
Speaker Recognition in non-linear signal processing and pattern recognition.

RUNE A. HEICK, RASMUS S. REIMER & NICOLAI GLUD

Aarhus University, Department of Engineering

Abstract

The Content of this paper seeks to present the knowledge gained throughout the non-linear signal processing and pattern recognition course from Aarhus University, department of engineering. The paper is split into multiple sections explaining the data used in the paper, the methods used to treat the data and the methods used for categorising the data.

I. INTRODUCTION

The idea behind the project is to recognise the speaker using the methods and categorisers learned in the course pattern recognition and machine learning (TINONS). The voices of all authors was recorded and imported to matlab. The features from the data was extracted in matlab using the Mel-frequency cepstral coefficient(Hereafter MFCC) method from the voicebox toolbox. The MFCC's are used as features for the classifiers that are tested in this paper.

II. DATA GATHERING

The data used in this project was gathered by recording three different persons reading the same article from the website "www.tv2.dk". The voices was recorded using the software Audacity¹ and the Lame mp3 codex². The data is then imported into matlab using the function `[data, Fs] = audioread(pathToFile)`. The data is then normalised by removing the mean of the data, and whitening the data. The files are in stereo and both channels are used by appending one channel to the other so to have one long array of data.

III. FEATURE EXTRACTION

Introtext to MFCC
Math
How we use it
Intermediate result

IV. FEATURES

Size and number of features and stuff.

¹<http://sourceforge.net/projects/audacity/>

²<http://lame.sourceforge.net/>

V. DIMENSIONALITY REDUCTION

e.g. finding projection vectors, choosing number of components, applications.

motivation: data compression speed up learning algorithm

I. PCA

Principal component analysis is used for reducing the number of dimensions of a feature space. It works by projecting the data in the feature space, down to a fewer dimensional feature space by minimizing the squared projection error. The reduced feature space does not necessarily share the same features, but new features are found which best retains the variance in the data.

PCA should mainly be used for compressing the data to save memory or reducing running time of learning algorithm. By reducing the amount of features most machine learning algorithms runs faster. PCA can also be used to prevent overfitting, but its usually better to use regularization.

Figure 1 shows a 3 dimensional feature space where all the data, within a small margin, lies in a 2 dimensional plane. PCA is used to find two vectors $u^{(1)}$ and $u^{(2)}$ which spans this 2D plane.

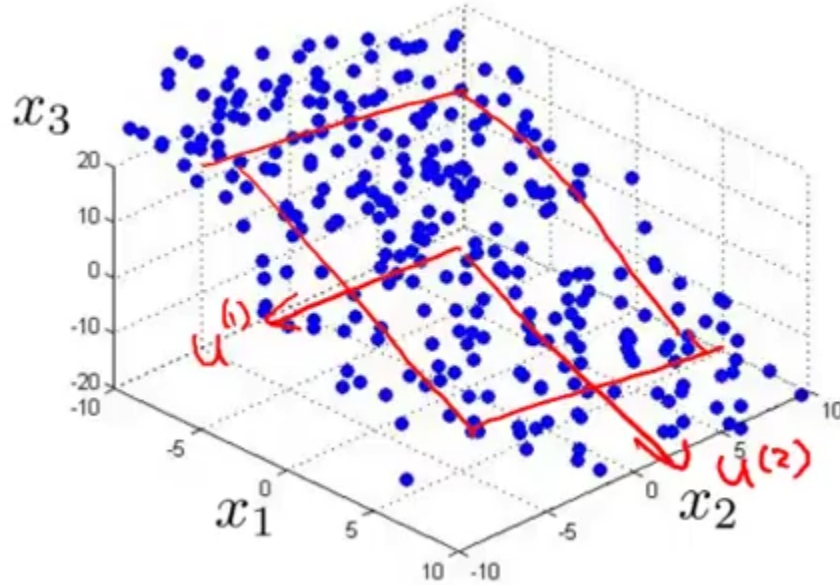


Figure 1: 3D to 2D pca illustration

Preprocessing of the data should be done before doing PCA.

Given the training set:

$$x = [x_1 \quad x_2 \quad \dots \quad x_m] \quad (1)$$

Ensure that every feature has zero mean by doing mean normalization:

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \quad (2)$$

$$x_j = x_j - \mu_j \quad (3)$$

Feature scaling can also be done if the features have very different value ranges. TODO

After preprocessing the data, we can do PCA on it.

We start by computing the covariance matrix Σ :

$$\Sigma = \frac{1}{m} \sum_{i=1}^n x^{(i)} x^{(i)T} \quad (4)$$

The covariance matrix describes how the different features relates. When doing feature reduction we want to remove features which has high correlation with other features. An example could be a feature which describes a length in cm and another feature describing the same length in inches. These features will have very high correlation and one of them can be removed from the feature space without losing much information.

Then we compute the eigenvectors of covariance matrix:

$$U = \begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(n)} \end{bmatrix} \in \mathbb{R}^{n \times n} \quad (5)$$

The eigenvectors will lay in the directions of most variance in the data. This is what is shown on Figure 1. The longer the eigenvector, the more variance it describes. Therefore we want to keep the longest eigenvectors and remove the shortest eigenvectors.

The eigenvectors are ordered by length in the matrix U . The longest is the first.

We select the first k eigenvectors to get the reduced set of eigenvectors:

$$U_{reduce} = \begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(k)} \end{bmatrix} \quad (6)$$

We can now calculate the new feature vectors:

$$z = U_{reduce}^T x \quad (7)$$

We have now reduced the feature space to a k dimensional feature space. Say we want to retain at least 95% of the variance in the data. We do this by picking the smallest value of k so that:

$$\frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}} \geq 0.95 \quad (8)$$

The matrix S is found by doing singular value decomposition (SVD). The matrix S has the form:

$$S = \begin{bmatrix} S_{11} & 0 & 0 & 0 & 0 \\ 0 & S_{22} & 0 & 0 & 0 \\ 0 & 0 & S_{33} & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & S_{nn} \end{bmatrix} \quad (9)$$

In our project we use PCA to reduce our feature space from 64 dimensions down to 40 dimensions. We do this to increase the speed of our learning algorithms while still retaining almost all of our data ($\geq 99.99\%$) as seen on Figure 2.

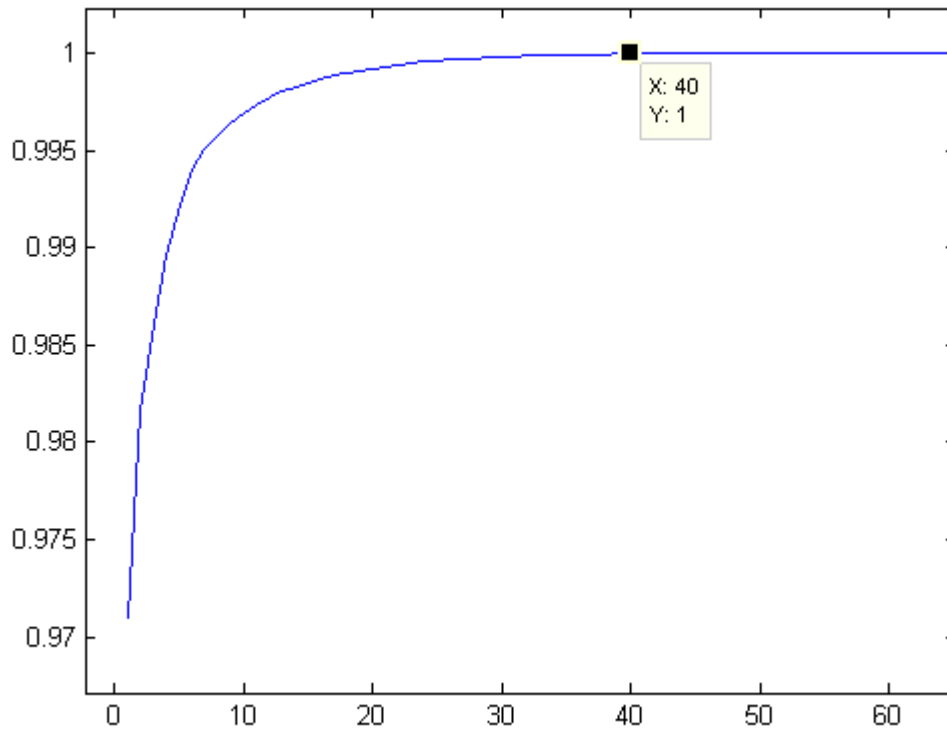


Figure 2: 3D to 2D pca illustration

II. Fisher

Introtext

Math

How we use it or why we don't use it

Intermediate result

VI. CLASSIFIERS

Classifiers were first known from the world of linear regression. The classifiers found in this section are featured in the non-linear signal processing and pattern recognition course. The section seeks to explain the basis of each of the classifiers along with how we have used them in our project. Intermediate results can be found in the section about the classifiers while the comparison between classifiers can be found in the Results section.

I. Linear Classifier

The goal of linear classification is to take an input vector with multiple x values and assign it to one of multiple classes K . This can be done with one or more linear decision boundaries. The first way to classify is called the one-vs-one linear classifier. This works for 2 classes as seen in figure 3.

If multiple clusters of x belonging to more than 2 classes are present we get ambiguous regions as one class might appear to be two different classes. An example of this can be seen in figure 4.

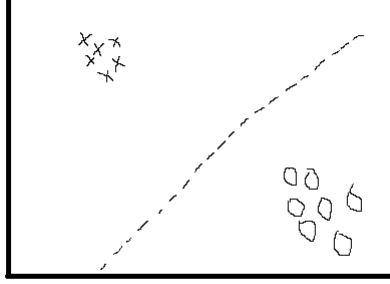


Figure 3: One-vs-one linear classifier for 2 classes

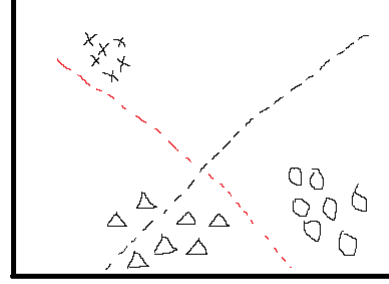


Figure 4: One-vs-one linear classifier for 3 classes

Another way to classify the 3 classes seen in figure 4 could be to utilise 1-of-k classification. This can be seen in figure 5. The 1-of-k classifier has no ambiguity in this case.

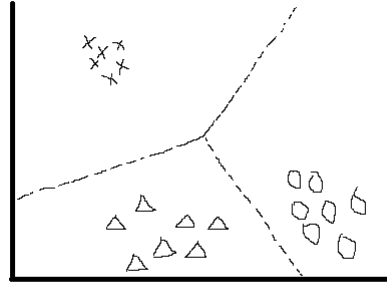


Figure 5: 1-of-k linear classifier for 3 classes

In math terms the one-vs-one can be written as:

$$y(\mathbf{x}) = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}} \quad (10)$$

This is because we can consider the output y to be a weighted sum of the inputs. The error function can be defined as:

$$E(w) = \sum_n (\hat{y}(w, x_n) - y_n)^2 \quad (11)$$

Where \hat{y} is the estimated y value and y_n is the true y value.

If we look at a case with more than two classes, the linear classifier is prone to ambiguity. We know that the ambiguity issue can be avoid by using the form:

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + \omega_{k0} \quad (12)$$

and choosing the value of x to be a part of class k if $y_k(x) > y_m(x)$ for all $m \neq k$. This leads to decision boundaries corresponding to the 1-of-k classifier where the decision boundaries join together in the middle corresponding to the image in figure 5.

Training:

Training the one-of-k function requires the use of two vectors in matlab: t & Z . t is vector of the correct classes while Z is a vector containing our features. In order to train the one-of-k classifier we use the following equation:

$$w^* = (Z^T Z)^{-1} Z^T t \quad (13)$$

This results in the estimated weights for the classifier. To classify the data we use the cost function described earlier in equation 10.

In order to observe the boundaries in the project, the data must be 2 or 3 dimensional. This will require either the use of the PCA or fisher reduction methods explained in early sections. This leads to the image seen in figure 6.

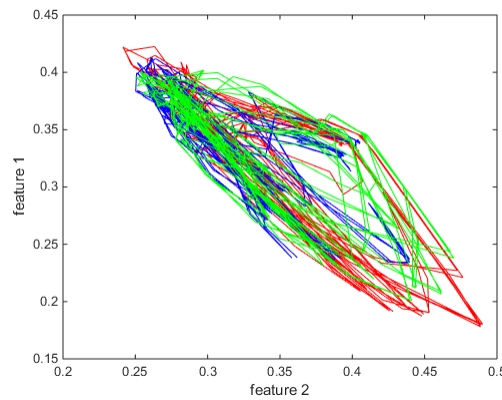


Figure 6: 2 dimensional 1-of-k linear classifier for 3 classes of speech

This does not provide a usable visual representation of the classifier. The choice was made to keep the data in the higher dimensions. The output from the cost function provides a sample and the values representing the three classes:

0.5333
0.2506
0.2160

The cost function classifies the first sample to belong to class 1 as an example.

Intermediate results:

The test data was split into 3 sections and run through the cost function. This resulted in three plots as can be seen in figure 7, 8 & 9. The classes are coloured: Class 1 = Red, Class 2 = Blue, Class 3 = Green.

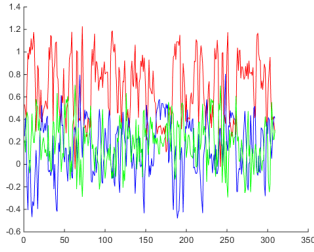


Figure 7: Output from first 1/3 of the data

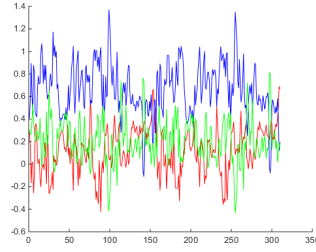


Figure 8: Output from middle 1/3 of the data

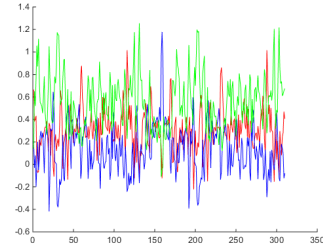


Figure 9: Output from last 1/3 of the data

When evaluating the peak values of the three plots, it can be observed that the first 1/3 of the data belongs to class 1, the middle 1/3 of the data belongs to class 2 and the rest of the data belongs to class 3.

II. Probabilistic models

In many cases the features follows a certain distribution. By using the information in the distribution it is possible to filter out outliers and determine how likely it is that the point belongs to a certain class. This can be very powerful since we not blindly put a sample in a class but also get information about the likelihood. This of course demands that a qualified guess of the distribution can be made. By using the probability for a given sample, given a certain class $P(x|C)$. Iteration over the the different classes, it is possible to decide were to classify the sample, and how certain we are of this decision. This is illustrated in figure X. (fix me)
figure 2

II.1 Discriminative Model

Instead of asking what the probability of the sample, given the class is $P(x|C)$, the conditional probability can be used. That is the probability of a class, given a sample $P(C|x)$. This can be found using Bayes rule:

$$P(C|x) = \frac{P(x|C)P(C)}{P(x)} \quad (14)$$

This kind of model is called a discriminative model, and is used for modelling the dependence of an unobserved class C on an observed variable x . This will for the Gaussian distribution be a sigmoid function. Using this model to classify it is no longer able to tell about the probability of a sample not being in any class, but on the same time also simplifies the classifier a lot. Compared to the linear classifier the discriminative classifier are able to create a mouth sharper decision bound. If the sharper decision bound is the goal then a easer approach is to estimate the optimal sigmoid for separation of classes directly. This can be done by optimizing a soft-max function to separate the classes. The soft-max function is expressed as:

$$Y_k(w_k, x) = \frac{e^{w_k x}}{\sum_{k=1}^K e^{w_k x}} \quad (15)$$

Comparing this to the previous probabilist function we can assume that:

$$P(t|w, x) = p_n^t (1 - p_n)^{1-t}, t \in [0, 1] \quad (16)$$

Here we see how likely it is that the class vector t , is correct given data point x , and some weights w in the soft-max. Where t is the class vector, that indicates which class the data point x is part of. The p_n is given by:

$$p_n = P(C|w, x) = y(w, x) \quad (17)$$

The challenge is now to find the optimal weights w_k , for each class, to create the best classifier. This can be done by combining the two equations 15, 16 to create a non linear optimisation problem.

$$L(w) = \log \prod_{i=1}^N y(w, x_i)^{t_i} (1 - y(w, x_i))^{1-t_i} = \sum_{i=1}^N t_i \log y(w, x_i) + (1 - t_i) \log(1 - y(w, x_i)) \quad (18)$$

This can be solved by many different optimizations strategies. By optimizing this for each class we will find the optimal weights for the soft-max class separator in equation 15.

In the speaker recognition case a soft-max function has been used to create a discriminative model, that are able to separate the classes as described above. A full probabilistic model using the $P(x|C)$ approach, has also been attempted by using a Gaussian mixture model. This is described in the section: II.6. In this analysis it is assumed that the data is Gaussian distributed. This assumption is made by looking at the histogram of the features. This is shown on figure (fix me). show figure 1

By running the optimizing the soft-max function a model is found that are capable of separating the test set in the three classes, whit a 21.58 % error rate. The results can be seen in table 1.

Total Error	21.58 %
Nicolai Error	24.27 %
Rasmus Error	10.68 %
Rune Error	29.77 %

Table 1: Discriminative Model Results

The results indicate that Rune is the one there is hardest to classify. Even though some samples are classified wrong, the majority of samples is correct classified. This makes this method valid for speaker recognition.

(fix me) mÃske have awesome glud billed .

II.2 Gaussian Mixture Models

One of the most common distributions is the Gaussian distribution $\mathcal{N}(\mu, \sigma)$. Sometimes this model is not quiet enough to capture the underlying distribution. In such cases can a mixture of Gaussian distributions be created to fit a underlying distribution. This principal is shown in figure 10.

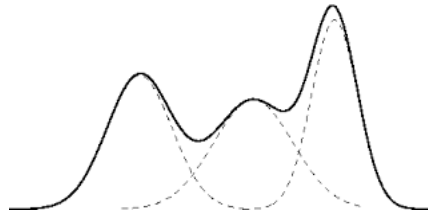


Figure 10: Gaussian Mixture Model Principe

This method can be generalized up in a multidimensional space, to fit the feature space. In a multidimensional model we introduce the covariance matrix Σ that like σ gives the deviation of a Gaussian model. The Gaussian mixture model can therefore be described as:

$$P(x) = \sum_{k=1}^K \Pi_k \mathcal{N}(x|\mu_k, \sigma_k) \quad (19)$$

Where Π_k is a weight called the mixture parameter, that is used to differentiate the individual Gaussian distributions in the model. knowing this we can also describe the mixture model from equation 19 as:

$$P(x) = \sum_{k=1}^K P(k)P(x|k) \quad (20)$$

In order to use a the mixture model, the different distributions must first be found. This topic will be discussed in section II.3 and II.4

II.3 k-means Algorithm

One way of finding groupings of data, in the feature space, is by using the k-means algorithms. This algorithm tries to split the data in k groups. This is done by following 4 three step iterating proses.

1. Initialize means: Select k random mean values in the feature set.
2. Assign responsibility: For each point, find the closest mean points and make it responsible for that point.
3. Calculate new mean: Move each mean point to the mean value of the cluster of points the mean is responsible for.
4. Continue whit step 2, till no change in points.

On figure 11 is illustrated how the k-means in 6 iterations split the data up in 3 clusters.

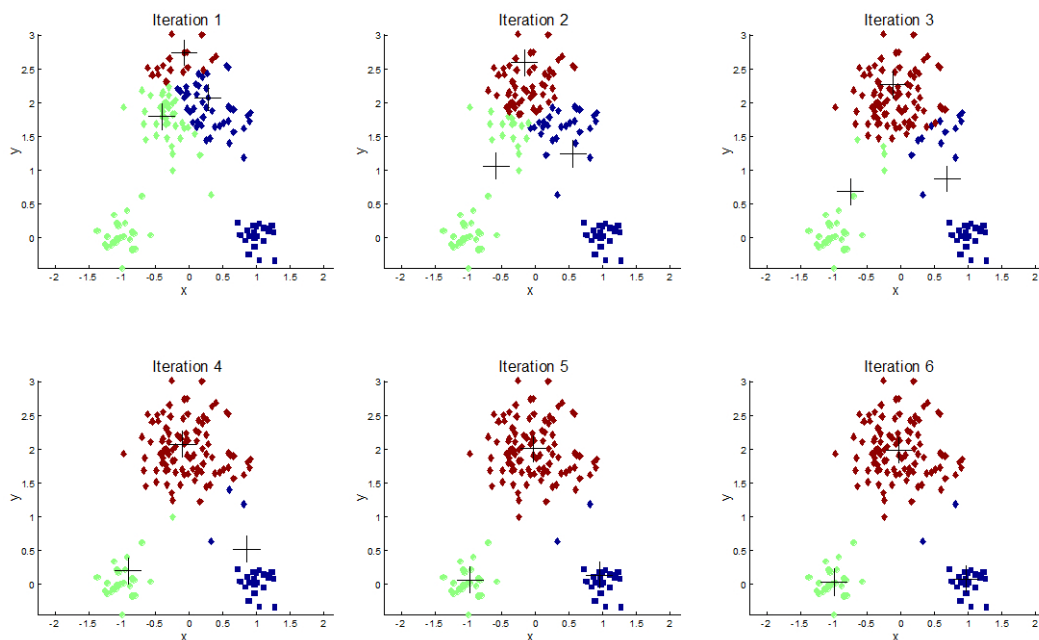


Figure 11: *K-means used on a data set*

The result of this process will differ depending on the initial mean guesses. This is especially the case if there is no natural groupings in the dataset. In this case the algorithm will still split up the data into k clusters even if they are side by side. This is illustrated on figure 13, where the k -means algorithm tries to cluster points distributed uniformly on a circle, in 7 clusters.

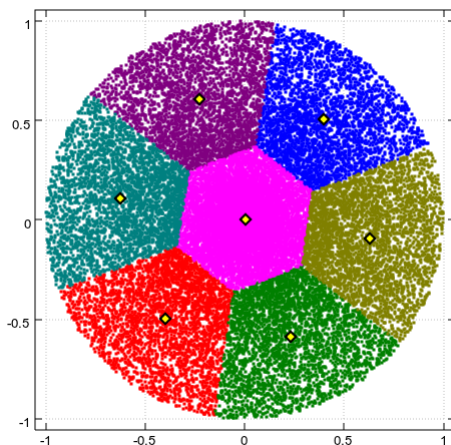


Figure 12: *K-means used on points distributed uniformly on a circle*

II.4 EM Algorithm For Gaussian Mixture Models

As discussed in section II.2 it is important to have a method of finding the best Gaussian distributions in the data set to create a good mixture model. The best Gaussian Mixture Models

that fits the data can be described as optimising equation 21.

$$L(x) = \sum_{n=1}^N \log \sum_k^K P(k)P(x_n|k) \quad (21)$$

To find the maximum of equation 21, we find the differentiated and set it equal zero.

$$\frac{\partial L(x)}{\partial \mu_k} = 0 \quad (22)$$

From equation 19 and 21 the following optimisations equations can be found:

$$N_k = \sum_k^K P(k|x_n) \quad (23)$$

$$P(k) = \Pi_k = \frac{N_k}{N} \quad (24)$$

$$\Sigma_k = \frac{1}{N_k} \sum_n^N P(k|x_n)(x_n - \mu_k)(x_n - \mu_k)^T \quad (25)$$

$$\mu_k = \frac{1}{N_k} \sum_n^N P(k|x_n)x_n \quad (26)$$

The variable N_k can be seen as the effective number of samples in a cluster, and N is the total number of samples. The $P(k|x_n)$ term can be seen as the responsibility a center point has for that point. Unlike the K-means algorithm the all center points are responsible for all points, but the level of responsibility is different from center point to center point. To find the optimal center points μ_k , and there deviations Σ_k the EM algorithm can be used. This is based on the idea of a E-step, called the estimation step, and a M-step called the maximisation step. Much like the k-means does this also have 4 iterative steps.

1. Initialize Parameters: Select k random values for μ_k , Σ_k , Π_k
2. E-Step: Update the responsibility by using Bayes' theorem stating:

$$P(k|x) = \frac{P(x|k)P(k)}{P(x)} = \frac{P(x|k)P(k)}{\sum_k^K P(x|k)P(k)} \quad (27)$$

3. M-Step: Maximize the Gaussian mixture model parameters by using equations 23, 24, 25 and 26.
4. Continue with step 2, till no change in Gaussian mixture model parameters.

Like the k-means algorithm does this algorithm also map a predetermined number of Gaussian distributions to the dataset. The optimal amount must be found through exploration. On figure ?? we see how the EM algorithm has been used to map 3 Gaussian mixture model to a dataset.

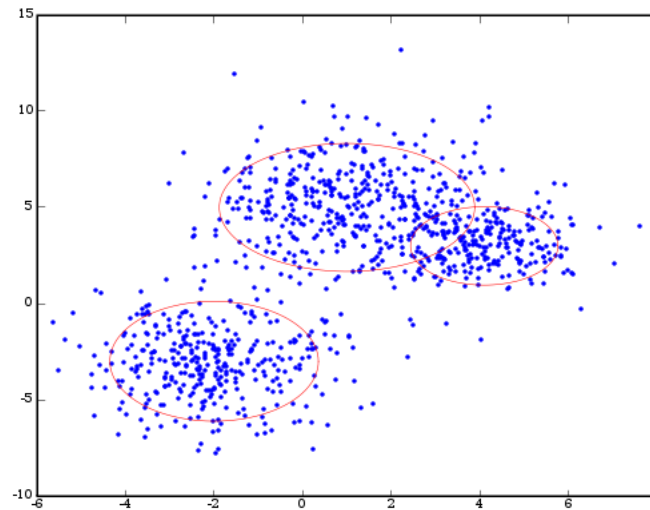


Figure 13: *EM Algorithm Used On Data To Create GMM*

Like for the k-means the initial start estimates for the mean value is important for a good result, and a fast execution time. A normal approach is to use the k-means algorithm first, and use the final means as the start estimate in the em algorithm.

II.5 Unsupervised Gaussian mixture model

hvað er det
hvordan er det brukt i projektet
del resultat

II.6 Supervised Gaussian mixture model

hvað er det
hvordan er det brukt i projektet
del resultat

III. Artificial Neural Network Classifier

e.g. graphical network model, training method, model flexibility (expressive power)
Introtext
Math
How we use it or why we don't use it
Intermediate result

IV. Sequential Models

Markov model and Hidden Markov Model.
e.g. meaning of parameters, left-to-right model, outline of training/testing method.
Introtext

Math

How we use it or why we don't use it

Intermediate result

V. Support Vector Machines

e.g. decision function, support vectors, soft margins, kernel trick.

Introtext

Math

How we use it or why we don't use it

Intermediate result

VII. RESULTS

Compare all the methods in a table in order to show the performance.

VIII. DISCUSSION

I. Subsection One

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

II. Subsection Two

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetur at, consectetur sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

IX. CONCLUSION

REFERENCES

[Figueredo and Wolf, 2009] Figueredo, A. J. and Wolf, P. S. A. (2009). Assortative pairing and life history strategy - a cross-cultural study. *Human Nature*, 20:317–330.