (some faster than others) – meaning that for higher D, prototypes converge closer to the class means. While the slope of the segments A decreases with increasing v, this relation does not seem to be linear, and nor is the relation between D_{jump} and v.

We note that choosing a different value for σ_d^2 (the variance of the dataset) here does not produce radically different results. Surely the values change due to the scaling, but for instance the $v = 1$ line still approaches the $D = \boldsymbol{\omega}_x$ line.

We now believe that the two segments we identified (A and B), are the only two segments that generate stable final prototypes for the Student's t method. In other words, at low D, all functions start out with having three possible final prototypes, two of them stable and defined by segment A, and segment B extended to the left. Then there will be a crossover point D_{jump}, and after the crossover point segment B is the only segment that gives (stable) final prototypes. While we have no definite proof for this claim, it is supported by our data.

4. Experiments And Results

In this section we discuss results and conclusions drawn from actual runs of the RSLVQ algorithm (as opposed to theoretical or numerical integration results). We first run the RSLVQ methods on simple artificial datasets, distributed according to the same dataset assumptions we used in our analysis. We then proceed to more complicated artificial datasets, and finally we look at how the RSLVQ methods perform on a real-life dataset.

4.1. **Artificial Datasets.**

In this section we look at results from actual runs of the RSLVQ methods, using datasets as described in section 2.4.

4.1.1. *RSLVQ Settings.*

We use datasets as described in section 2.4. In our tests we use three different D's: $D = \sqrt{3}$, $D = 2\sqrt{3}$ and $D = 3\sqrt{3}$. For each D, we generate five i.i.d. (independent and identically distributed) datasets, each having 1000 points per class. This gives us 15 datasets in total. For each dataset, we randomly select two-thirds of the datapoints to use as trainingset, and one-third to use as testset. We do this three times per dataset, so that we get 45 (hopefully distinct) trainingset/testset pairs. This gives us 45 runs per method (either Gaussian or Student's t), so 90 runs in total.

The RSLVQ algorithm was set to assume two classes, using one prototype per class. It does not perform dataset normalization. We let the algorithm run for $\texttt{n\_epoch} = 200$ epochs.

For the Student's t method we use $v = 3$ (degrees of freedom), and for the Gaussian method we use $\sigma^2 = 3.0$ (variance). Note that for the Gaussian method, this should make the final prototypes coincide with the class means.

During training, we let the learning rate (α_1 from equation (6) and α_2 from equation (7)) decay linearly over the epochs, according to the following formula:

$$\alpha(\texttt{epoch}) = \texttt{max}(\alpha_{\text{start}} - (\texttt{epoch} - 1) * \texttt{decay\_factor}, \alpha_{\min}) \tag{13}$$

where $\texttt{decay\_factor} = \frac{\alpha_{\text{start}}}{\texttt{n\_epoch}}$ and $\alpha_{\min} = 0.025 * \alpha_{\text{start}}$. The formula implies a linear decay of α over the first 97.5% of the epochs, after which it will remain constant.

Note that the only free variable in all of this is α_{start}. We used $\alpha_{\text{start}} = 0.15$ for the Student's t method, and $\alpha_{\text{start}} = 0.075$ for the Gaussian method. These values were chosen using trial and error, such that the accuracy fluctuations during training would roughly be the same for both methods. The decay of α is illustrated in Fig. 20.

4.1.2. *Results.*

We will look at the accuracy of the prototypes on the testset, averaged over the last 100 epochs of all 15 runs. This tells us how well the prototypes derived from the trainingset represent the full dataset. We will also visualize where the prototypes of the different runs end up and what decision boundaries they define. Since we know from our numerical integration results where the prototypes should end up given an infinitely large dataset, the results could show us the extent in which a reduced dataset affects the positioning of the final prototypes (i.e. how robust the methods are).

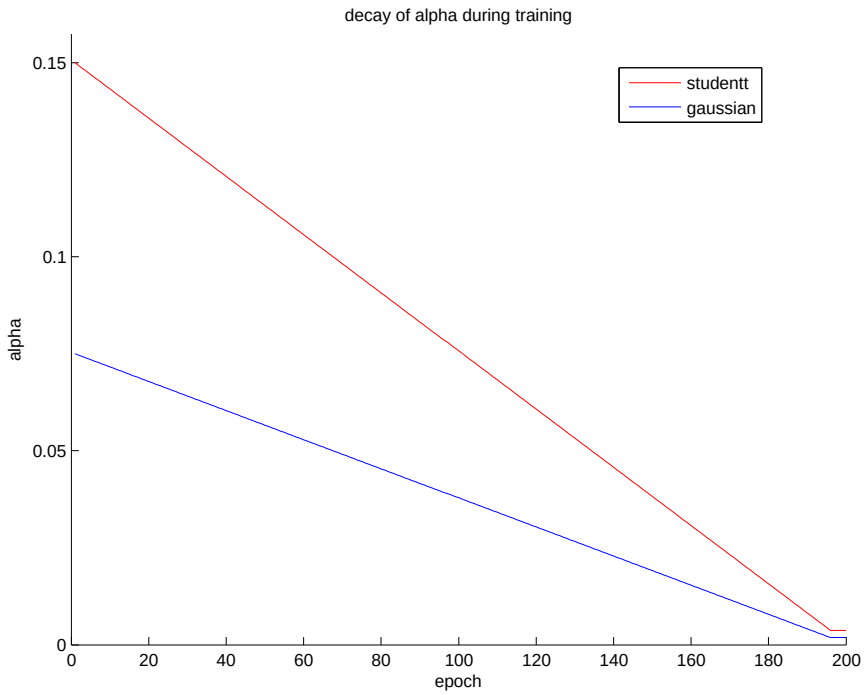

FIGURE 20. The decay of α according to equation (13).

accuracy	$D = \sqrt{3}$	$D = 2\sqrt{3}$	$D = 3\sqrt{3}$
Gaussian	0.845396	0.977211	0.994826
Student's t	0.845320	0.978667	0.996875

TABLE 3. Table of accuracies measured on the testset, averaged over the last 100 epochs of 15 runs.

$\boldsymbol{\omega}_x$	$D = \sqrt{3}$	$D = 2\sqrt{3}$	$D = 3\sqrt{3}$
Gaussian	1.721 (0.013)	3.258 (0.136)	5.776 (0.749)
Student's t	5.589 (0.453)	4.097 (0.009)	5.164 (0.003)

TABLE 4. Table of average $\boldsymbol{\omega}_x$ value and its variance in parentheses. The values were taken using prototypes after 200 epochs of training, averaged over 15 runs. Ideally, these results should match the second row of Table 2 for Student's t, and the third row of Table 1 for Gaussian.

Table 3 lists the accuracies averaged over the last 100 epochs of 15 runs for the Gaussian and Student's t methods. Variances were very low, all values were within a couple percent of the mean. These values are very close to the theoretical maximum, i.e. the percentage of overlap of two normal distributions (0.8413 for $D = \sqrt{3}$, 0.9772 for $D = 2\sqrt{3}$ and 0.9987 for $D = 3\sqrt{3}$). However, what matters is not the position of the prototypes, but the position of the decision boundary between them.

Table 4 shows the values of $\boldsymbol{\omega}_x$ after 200 epochs of the RSLVQ algorithm, averaged over 15 runs. Low variance means that the result is accurate and that most runs converged close together, high variance means that the results changed a lot for different runs and that some runs did not converge at all. We can see that for

$D = \sqrt{3}$, the Gaussian method has lower variance, while the Student's t method has lower variance for $D = 2\sqrt{3}$ and $D = 3\sqrt{3}$. Recall that the high variance of the Student's t method for $D = \sqrt{3}$ is because it is close to D_{jump}, as described in section 3.4.4.

If we compare the results from Table 4 with our numerical integration results from Tables 1 and 2, then we see that where the variance in the results is low, our predictions are quite accurate. When the variance in the results is high, and some runs do not converge, we see a greater deviation from our predictions.

The Figs. 21, 22, 23, 24, 25 and 26 show the position of the prototypes after 200 epochs.

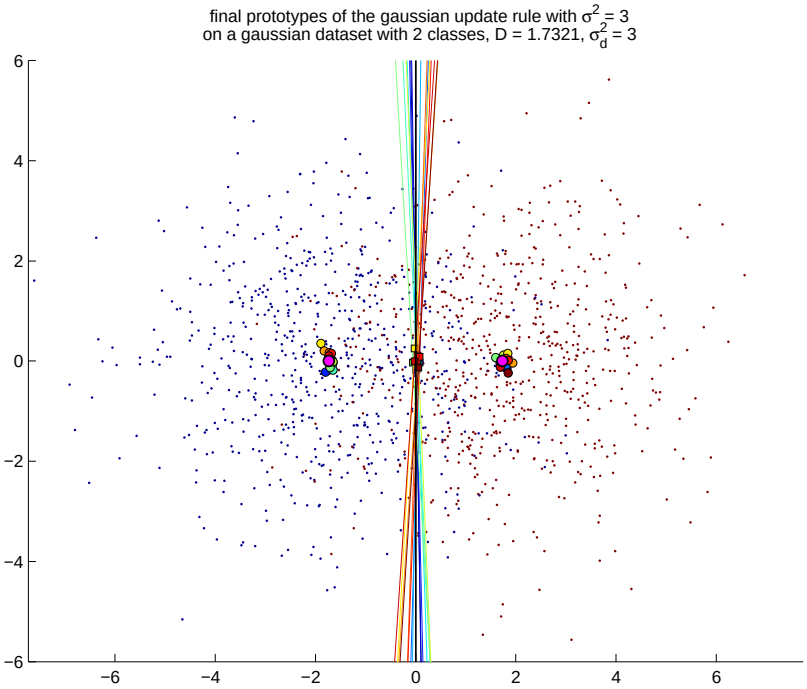

FIGURE 21. The final prototypes of the Gaussian method for 15 i.i.d. runs (each in a different color) on datasets with 2 classes. The squares on the decision boundaries are the means of the two prototypes for each run. The small points are one of the datasets used, and serve to illustrate the scale. The class means of the dataset are shown in magenta, with a thickened border. Here, $D = \sqrt{3}$ is the distance from the class means to the origin.

For $D = \sqrt{3}$ (Figs. 21 and 22), the Student's t method converged with low speed and accuracy (different runs converge to different positions). Furthermore, the prototypes converge far away from the class means. Also, the prototype means show a vertical spread, so some are not exactly in between the class means of the dataset. We recall from section 3.4.4 that this is because $D = \sqrt{3} \approx D_{\text{jump}}$ for the Student's t method with $v = 3$. However, surprisingly, the spread of the decision boundaries is not noticeably worse than that of the Gaussian method.

For $D = 2\sqrt{3}$ (Figs. 23 and 24), we can see that even though the prototypes of the Gaussian method are still closer to the class means on average, they converge with less accuracy and the spread of the decision boundaries is larger than for the Student's t method.

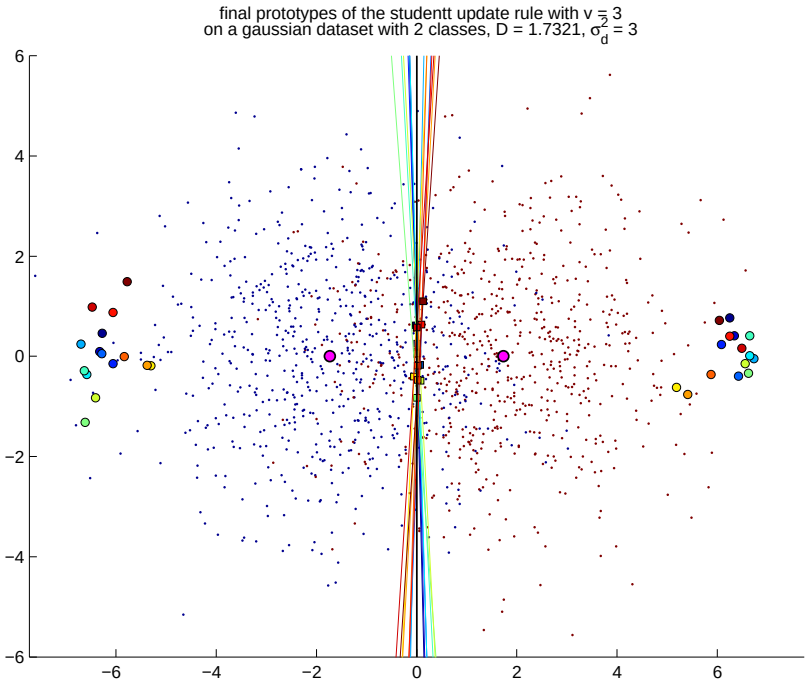

FIGURE 22. Similar to Fig. 21, for the Student's t method and $D = \sqrt{3}$.

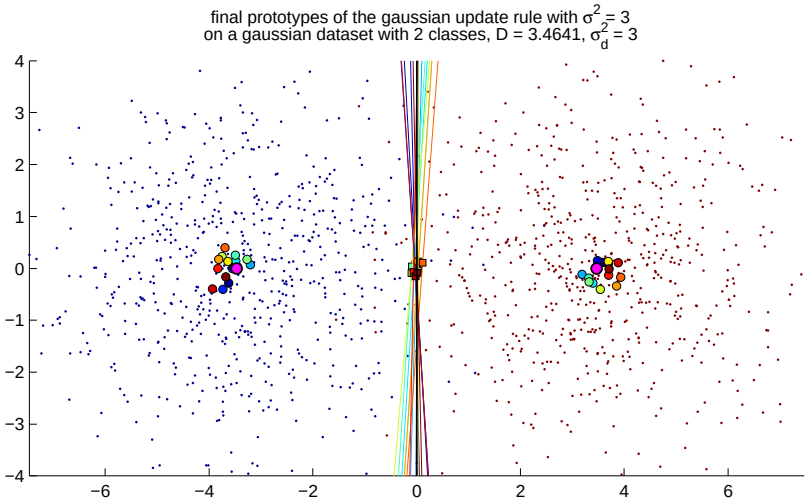

FIGURE 23. Similar to Fig. 21, for the Gaussian method and $D = 2\sqrt{3}$.

For $D = 3\sqrt{3}$ (Figs. 25 and 26), we note that for the Gaussian method the prototypes moved very erratic. On some runs they did not even converge (no signs of slowing down). For the runs that did converge, the area of convergence was very large (i.e. covering a quarter of the dataset). However, according to our numerical results, all runs at any reasonable distance should converge. Our integrations show that the Δ_{forces} term for this D is very small around the intersection point. It is so small that in practice, the Gaussian method at this D relies too much on individual outliers that may or may not be present in a relatively small dataset (recall section 3.1.3). This explains why some runs converge and some do not: different runs

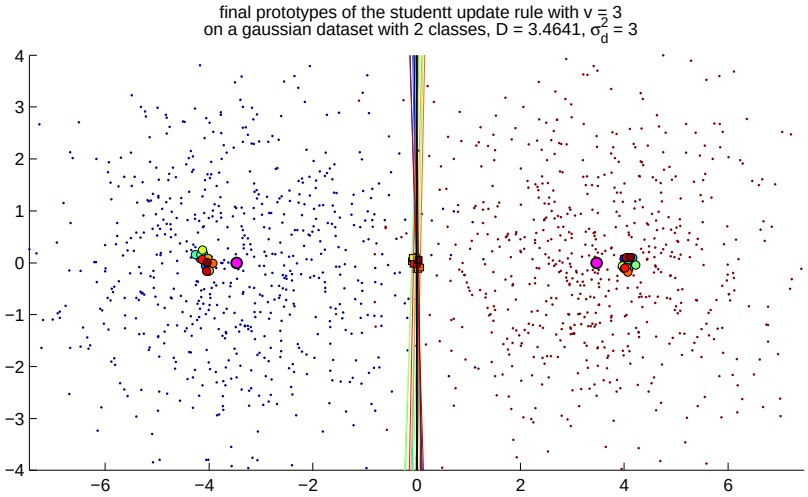

FIGURE 24. Similar to Fig. 21, for the Student's t method and $D = 2\sqrt{3}$.

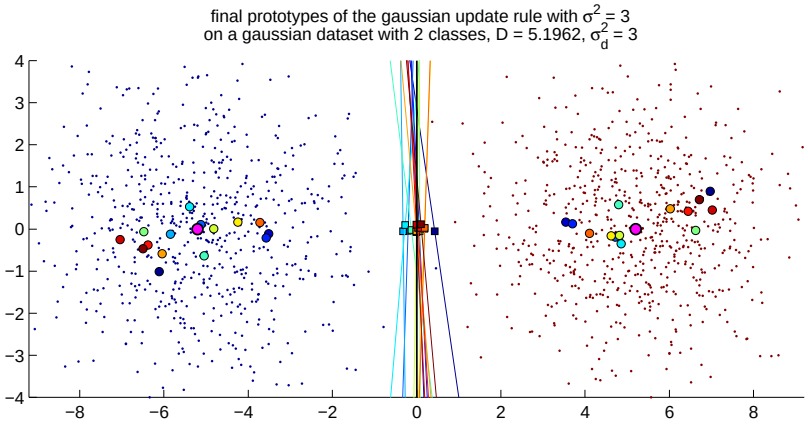

FIGURE 25. Similar to Fig. 21, for the Gaussian method and $D = 3\sqrt{3}$.

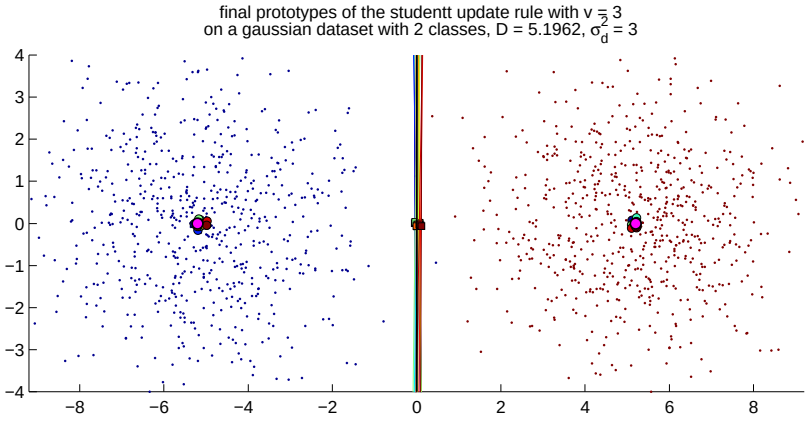

FIGURE 26. Similar to Fig. 21, for the Student's t method and $D = 3\sqrt{3}$.

use different trainingsets, which may contain different outliers. This is a result of having a reduced dataset and the update rule taking very few points into account.

As a consequence, we see a horizontal spread for the decision boundaries near the middle, which negatively impacts performance. On the contrary, the Student's t method converges very close to the class means, with high accuracy. This is again because the Student's t update rule takes more points into account (section 3.1.3).

Moreover, in general, our numerical approximations for the final prototypes of the Student's t method were more accurate than those for the Gaussian (not only was the average deviation much smaller, also the mean position for the Student's t prototypes averaged over the 15 runs had a lower variance than those of the Gaussian). Again, this can be explained by the fact that for the Student's t update rule, a lot of points in the dataset affect the position of the prototypes, while for the Gaussian update rule only a few do. We also note that for higher D, the final prototypes of the Student's t method start converge closer to the class means, like we predicted in section 3.4.4.

For the Gaussian method, we should look at D always in context of the amount of overlap. We can express this by the formula:

$$r_{\text{overlap}} = \frac{D}{\sigma_d} \tag{14}$$

where σ_d is the standard deviation of the classes in the dataset, and D is the absolute distance of a class mean to the origin. We know that the Gaussian rule has problems converging when there is hardly any overlap. This corresponds with $r_{\text{overlap}} \gtrapprox 2$. We note here that choosing a higher variance for the Gaussian method (i.e. $\sigma^2 = 6$) improves the accuracy of the decision boundaries only very slightly. It does not solve the convergence problems so a lot of the runs will still not converge.

We conclude that on these datasets, the Student's t method performs equal to or better than the Gaussian method, if all we look at is the accuracy in determining the decision boundary. If the position of the prototypes relative to the class means is deemed more important, then the Gaussian method is a clear winner for all but $D \geq 3\sqrt{3}$.

4.1.3. *Speed Of Convergence.*
From actual runs of the RSLVQ algorithm, we have deducted that the speed of convergence is related to the slope around the intersection of the final prototype in the Δ_{forces} graphs. Reason for this is that when a line is steep close to its intersection, forces around the final prototype quickly increase, so updates are performed in large steps, leading to faster convergence.

We can see this for instance, looking at Figs. 15, 16 and 17, where the steepest line has the best convergence properties. Also, Figs. 12, 13 and 14 show that the line with $v = 1$ clearly has the best convergence properties for the Student's t distribution. So if we do not know anything about the distribution of the dataset, using $v = 1$ for the update rule is our safest bet.

Speed of convergence is a quantity we mean to measure in terms of how many iterations it takes before the RSLVQ algorithm reaches its converging area, which depends on how large the update steps are in relation to the distance between start and end point. By accuracy of convergence we mean how close final prototypes of different datasets (instanced using the same parameters) lie together. While the speed of convergence helps to ensure that prototypes converge, the accuracy of convergence depends on other factors, as we will see in the next section.

4.1.4. *Accuracy Of Convergence.*
To explain some of the unexpected results and to get a better understanding of how

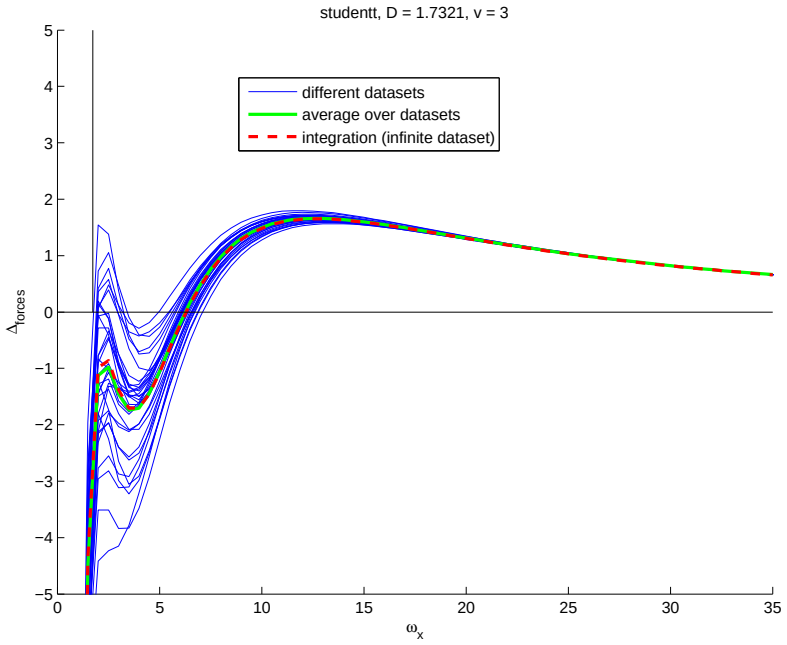

FIGURE 27. Shown is Δ_{forces} as a function of $\boldsymbol{\omega}_x$ for the Student's t method (with $v = 3$) on the dataset with $D = \sqrt{3}$. The possible final prototypes are the intersections with the horizontal axis (i.e. the $\boldsymbol{\omega}_x$ for which $\Delta_{\text{forces}} = 0$). The vertical line is drawn at $\boldsymbol{\omega}_x = D$.

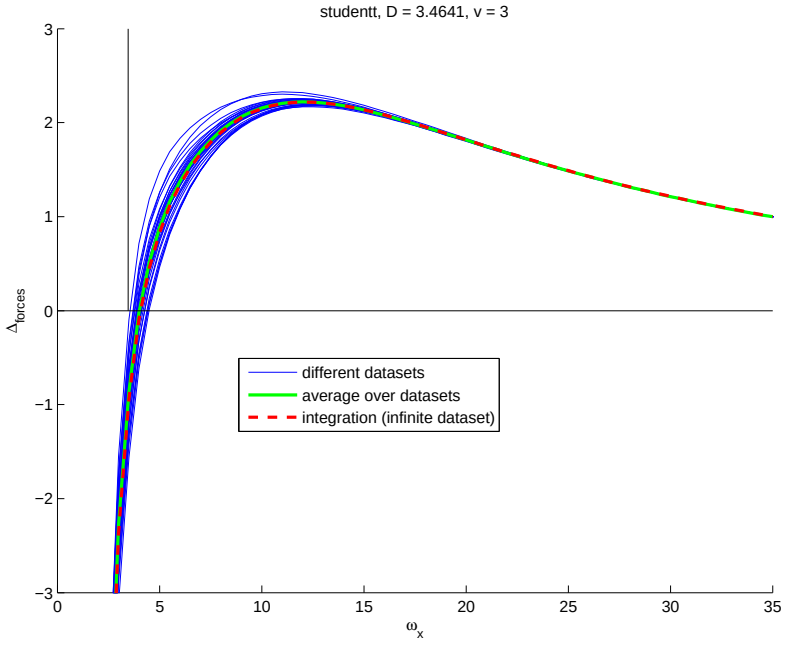

FIGURE 28. Similar to Fig. 27, for the Student's t method and $D = 2\sqrt{3}$.

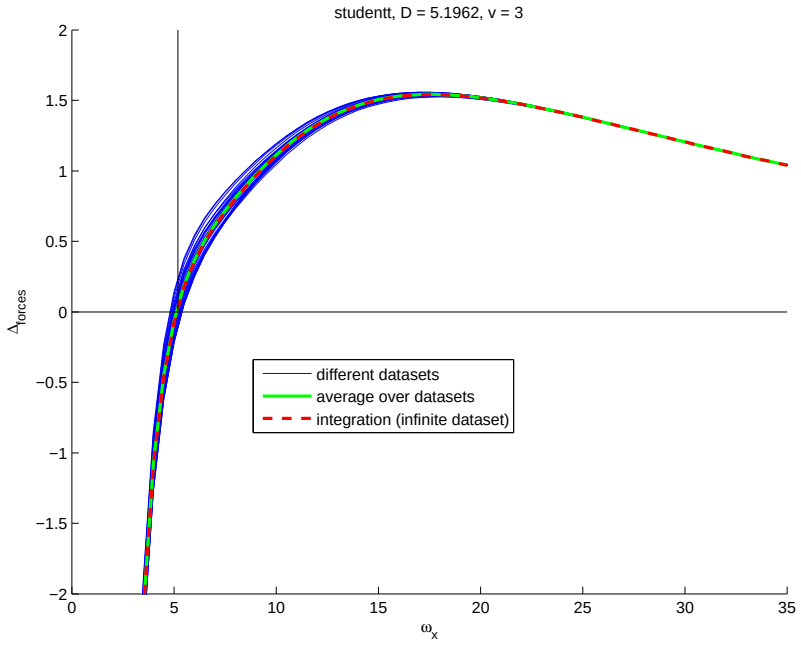

FIGURE 29. Similar to Fig. 27, for the Student's t method and $D = 3\sqrt{3}$.

the finite size of the datasets influences the final prototypes, we have made some plots (Figs. 27 to 32). In these plots we compare the results of 32 i.i.d. datasets (the blue lines), each consisting of 2000 points, i.e. 1000 points per class; the average of these datasets (the green line); and the results of our numerical integrations (the red dotted line), which use a probability density distribution rather than an actual dataset, and should thus better represent an infinite dataset. These figures are similar to those we showed in section 3.4.2, except here we use one meta-parameter setting and results from different datasets to show the variance in results.

In these figures, we compare our numerical integration results with actual, generated datasets. We use these images to show two things. Firstly, we can immediately see that in all figures, the green line is very close to the red line. This is a confirmation that our integration results are very close to the average of the actual datasets. And thus that our way of modeling two classes using points on a rectangular grid and treating each point twice is quite accurate. The images in this section also serve to show what the variance of the final prototypes is, i.e. how much two i.i.d. datasets can differ with respect to the position of their final prototypes.

We applied scaling (by standard deviation) so that the lines could be viewed in the same figure. For scaling the individual blue lines, we assumed the standard deviation of their average. Scaling is allowed here, since we are interested in where the Δ_{forces} quantity intersects the horizontal axis, not in what its absolute values are.

We start with the images of the Student's t method, and with probably the most interesting image, Fig. 27, which gives some remarkable insights. We notice a high variance in results around the first two local extrema. This is especially relevant since some blue lines are shown to have three intersections with the horizontal axis, while others have only one. This means that different instances of exactly the same distribution may have final prototypes at very different positions. Furthermore, we see that for some datasets, Δ_{forces} stays close to zero for a while (the line does not go

very far into the negative), which as we know generally results in slower convergence. This explains our earlier observations of the Student's t method having a high variance in converging on these datasets (Table 4), and why some runs converged very slowly.

The reason why there is such a large variance at lower $\boldsymbol{\omega}_x$ values is because $D = \sqrt{3} \approx D_{\text{jump}}$ for the Student's t method with $v = 3$, as described in section 3.4.4. We note that even in the high variance part, our integration results match the average over the datasets very closely.

In Figs. 28 and 29 around the intersection we see a tight bundle of steep lines. This indicates fast and accurate convergence for all runs, confirming what we found earlier when using the actual RSLVQ implementation.

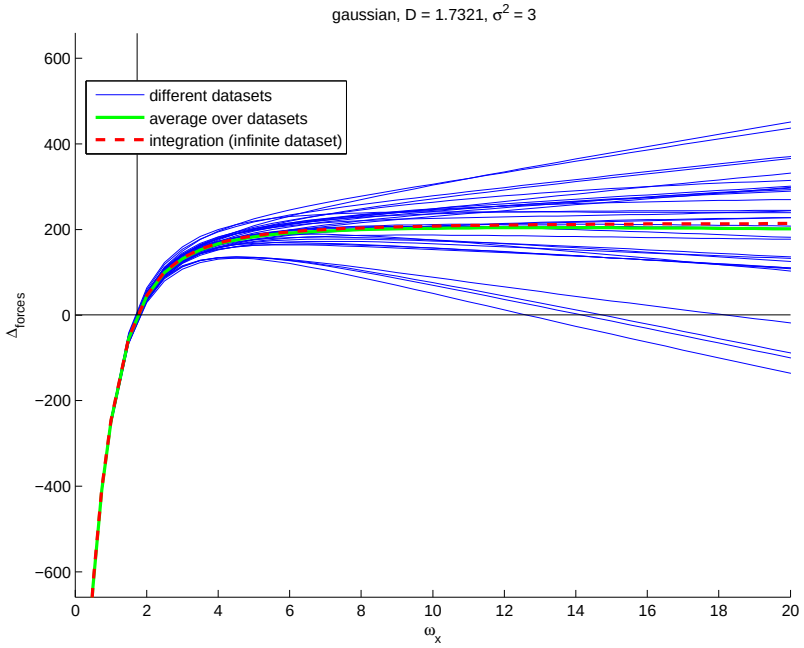

FIGURE 30. Shown is Δ_{forces} as a function of $\boldsymbol{\omega}_x$ for the Gaussian method (with $\sigma^2 = 3$) on the dataset with $D = \sqrt{3}$. The possible final prototypes are the intersections with the horizontal axis (i.e. the $\boldsymbol{\omega}_x$ for which $\Delta_{\text{forces}} = 0$). The vertical line is drawn at $\boldsymbol{\omega}_x = D$.

For the Gaussian method we start with Fig. 30. This image shows that at high $\boldsymbol{\omega}_x$, individual datasets start to fan out (their average is still close to our integration results). The blue lines fan out because of the amplifying effect that the $(\boldsymbol{\xi} - \boldsymbol{\omega}_j)$ term in the Gaussian update rule (equation (6)) has. This term makes the repulsive and attractive forces increase linearly with $\boldsymbol{\omega}_x$.

Fig. 31 shows a different picture. Here, because of the increased distance between the datapoints of different classes, the update forces have become much more weak (as fewer points influence the position of the final prototypes). This causes a lot of blue lines to quickly become negative again, and in some cases, never to intersect with the x-axis at all. Actual runs confirm that for those datasets, the prototypes keep diverging, and thus do not converge to a single position. Also note how the diverging of the blue lines starts earlier for this D. This is because the intersection occurs later in the curve (compared to Fig. 30), closer to its maximum.

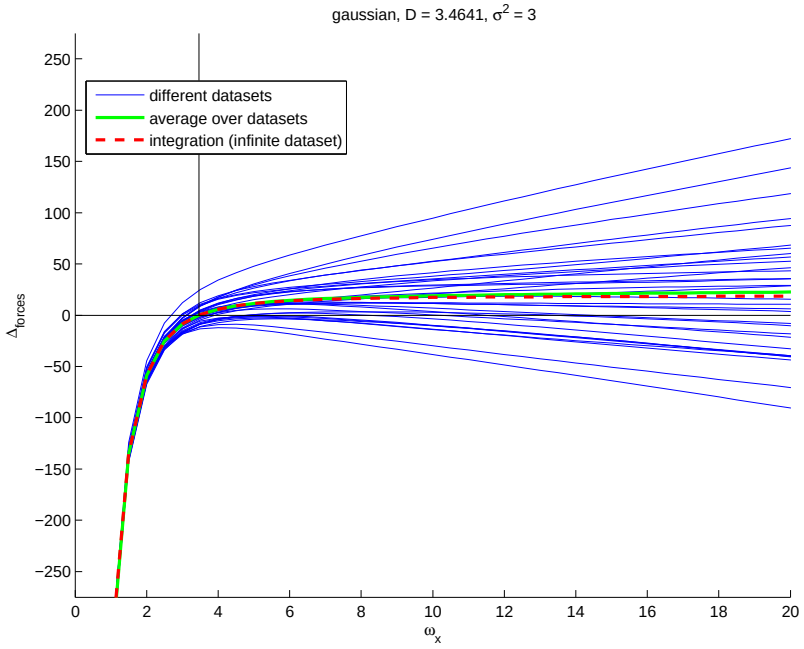

FIGURE 31. Similar to Fig. 30, for the Gaussian method and $D = 2\sqrt{3}$.

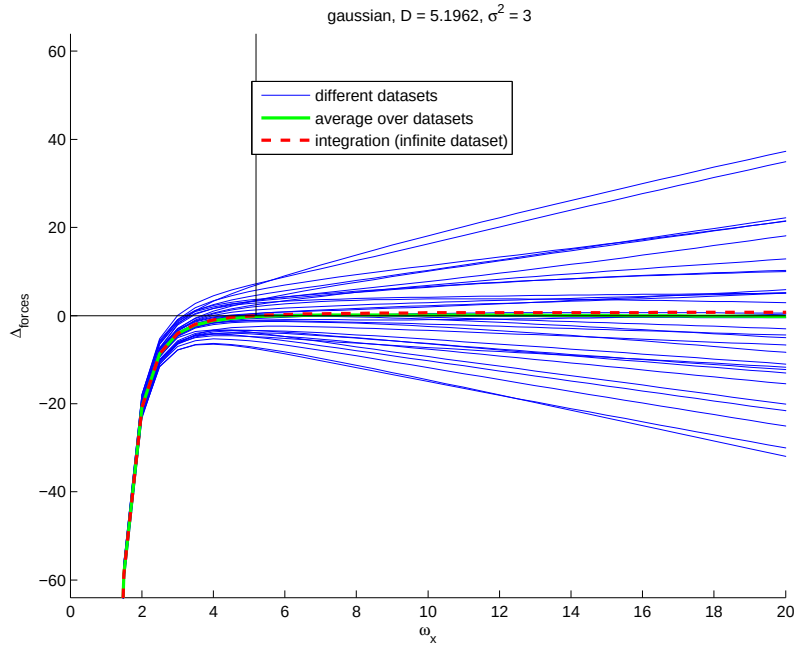

FIGURE 32. Similar to Fig. 30, for the Gaussian method and $D = 2\sqrt{3}$.

The same holds for Fig. 32. We saw earlier (e.g. Fig. 25) that many of the runs at this D did not converge. Some did converge however, and here we can see why. Non-steep forces with a high variance around the intersection are causing many of the blue lines never to reach the x-axis, and thus have prototypes that will always keep diverging. In this context, a blue line that never becomes positive means

that the left prototype will always be pushed to the left, assuming that the right prototype is similarly pushed to the right; ergo, the prototypes always diverge.

4.2. **More Complicated Datasets.**
In this section we compare how the RSLVQ methods respond to slightly more complicated artificial datasets.

4.2.1. *Multiple Classes.*

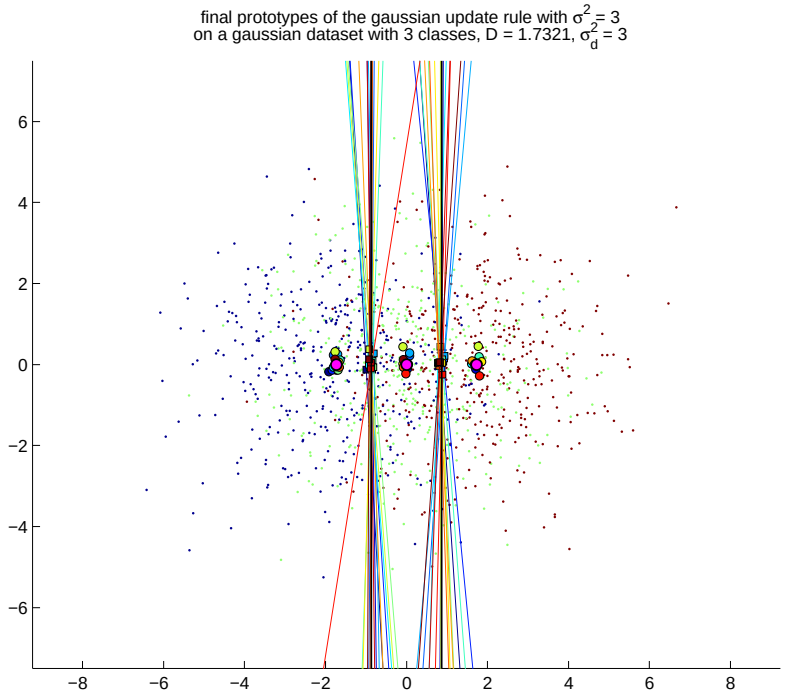

FIGURE 33. The final prototypes and decision boundaries of the Gaussian method for 15 i.i.d. runs (each in a different color) on datasets with 3 classes. The small points are one such dataset, and serve to illustrate the scale. The class means of the dataset are shown in magenta, with a thickened border. Here, $\tilde{D} = \sqrt{3}$ is the distance from the class means to the origin, with the exception of the class that is centered at the origin.

For datasets with two classes, we saw that the final prototypes of the Student's t method deviate from the class means for $D = \{\sqrt{3}, 2\sqrt{3}\}$. See for example Fig. 22 and Fig. 24, where the prototypes are pushed out to the left and right. The cause of this is deviation from the $D = \boldsymbol{\omega}_x$ line, shown in Fig. 19. The green line ($v = 3$) at $D = \sqrt{3}$ has a large deviation from the $D = \boldsymbol{\omega}_x$ line, which decreases with increasing D.

To find out how these deviations manifest themselves when there are more than two classes, we created plots (Figs. 33 to 38) that show the final prototypes of 15 runs on datasets with three classes on a line ($\sigma_d^2 = 3.0$), for distances $\tilde{D} = \{\sqrt{3}, 2\sqrt{3}, 3\sqrt{3}\}$. Here, $\tilde{D}$ is the distance between adjacent classes in three-class datasets. We chose $v = 3$ for the Student's t method, and $\sigma^2 = \sigma_d^2 = 3.0$ for the Gaussian method.

The Gaussian method behaves exactly as one would expect, so we focus on the results of the Student's t method. For $\tilde{D} = \sqrt{3}$ we see interesting behavior for the Student's t method (Fig. 34). Apparently, $\tilde{D} = \sqrt{3} \approx \tilde{D}_{\text{jump}}$. We can see that some

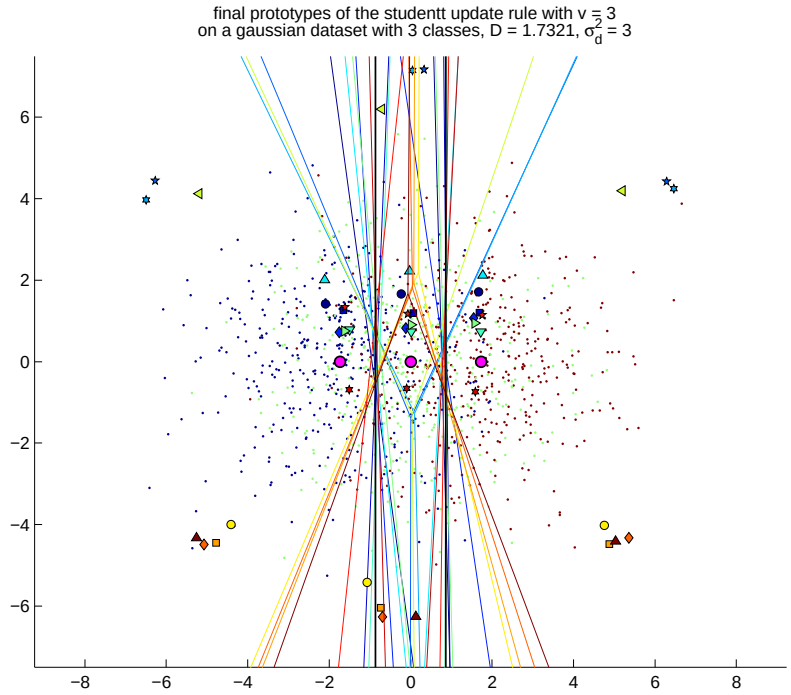

FIGURE 34. Similar to Fig. 33, for the Student's t method and $D = \sqrt{3}$. Because the final prototypes are so spread out on this image, each run has a unique combination of color and shape.

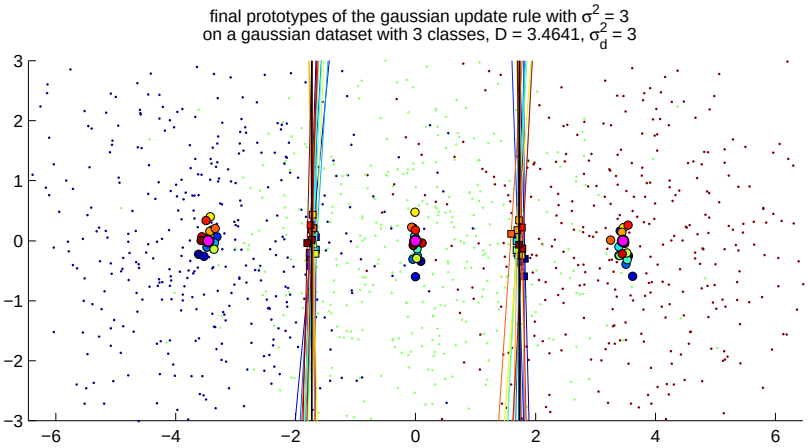

FIGURE 35. Similar to Fig. 33, for the Gaussian method and $D = 2\sqrt{3}$.

of the runs converged to pre-jump prototype positions (hardly any horizontal deviation), while others converged to post-jump prototype positions (huge deviations). We did not see any pre-jump prototypes for $D = \sqrt{3}$ on the two-class datasets, so apparently $\sqrt{3}$ is closer to $\tilde{D}_{\text{jump}}$ than to D_{jump} (or $\tilde{D}$ is perceived smaller than D because of the added class).

For the runs of Fig. 34 that converged to post-jump prototype positions, all prototypes converged either strictly above or strictly below the class means. This indicates a vertical component in the deviations, that appears to be positive or

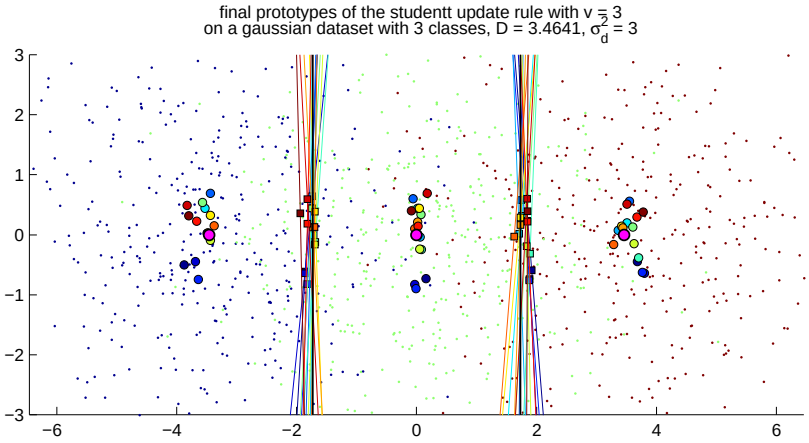

FIGURE 36. Similar to Fig. 33, for the Student's t method and $D = 2\sqrt{3}$.

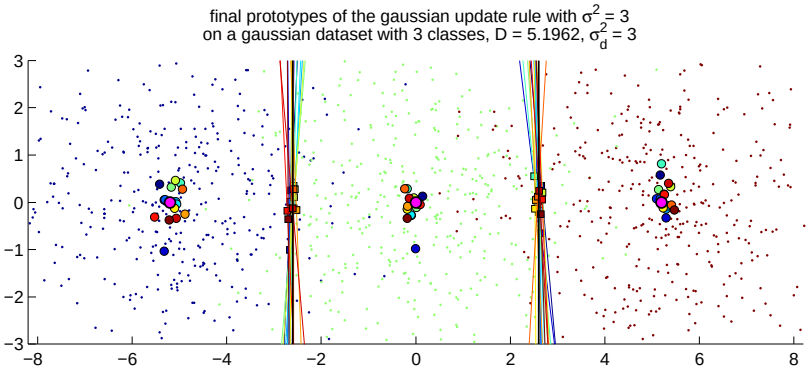

FIGURE 37. Similar to Fig. 33, for the Gaussian method and $D = 3\sqrt{3}$.

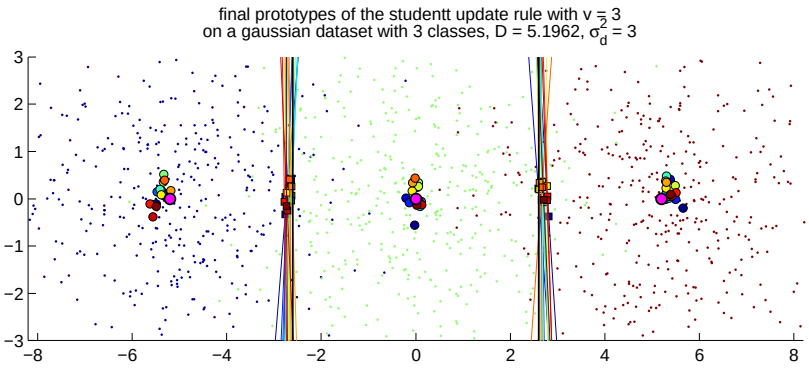

FIGURE 38. Similar to Fig. 33, for the Student's t method and $D = 3\sqrt{3}$.

negative with equal probability. The left and right prototypes have an additional horizontal deviation. The middle prototypes have no significant horizontal deviation, but a larger vertical deviation. The pre-jump prototypes have primarily vertical deviations.

We conclude that for runs converging to post-jump prototypes (which includes all runs of Figs. 36 and 38), the left and right prototypes have horizontal deviations

that decrease with increasing $\tilde{D}$, while the middle prototype has no significant horizontal deviation (it appears to be canceled out). For $\tilde{D}$ nearing $\tilde{D}_{\text{jump}}$, we see additional significant vertical deviations for all prototypes.

Like we saw for the two-class datasets, and for the same reasons, the accuracy of the Student's t method increases (and deviations decrease) as $\tilde{D}$ increases, while the accuracy of the Gaussian method decreases as $\tilde{D}$ increases.

Besides the results near $\tilde{D}_{\text{jump}}$, there is another significant difference between the RSLVQ methods. Due to the deviations of the prototypes from the class means, the decision boundaries of the Student's t method at $\tilde{D} = \{2\sqrt{3}, 3\sqrt{3}\}$ are not exactly halfway in between the class means, but slightly shifted away from the middle. Note that although we saw the same type of deviations for two-class datasets, they do not affect the decision boundaries there.

4.2.2. *Different Amounts Of Points Per Class.*
We know that prototypes stabilize because the forces that act on them cancel each other out. We also know from Figs. 7 and 8 that with respect to the left prototype, the mean force of the black points is positive (to the right), and the mean force of the white points is negative (to the left). Thus, if we were to take a dataset with significantly less white points than black points, we would expect to see the left prototype shift towards the right. Analogously, we would expect to see the right prototype shifted to the right as well. In other words: we expect the decision boundary to shift toward the weaker class.

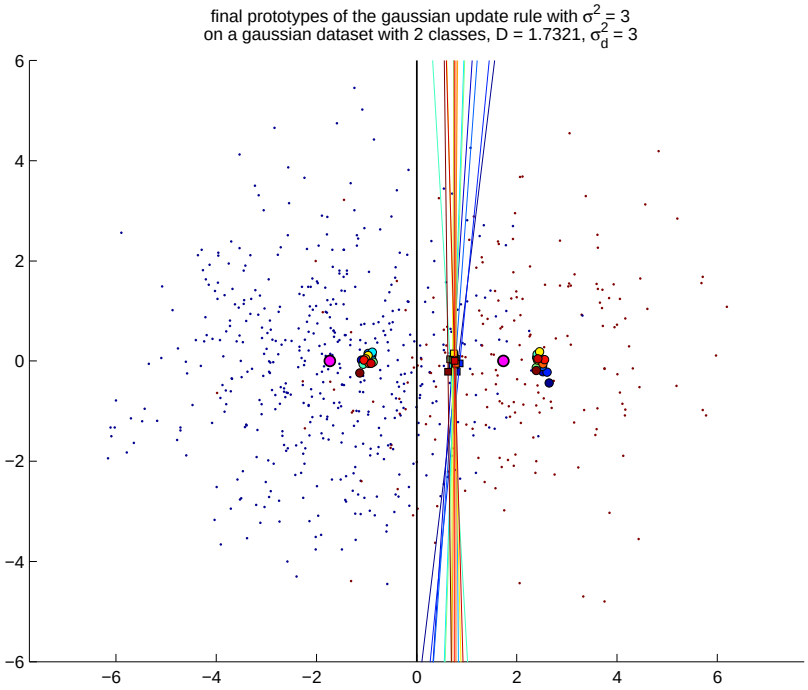

FIGURE 39. The final prototypes of the Gaussian method for 15 i.i.d. runs (each in a different color) on datasets with 2 classes, with a 7:3 ratio for the blue:red points. The squares on the decision boundaries are the means of the two prototypes for each run. The small points are one of the datasets used, and serve to illustrate the scale. The class means of the dataset are shown in magenta, with a thickened border. Here, $D = \sqrt{3}$ is the distance from the class means to the origin.

To test this theory, we have created datasets with a 7:3 ratio for points belonging to the left:right class. In Figs. 39 and 40 we can see that prototypes for both the Gaussian and Student's t method are shifted towards the right, as well as their decision boundaries. This confirms our expectations.

We conclude that both methods respond to changes in the ratio of points per class in roughly the same way: with a shift of prototypes and decision boundary toward the weaker class.

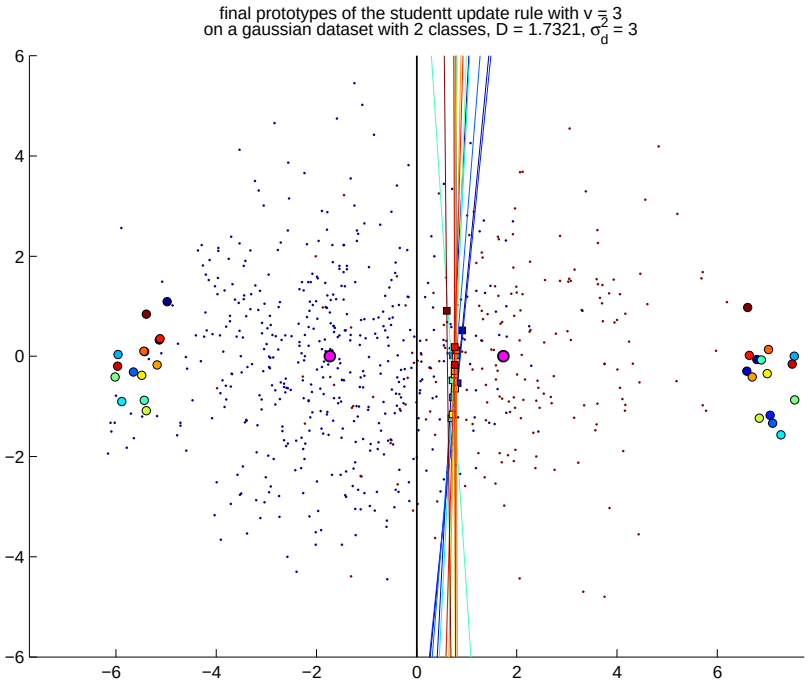

FIGURE 40. The final prototypes of the Student's t method for 15 i.i.d. runs (each in a different color) on datasets with 2 classes, with a 7:3 ratio for the blue:red points. The squares on the decision boundaries are the means of the two prototypes for each run. The small points are one of the datasets used, and serve to illustrate the scale. The class means of the dataset are shown in magenta, with a thickened border. Here, $D = \sqrt{3}$ is the distance from the class means to the origin.

4.2.3. *Different Variances Per Class.*
We will be looking at what happens for the RSLVQ methods when classes have different variances. This is a scenario that can occur even on normalized datasets. While normalization can change class variances, the ratio between class variances is not changed in this process.

We have tested this scenario using a dataset with two classes on a line like before, only with differing class variances. The left class has $\sigma_d^2 = 3.0$ and the right class has $\sigma_d^2 = 1.0$. A selection of the results is shown in Figs. 41, 42, 43 and 44.

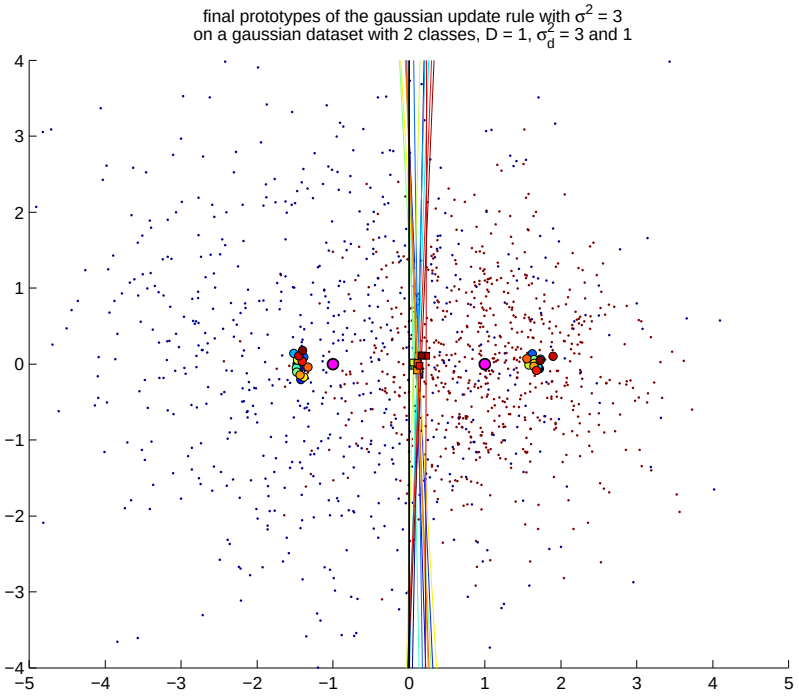

FIGURE 41. The final prototypes of the Gaussian method for 15 i.i.d. runs (each in a different color) on datasets with 2 classes, with the blue and red class having $\sigma_d^2 = 3.0$ and $\sigma_d^2 = 1.0$, respectively. The squares on the decision boundaries are the means of the two prototypes for each run. The small points are one of the datasets used, and serve to illustrate the scale. The class means of the dataset are shown in magenta, with a thickened border. Here, $D = 1$ is the distance from the class means to the origin.

The two classes having different variances has two effects which we discuss below.

The first is that for the Gaussian method, the prototypes do not end up at the class means anymore. This is because the classes have different variances and we know that for datasets with equal variance, $\boldsymbol{\omega}_x = \frac{\sigma^2 D}{\sigma_d^2}$ holds. Since we can only choose one variance to use in the update rule, there is no easy way to make both prototypes overlap with the class means. In Figs. 41 and 43, we chose $\sigma^2 = 3.0$. Since this is the highest of both variances, we see that for both prototypes $\boldsymbol{\omega}_x > D$.

The second effect is found at the decision boundary for both the Student's t and the Gaussian method. Although we have not included all figures to show this properly, we note that for all runs on datasets with $D = 1$, 2 or 3, the decision boundaries are slightly to the right of the $x = 0$ line. An exception here is $D = 1$ for the Student's t method (shown in Fig. 42).

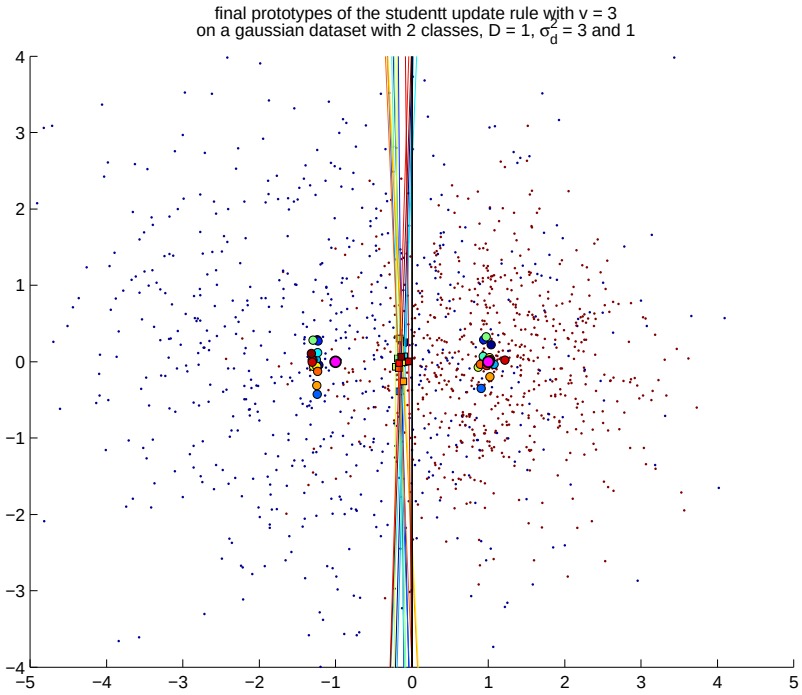

FIGURE 42. The final prototypes of the Student's t method for 15 i.i.d. runs (each in a different color) on datasets with 2 classes, with the blue and red class having $\sigma_d^2 = 3.0$ and $\sigma_d^2 = 1.0$, respectively. The squares on the decision boundaries are the means of the two prototypes for each run. The small points are one of the datasets used, and serve to illustrate the scale. The class means of the dataset are shown in magenta, with a thickened border. Here, $D = 1$ is the distance from the class means to the origin.

To explain why the decision boundary is shifted to the right, recall how for the Gaussian update rule, the majority of the points has no influence on the decision boundary; the decision boundary is primarily determined by outliers, points on the other side of $x = 0$. The class with the highest variance also has the most outliers, and so at $x = 0$ it will have a stronger total force than the class with the smaller variance. Ergo, the decision boundary shifts toward the class with the smallest variance. The same holds for the Student's t method to a lesser degree. Note that this effect is caused by the distribution of the dataset. No matter what meta-parameters we use in the update rule, the class with the higher variance will always have more and more extreme outliers (on average).

In the following, let x_{db} be the position of the decision boundary. The biggest difference between both methods is found at $D = 1$, where $x_{\text{db}} > 0$ for the Gaussian method and $x_{\text{db}} < 0$ for the Student's t method (although all decision boundaries are close to $x = 0$). At $D = 2$ and higher, $x_{\text{db}} > 0$ for both methods, and increases with increasing D. Furthermore, x_{db} increases for the Student's t method with increasing v. Changing the variance for the Gaussian method however, does not affect x_{db}. The negative x_{db} at $D = 1$ is probably related to the instabilities around D_{jump} for the Student's t method.

We conclude that although the methods respond in a similar way when confronted with two overlapping classes that do not have the same class variance, their decision boundaries may converge to slightly different offsets.

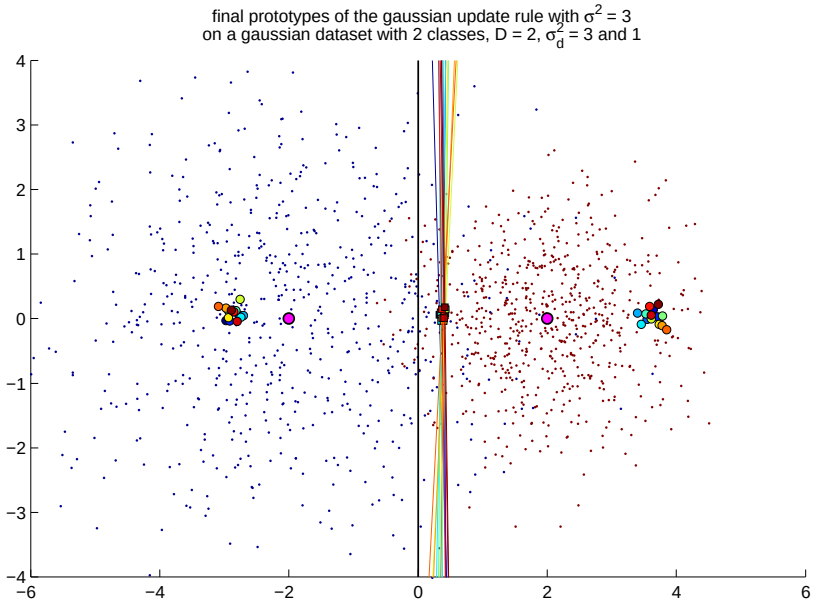

FIGURE 43. The final prototypes of the Gaussian method for 15 i.i.d. runs (each in a different color) on datasets with 2 classes, with the blue and red class having $\sigma_d^2 = 3.0$ and $\sigma_d^2 = 1.0$, respectively. The squares on the decision boundaries are the means of the two prototypes for each run. The small points are one of the datasets used, and serve to illustrate the scale. The class means of the dataset are shown in magenta, with a thickened border. Here, $D = 2$.

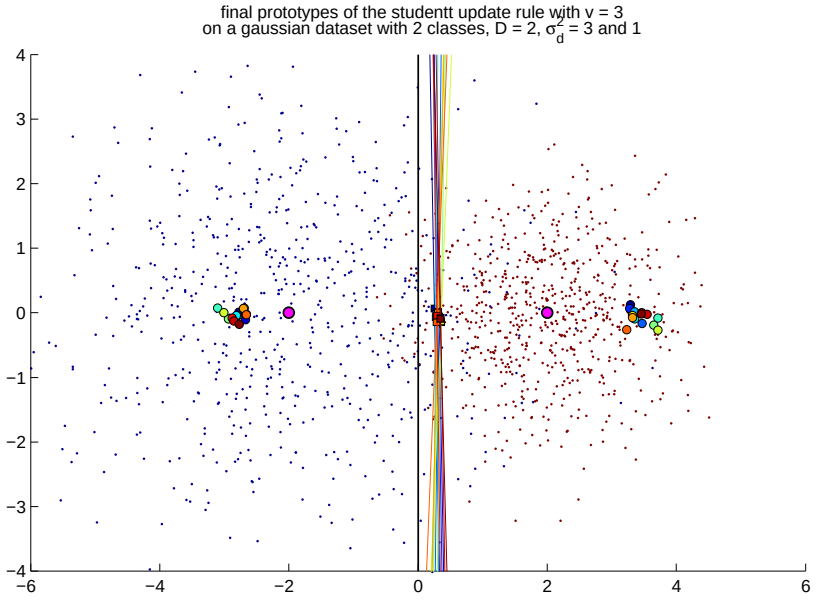

FIGURE 44. The final prototypes of the Student's t method for 15 i.i.d. runs (each in a different color) on datasets with 2 classes, with the blue and red class having $\sigma_d^2 = 3.0$ and $\sigma_d^2 = 1.0$, respectively. The squares on the decision boundaries are the means of the two prototypes for each run. The small points are one of the datasets used, and serve to illustrate the scale. The class means of the dataset are shown in magenta, with a thickened border. Here, $D = 2$.

4.3. **Real-Life Dataset.**
Thus far, we have assumed that classes in our datasets follow a Gaussian distribution. However, for many real-life datasets, this will not be the case. To see which of our conclusions hold up in practice, we compare the performance of the RSLVQ methods on a real-life dataset.

4.3.1. *Dataset And Parameters.*
We use the tiling microarray dataset based on genome region [4413428 : 4540601] in chromosome 3 of *C. elegans*, as described in [Bieh07]. The samples of the dataset are genomic positions, classified as either exonic or intronic. Labeling is done according to the current genome annotation, and for the sake of simplicity we will assume that this labeling is always correct. The dataset contains 24 features, such as median PM (perfect match) and M (mismatch) transcription intensities, correlations of neighboring genome positions, melting temperatures, etc. For more information on the dataset, we refer to [Bieh07].

The dataset consists of 2 classes in 24 dimensions and has a total of 4120 samples, 2587 of which are labeled as exonic, and the other 1533 as intronic. The dataset is first normalized (per dimension). We use one prototype per class. For our tests we use two-thirds of the dataset for training, and the rest for testing. The results presented are averaged over 15 runs, each run with (hopefully distinct) randomly selected training- and testsets.

After running initial tests up to 500 epochs, we found that after 200 epochs the accuracy does not change significantly anymore. By starting with higher learning rates and using slower (linear) decay of α, we can achieve the same prototypes after 200 epochs as by using more epochs. As for the learning parameters, we choose $\alpha_{\text{start}} = 0.005$ (see equation (13) for the use of α_{start}) for the Gaussian configuration with $\sigma^2 = 1.0$, $\alpha_{\text{start}} = 0.0025$ for the Gaussian configuration with $\sigma^2 = 0.5$, and $\alpha_{\text{start}} = 0.01$ for all Student's t configurations. We let α decay linearly, as described in section 4.1.1.

We have tested the Gaussian method using $\sigma^2 = \{0.5, 1.0\}$. For the Student's t method, we tested with $v = \{1, 10, 30, 50, 100\}$.

4.3.2. *Results.*
Figure 45 shows the mean testset accuracy during training, averaged over 15 runs. Table 5 shows the mean misclassification rates on the testset after 200 epochs, averaged over 15 runs. The rows in this table are sorted from best to worst performance. Figure 46 shows the spread in testset accuracy of individual runs for each configuration.

We can see from Table 5 that all configurations perform better on average than simply using the class conditional means.

Looking at Fig. 46, we see that both Gaussian configurations give roughly the same performance. This tells us that changing the variance used in the update rule does not significantly impact performance. Changing the variance changes the distance of the prototypes to the decision boundary; it does not significantly change the decision boundary itself.

The worst performing configurations in our tests are clearly the Student's t configurations with low v ($v = \{1, 10\}$). We will explain this in the discussion (section 5.1).

mean rate of misclassification	method	meta-parameter value
10.7502% (0.50370%)	Student's t	$v = 30$
10.7745% (0.60152%)	Student's t	$v = 50$
10.8279% (0.61842%)	Gaussian	$\sigma^2 = 1.0$
10.8473% (0.60529%)	Gaussian	$\sigma^2 = 0.5$
10.8570% (0.60215%)	Student's t	$v = 100$
11.0706% (0.55126%)	Student's t	$v = 10$
13.1974% (0.51589%)	Student's t	$v = 1$
13.5922% (0.00000%)	CCM	

TABLE 5. The mean rate of misclassification on the tiling microarray data, sorted from best to worst performance. The second value in the first column denotes the standard deviation. Each row (except the last) is calculated as the accuracy on the testset after 200 epochs, averaged over 15 runs. CCM is the performance we would get if we use the class conditional means as prototypes.

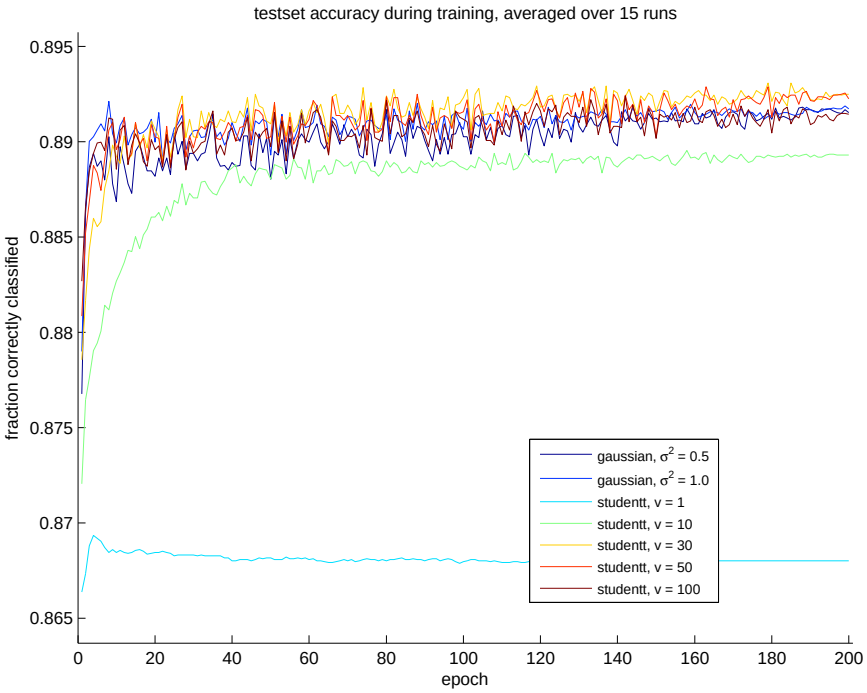

FIGURE 45. The accuracies on the testset during training, for the tiling microarray data. Each line represents the average taken over 15 runs.

The best performing configuration in our tests is the Student's t method with $v = 30$. In Fig. 46 we can see that this is mainly because its worst-case performance is significantly higher than that of Gaussian configurations. The Student's configurations with $v = \{30, 50\}$ also have the highest accuracy for a single run (90.09%), although this value is not significantly higher than the peaks of the Gaussian configurations.

If we choose v higher than 30 ($v = \{50, 100\}$), we can see that performance decreases again. The Student's t configuration with $v = 100$ gives roughly the same

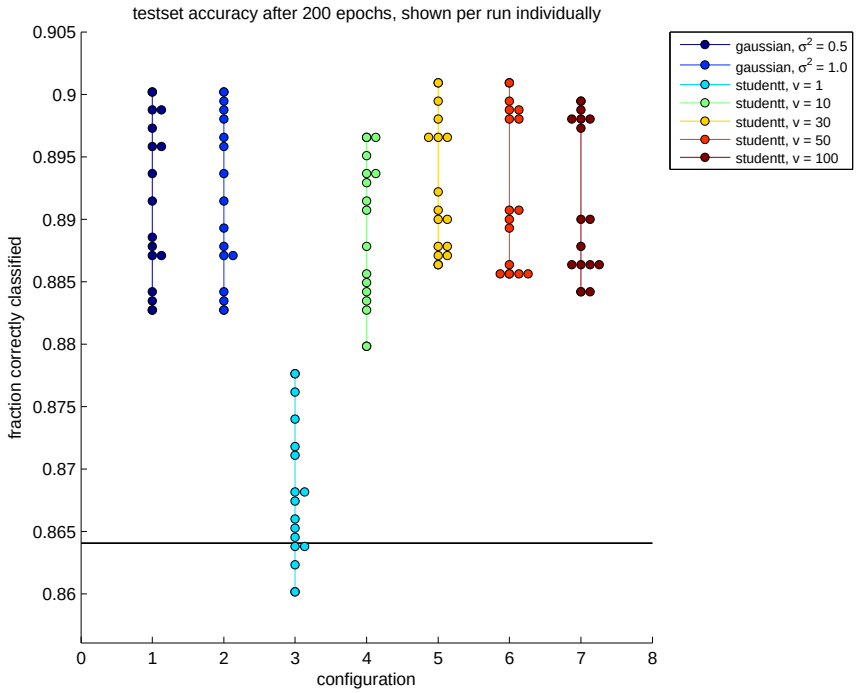

FIGURE 46. Pseudo-boxplot of the accuracies on the testset after 200 epochs, for the tiling microarray data. The markers on the lines are the accuracies of the individual runs. The black horizontal line is the accuracy when using the class conditional means as prototypes.

performance as the Gaussian configurations. This is no coincidence: we know that for $v \to \infty$, the Student's t distribution becomes the Gaussian distribution. This is why the Student's t method at high v drops down to Gaussian-like performance.

4.3.3. *Interpretation.*
We conclude that changing v for the Student's t method can greatly impact performance, unlike changing the variance for the Gaussian method. On this particular multidimensional normalized dataset, we have seen that there is an optimal v somewhere around 30, for which the Student's t method slightly outperforms the Gaussian configurations. Choosing v too low or too high decreases performance.

While the Gaussian method may very well be optimal in some ways, it has a larger spread of accuracies between the different runs than some of the Student's t configurations. We think that this is because the method relies mostly on outliers to determine the decision boundary, and outliers can change a lot between different runs.

The Student's t method on the other hand, because of its heavy-tailed property, takes more points into account, and is thus more robust against different subsets of the same dataset. This gives better worst-case performance and thus better average performance and less spread in accuracy values. Note for instance how in Table 5 the Student's t configuration with $v = 30$ has the lowest mean rate of misclassification as well as the lowest standard deviation for this number.

5. Discussion

In this section we present general observations and conclusions drawn from our results.

5.1. **Dataset Normalization.**

In this section we discuss the effects of dataset normalization on the RSLVQ methods. The type of normalization we consider here is applied per dimension, scaling all values to zero mean and unit variance. Real-life datasets often have multiple dimensions, with different variances in each dimension. Dataset normalization is performed to make the variances of the different dimensions equal, so that summed up distances in the Euclidean distance term (used in the update rules) have the same weight.

However, this is not entirely correct. Since the normalization is applied per dimension and not per class individually, there is no saying as to what the actual class variances will be. For example, if in some dimension, class A has three times the variance of class B prior to normalization, then the same will hold after normalization.

A second motivation for normalization is the desire to bring datasets to the same scale, so that we do not have to figure out specific meta-parameters for each new dataset. To see if this is indeed the case, we look at the meta-parameters that depend on the scale of the dataset, or to be more precise: on the class variance σ_d^2 and the class means D.

For the Student's t distribution, we have the meta-parameter v, the degrees of freedom. According to the literature [Weis09], a Student's t distribution has a defined variance $\tilde{\sigma}^2 = \frac{v}{v-2}$, for $v \geq 3$. So high values of v give a variance of about 1. However, we have not been able to derive a direct relation between v and the parameters of the dataset. Therefore, we may not simply assume that the variance of the distribution used in the Student's t update rule should match the variance of the dataset.

Rather, recall from Fig. 19 that the value of $\boldsymbol{\omega}_x$ 'jumps' at a certain D_{jump}. For values of D close to D_{jump}, convergence is slow, inaccurate and unpredictable (the jump is a significant deviation from the $D = \boldsymbol{\omega}_x$ line). The value of D_{jump} depends on σ_d^2 and v. If we assume $\sigma_d^2 = 1$ for normalized datasets, then we have to choose a v such that no class c in any dimension i has a $D_{c,i}$ close to D_{jump}.

We refer to Fig. 47, which is like Fig. 19, but for $\sigma_d^2 = 1.0$; it shows the position of the first final prototype of the Student's t method, for various v. Because of the normalization, we may assume that roughly $0 \leq D_{c,i} \leq 2$ for the bulk of the $D_{c,i}$. However, Fig. 47 shows that this is exactly where all the lower v curves have their jump. This is why on the normalized real-life dataset (section 4.3.2), we saw the worst performance from Student's t configurations having a low v. So for normalized datasets, the only option would be to choose a high value for v so that we get a high value for D_{jump}, ensuring that all $D_{c,i}$ occur before the jump.

Note that the first segment of the curves in Fig. 47 better approximates the $D = \boldsymbol{\omega}_x$ line for increasing v. Less deviation from this line means more predictable behavior, i.e. prototypes that move where we would expect them to move. We would need a $v \gtrapprox 1000$ to not notice any deviation at all. However at this v, the Student's t update rule acts exactly like the Gaussian update rule, so we would gain nothing by using the Student's t method.

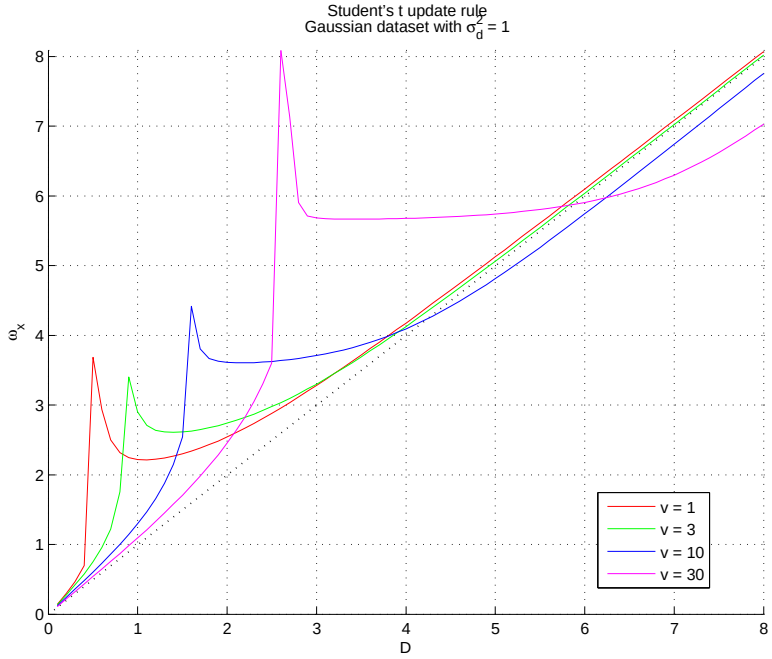

FIGURE 47. The position of the first stable prototype ($\boldsymbol{\omega}_x$) as a function of D (half the distance between the class means in the dataset), for the Student's t method. The variance of the dataset was fixed at $\sigma_d^2 = 1$. We refer to the D where the peaks occur as D_{jump}.

Scaling the datasets to a higher variance would push the jumps back, but also increases the D of the datasets, so that does not solve anything either. Thus, for normalized datasets we can either choose a high v to avoid the jump, ending up with Gaussian-like behavior, or choose a low v, and deal with slow convergence, inaccuracy and unpredictability.

As we have seen in section 4.3.2, there is a third option however. From the performance of the $v = 30$ configuration on the real-life dataset, we can conclude that with small deviations from the $D = \boldsymbol{\omega}_x$ line, we can still get good results. In between the low and high values for v, there are values for which the negative effects of D_{jump} are minimized, yet the heavy-tailed properties of the Student's t distribution are retained. Student's t configurations with such v outperform the Gaussian method. Recall that we also concluded earlier that prototype deviations do not necessarily affect the accuracy of decision boundaries (compare e.g. Fig. 21 with Fig. 22).

As for the Gaussian method, we can simply use $\sigma^2 = 1.0$, which is arguably the best choice if all we know about the dataset is that it is normalized. Furthermore, note that by using $\sigma^2 = 1.0$ on a normalized dataset, we practically never have problems with too little overlap leading to inaccurate and slow convergence. However, as we have seen in section 4.3.2, performance on different datasets has a relatively large variance.

We conclude that even on normalized datasets, the heavy-tailed property of the Student's t distribution still proved useful, as it gives a more robust classifier.

5.2. **Conclusions.**
In this paper we introduced an RSLVQ variant that uses Student's t distributions. We found that this RSLVQ variant can be a viable alternative to conventional Gaussian-based RSLVQ on normalized datasets as well as on datasets that were not normalized. On a real-life dataset it outperformed Gaussian-based RSLVQ, showing lower variance in testset accuracies and better worst-case performance.

The main difference between Gaussian-based and Student's t-based RSLVQ is in the points that are taken into account when calculating prototype updates. For Gaussian-based RSLVQ, these are points near the decision boundary, while for Student's t-based RSLVQ, only points relatively close to their class means are used. Thus, the distinction becomes apparent when the class means are far away from the decision boundary – even in normalized datasets.

So, when separation between the classes increases, the number of points that influences the decision boundary for Gaussian-based RSLVQ decreases, and as a result, variance in convergence increases (lower accuracy). This does not happen for Student's t-based RSLVQ. This effect is not limited to datasets with clearly separated classes, but also plays a role in real-life datasets.

A related disadvantage of Gaussian-based RSLVQ is that its update strengths can be so small that some runs do not converge. These convergence problems can often be resolved by either using a high variance in the update rule or by normalizing the dataset and using $\sigma^2 = 1.0$. In both cases however, we notice a loss of accuracy compared to Student's t-based RSLVQ.

A disadvantage of Student's t-based RSLVQ is the unpredictability in where the prototypes end up. Moreover, there is a certain range for D where the prototype positions jump, causing large variance and unpredictability in results. In most cases, problems can be avoided by choosing a low v for datasets that were not normalized (i.e. $v = 1$), and a higher v for normalized datasets (i.e. $v = 30$). For v chosen very high (i.e. $v = 1000$), the behavior of Student's t-based RSLVQ approaches that of Gaussian-based RSLVQ.

As illustrated by the example above, we saw that unlike the variance σ^2 for Gaussian-based RSLVQ, parameter v heavily influences Student's t-based RSLVQ, and can be fine-tuned for a dataset to achieve optimal performance.

We remark that the measured differences on the real-life dataset were small, and although we can explain them from the theory, the extent of their influence is dataset-specific. Further testing on other real-life datasets is needed to determine if Student's t-based RSLVQ is a consistent, significant improvement over Gaussian-based RSLVQ.

6. Future Work

6.1. **Priors And Dimension Specific Variances.**

As a potential improvement to our current implementation of RSLVQ, we have looked at the role of the prototype priors in the update rule. As we can see in equation (4), the formulas for $P_y(j|\boldsymbol{\xi})$ and $P(j|\boldsymbol{\xi})$ originally included terms $P(j)$, denoting the prior probability of a prototype. In later formulas these are canceled out under the assumption that all prototypes have equal prior probability.

In most cases this assumption makes sense, but in the case of an N-class problem with one prototype per class, one could opt whether it would not be better to use the number of points in the trainingset as priors. For example if we have a two-class problem with a trainingset of 70% class 0 and 30% class 1 points, we could use those as priors in the update rules. This is intuitive, because when we have one prototype per class, each prototype basically represents its class. And since the dataset itself is modeled as a mixture of probability density functions centered around those prototypes, it would better represent the actual dataset if those priors were taken into account.

However, after implementing and testing this idea, we found that it gave slightly worse results. The net effect is that the prototypes no longer converged to the class means, but were offset because of the weight factors. This is expected behavior however, and we think that the results are skewed because of another inaccuracy.

Our implementation of RSLVQ uses the same variance for all classes/prototypes. We think that introducing weights when we do not have class/prototype specific variances leads to scaling errors that end up giving slightly worse results.

The solution would be to use class/prototype specific variances. Since the datasets are normalized per dimension, simply using one variance per class would not be a meaningful improvement; we need variances per dimension for each class. Since the implementation uses a one-dimensional pdf based on a Euclidean distance measure taken over all dimensions, implementing dimension specific variances is no trivial task.

We came up with two different ways to implement dimension specific variances per class.

- Modify the Euclidean distance calculation to use individual weights for the distances of different dimensions. Then use that modified Euclidean distance as input to a one-dimensional pdf like before.
- Use one one-dimensional pdf per dimension. Choose the variances of the pdfs to match the class variances of the sample being used. Then use the sum of the pdfs as measure of 'probability'. This option has the advantage that there is a clear relation between the variances in the dataset and those used in the update rule.

We have not implemented these approaches however, due to limited time and the fact that the whole concept applies to the Gaussian method only.

6.2. **Multiple Prototypes Per Class.**

In all tests we assumed a single prototype per class. More complicated datasets often require the use of multiple prototypes per class. It would be interesting to see how the Student's t method performs in such scenarios.

6.3. **Relevance Learning For The Student's t Method.**
An interesting experiment would be to use global matrix RSLVQ (an adaptation of RSLVQ that applies relevance learning) as described in [Schn09] for the Student's t update rule.

We note here that we have tested global matrix RSLVQ for the Gaussian update rule on the real-life dataset of section 4.3, but we did not see a significant improvement in testset accuracy.

The use of a global relevance matrix simplifies the classifier and effectively pulls the prototypes closer to the class means. When projecting the dataset onto the eigenvectors of Λ, it shows increased separation between the points of different classes.

Since we know that the Student's t method performs better (and the Gaussian method performs worse) when datasets become more separated, global matrix RSLVQ might give the Student's t method an additional boost in testset accuracy that we did not see for the Gaussian method.

Acknowledgments

I would like to thank M. Biehl and A.W. Witoelar for feedback, suggestions and for proofreading this paper. I would also like to thank P. Schneider for feedback and suggestions, and for providing a Matlab implementation of the original Gaussian-based RSLVQ algorithm along with several real-life datasets.

References

[Bieh07] Biehl, M, R Breitling and Y Li. 2007. Analysis of Tiling Microarray Data by Learning Vector Quantization and Relevance Learning. *Lecture Notes in Computer Science* 4881:880–889.

[Duda01] Duda, RO, PE Hart and DG Stork. 2001. Pattern Classification. Publisher *Wiley, New York*.

[Koho86] Kohonen, T. 1986. Learning Vector Quantization for Pattern Recognition. *Report TKK-F-A601*, Helsinki University of Technology.

[Koho90] Kohonen, T. 1990. Improved versions of Learning Vector Quantization. *1990 IJCNN International Joint Conference on Neural Networks* 545–550.

[Schn09] Schneider, P, M Biehl and B Hammer. 2009. Distance learning in discriminative vector quantization. *Neural Computation* in press.

[Seo03] Seo, S and K Obermayer. 2003. Soft Learning Vector Quantization. *Neural Computation* 15(7):1589–1604.

[Weis09] Weisstein, EW. 2009. Student's t-distribution. *MathWorld*. http://mathworld.wolfram.com/Studentst-Distribution.html

Appendix A. Proof that updates of the Gaussian update rule in the x-dimension do not depend on y

Consider a dataset with two classes, A and B. Both classes have their class mean on the x-axis. We denote the x-coordinate of the mean of class A as $\mu_{A:x}$, and for B this is $\mu_{B:x}$. Let P define the points $[x_P, y_P]$; we assume that x_P is fixed so that P defines points on a vertical line.

The Euclidean distance of the mean of class A to points P is $\sqrt{(\mu_{A:x} - x_P)^2 + y_P^2}$, and for B it is $\sqrt{(\mu_{B:x} - x_P)^2 + y_P^2}$. As in the RSLVQ implementation, the Euclidean distances are weighed by a Gaussian pdf with zero mean and variance σ^2.

Being the only part in the x-dimension of the update rule (equation (6)) that involves y, we want to prove that the ratio of pdfs $r = P(j|\boldsymbol{\xi})$ does not change for different y_P, as long as the other variables are fixed. This ratio of pdfs can be written as:

$$r = \frac{\text{normpdf}(\sqrt{(\mu_{A:x} - x_P)^2 + y_P^2}, 0, \sigma)}{\text{normpdf}(\sqrt{(\mu_{A:x} - x_P)^2 + y_P^2}, 0, \sigma) + \text{normpdf}(\sqrt{(\mu_{B:x} - x_P)^2 + y_P^2}, 0, \sigma)}$$

Writing out the formula for `normpdf` gives:

$$r = \frac{\frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-((\mu_{A:x} - x_P)^2 + y_P^2)}{2\sigma^2}}{\frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-((\mu_{A:x} - x_P)^2 + y_P^2)}{2\sigma^2} + \frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-((\mu_{B:x} - x_P)^2 + y_P^2)}{2\sigma^2}}$$

Terms cancel each other out, and rewriting `exp` gives:

$$r = \frac{e^{\frac{-((\mu_{A:x} - x_P)^2 + y_P^2)}{2\sigma^2}}}{e^{\frac{-((\mu_{A:x} - x_P)^2 + y_P^2)}{2\sigma^2}} + e^{\frac{-((\mu_{B:x} - x_P)^2 + y_P^2)}{2\sigma^2}}}$$

Now realize that we can split up the powers of e, and rewrite as:

$$r = \frac{e^{\frac{-(\mu_{A:x} - x_P)^2}{2\sigma^2}} \cdot e^{\frac{-y_P^2}{2\sigma^2}}}{e^{\frac{-(\mu_{A:x} - x_P)^2}{2\sigma^2}} \cdot e^{\frac{-y_P^2}{2\sigma^2}} + e^{\frac{-(\mu_{B:x} - x_P)^2}{2\sigma^2}} \cdot e^{\frac{-y_P^2}{2\sigma^2}}}$$

Again we cancel out common terms, leaving:

$$r = \frac{e^{\frac{-(\mu_{A:x} - x_P)^2}{2\sigma^2}}}{e^{\frac{-(\mu_{A:x} - x_P)^2}{2\sigma^2}} + e^{\frac{-(\mu_{B:x} - x_P)^2}{2\sigma^2}}}$$

Since y_P no longer occurs in the formula for r, we conclude that this ratio does not depend on y_P. Thus, we have proven what we set out to prove, namely that the ratio of pdfs does not change for different values of y_P.

With respect to the Gaussian update rule in the x-dimension, the ratio of pdfs was the only place where y was used. Since y could be canceled out here, we have proven that updates of the Gaussian update rule in the x-dimension do not depend on y at all.

Interpreting the formulas, we conclude that the y_P^2 terms could be canceled out because both classes have the same mean value in this dimension. When the means of the two classes in a certain dimension are equal, then the distance of any point to the means in that dimension is equal as well. So any change in that distance affects both classes in the same way, and can thus be canceled out in the ratio of pdfs. As a side note, we remark that since the y-dimension is canceled out, the ratio is actually a ratio of pdfs of the one-dimensional Euclidean distance measure (which is simply the absolute distance) in the x-direction.