A SUPERVISED APPROACH TO TRAINING GAUSSIAN MIXTURE MODEL CLASSIFIERS

by

Vineet Dilip Gundecha

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

July 2017

Acknowledgements

I would like to sincerely thank my supervising professor Dr. Michael Manry for his guidance and advice. His continual support was essential in successfully completing my thesis.

I dedicate this thesis to my parents, Mr. Dilip Gundecha and Mrs. Savita Gundecha. Their support and encouragement helped me throughout my Master's.

July 31st, 2017

Abstract

A SUPERVISED APPROACH TO

TRAINING

GAUSSIAN MIXTURE

MODEL CLASSIFIERS

VINEET DILIP GUNDECHA, MS

The University of Texas at Arlington, 2017

Supervising Professor: Dr. Michael T. Manry

A new method for training Gaussian Mixture Model (GMM) classifiers is presented. First, an objective function is defined in terms of the number of clusters, K, per class, the mean vectors, the inverse covariance matrices for each class, and the prior probabilities for each class. For each increment in K, gradients of the objective function improve upon the prior probabilities, mean vectors, and inverse covariance matrices. Improvement in accuracy for different datasets are shown and results are compared with the EM algorithm.

List of Figures

List of Tables

Chapter 1

INTRODUCTION

This chapter introduces the problem of classification in the field of pattern recognition. It also gives a brief review on Gaussian Naïve Bayes classifiers and Gaussian Mixture Model classifiers.

1.1 Classifiers

In pattern recognition and machine learning, classification is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations (or feature vectors) whose category membership is known [1]. An example would be classifying a cancer tumor as benign or malignant as described by observed characteristics of the patient and the tumor. Machine learning has enjoyed remarkable success in recent days on a wide range of tasks – e.g., image classification [25], speech recognition [26], spam filtering [27]. An algorithm that implements classification is known as a classifier. Formally, classifiers are mathematical functions that map input data to a category. Some examples of classifiers are perceptron [5], support vector machines [2], decision trees [3], multi-layer perceptron [4]. All classifiers have a set of parameters that need to be trained using the training data. The procedure of tuning these weights is called as training, and the algorithm used for training is called the training algorithm. Training algorithms are broadly classified as supervised or unsupervised. In supervised learning algorithms, the training data is labeled, i.e. the category membership for all examples is known.

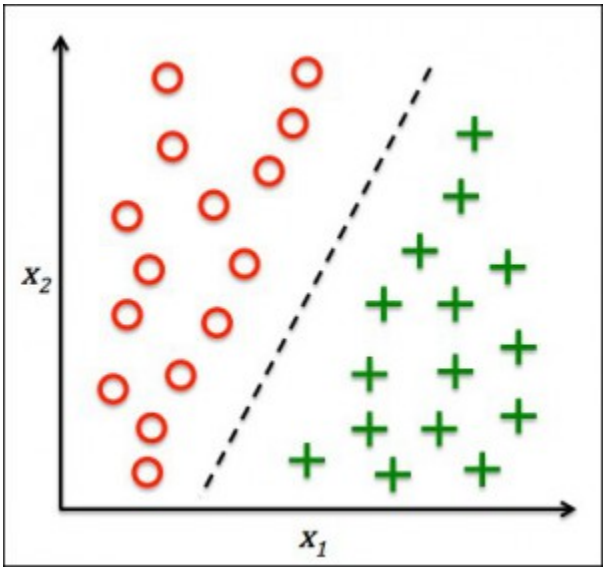

Figure 1.1: A classifier separates the input space. The circles and crosses are patterns belonging to two different classes. The dotted line is the decision boundary.

1.2 Classifier Design

An example pattern recognition system is shown in figure 1.1 below. The input is first processed by a feature extractor to extract information that can be helpful for discriminating the classes. As an example, for a system that classifies binary images of handwritten digits, counting the number white pixels, or detecting edges can be helpful for classification. The extracted features are then given as input to the classifier. The process of providing input and allowing the classifier to adapt its parameters is called learning. In supervised learning [7], the classifier is also provided with the desired outputs along with the inputs. After training, the classifier performance is evaluated on a testing set which consists of input patterns that the classifier has not seen while training.

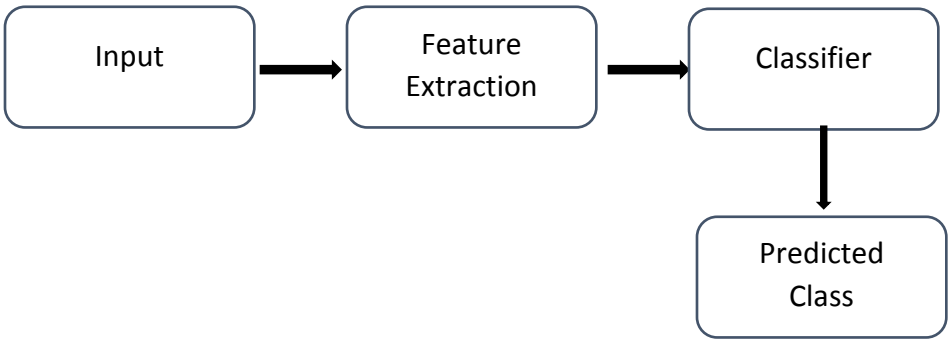

Figure 1.2: Pattern recognition system

1.2 Bayes classifier

Bayes Classifiers try to minimize the probability of error. In order to construct a Bayes classifier, we need to model the conditional density of the feature vectors given the class, denoted by $f(\boldsymbol{x}|i)$. This is then combined with the a-priori probabilities of each class, $P(i)$, to get the discriminants $d(i)$ using the Bayes rule [6]. The class with the maximum value of the discriminant is chosen as the predicted class.

There are three common forms of the Bayes discriminant.

(B1) $d(i) = P(i)f(\boldsymbol{x}|i)$

(B2) $d(i) = \mathrm{g}(P(i)f(\boldsymbol{x}|i))$, where g() is an increasing function

(B3) $d(i) = P(i \mid \boldsymbol{x})$

For all the above forms of discriminants, the class with the maximum value of the discriminant is chosen as the predicted class. It is difficult to estimate the conditional densities in (B1) and (B2) above because of the limited training data available. Different classifiers make some assumptions about the data to simplify the density.

1.3 Bayes Gaussian Classifier

Bayes Gaussian classifiers use the same idea as Bayes classifier in that they use the Bayes rule to calculate the discriminants of each class. But they simplify the problem of estimating the conditional density of the feature vectors given the class by making the assumption that the feature vectors come from a multi-variate Gaussian distribution [10]. Bayes Gaussian classifiers model the conditional probability density function of the data given the class as a multi-variate Gaussian distribution [23]. A Gaussian distribution is parameterized by a mean vector and a covariance matrix. The training procedure of a Bayes Gaussian classifier involves finding the mean vector and covariance matrix of each class. Bayes theorem is used to combine the probability density with the prior probabilities of each class to get the posterior probabilities of the classes given the data. A simple decision rule of picking the most probable class is used to obtain the predicted class.

A modification of Bayes Gaussian classifiers is the naïve Bayes classifier. Along with assuming that the feature vectors come from a Gaussian distribution, naïve Bayes classifiers also assume that the features are independent. This leads to a simplified form of the conditional density. Naïve Bayes finds applications in medical diagnosis [11], text retrieval [12], credit scoring [24], etc. Gaussian naïve Bayes classifiers tend to perform poorly if the feature vector for a class does not come from a Gaussian distribution. For e.g.., a single Gaussian is not suitable to model a bimodal Gaussian distribution. To address this issue, Gaussian mixture models are used.

1.4 Gaussian Mixture Model (GMM)

Gaussian mixture models [8] are a natural extension to naïve Bayes and Bayes Gaussian classifiers. Instead of modelling the probability density of each class as a single Gaussian, GMM uses a sum of multiple Gaussians. The contribution of each Gaussian is weighted. The procedure of finding the mean vectors and covariance matrices is more

complicated than naïve Bayes. The popular algorithm to train GMM is the EM algorithm [9]. GMM have recently been used for feature extraction from speech data in speech recognition [15]. They have also been used in object tracking of multiple objects [16]

1.5 Proposed method

This thesis proposes a novel training algorithm for Gaussian naïve Bayes and Gaussian mixture model classifiers. Instead of using the EM algorithm for optimizing the mean vectors and covariance matrices, we use gradients of an error function w.r.t the elements of the mean vectors and the covariance matrices to iteratively update them.

Chapter 2 reviews the complete structure of GMM classifiers, including the Expectation Maximization algorithm. Chapter 3 explains the proposed algorithm for training GMM classifiers using the gradient descent technique. Chapter 4 introduces different variants of the proposed algorithm. Chapter 5 presents the results on different datasets

Chapter 2

STRUCTURE OF BAYES GAUSSIAN, NAIVE BAYES AND GAUSSIAN MIXTURE MODEL CLASSIFIERS

This chapter reviews the structure of Bayes Gaussian, naïve Bayes and Gaussian mixture model classifiers. It also introduces notation that will be used for the rest of the chapters.

2.1 Notation

N is the number of features (inputs) in each pattern. $\boldsymbol{x_p}$ is the $\mathrm{p^{th}}$ N-dimensional input vector. N_v is the number of training patterns in the training data. N_c is the number of classes. $N_v(i)$ is the number of training patterns for the i^{th} class. $\boldsymbol{m}_i$ is the N-dimensional mean vector of the i^{th} class. $\boldsymbol{\Sigma}_i$ is the NxN covariance matrix of the i^{th} class. K is the number of components in a mixture. $N_v(k,i)$ is the number of patterns in the k^{th} component of the i^{th} class. $\boldsymbol{m}_{ik}$ is the N-dimensional mean vector for the k^{th} component of i^{th} class. $\boldsymbol{\Sigma}_{ik}$ is the NxN covariance matrix for k^{th} component of i^{th} class. $\mathbf{a}_{ki}$ is the weighting parameter for the k^{th} component of the i^{th} class. P(i) is the a-priori probability of the i^{th} class. $\boldsymbol{t_p}$ is the $\mathrm{p^{th}}$ $\mathrm{N_c}$ dimensional target vector. $i_c(p)$ is the correct class or the ground truth for the p^{th} pattern

2.2 Structure of Bayes Gaussian classifiers

Bayes Gaussian classifiers are simple probabilistic classifiers based on the Bayes rule. Given the training data, these classifiers model the probability density function for each class as a multivariate Gaussian. Each Gaussian is parameterized by a mean vector and a covariance matrix that is calculated from the training data. The discriminant is calculated by using the Bayes rule which combines the prior probabilities of each class with the modeled probability density.

1) For each class i, the mean vector is estimated as

$$m_i(n) = \frac{1}{N_v(i)} \sum_{p:i_c(p)=i}^{N_v(i)} x_p(n) \tag{2.2.1}$$

for 1 ≤ n ≤ N

2) The i^{th} class covariance matrix is then estimated as

$$\boldsymbol{\Sigma}_i = \frac{1}{N} \sum_{p=1}^{N_v} (\boldsymbol{x}_p - \boldsymbol{m}_i)(\boldsymbol{x}_p - \boldsymbol{m}_i)^T \tag{2.2.2}$$

3) Once we have the mean vector and covariance matrix, the conditional density is given as:

$$f(\boldsymbol{x}|i) = \frac{exp\left(-\frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{m}_i\right)^T \boldsymbol{\Sigma_i}^{-1}\left(\boldsymbol{x} - \boldsymbol{m}_i\right)\right)}{\sqrt{(2\pi)^N |\boldsymbol{\Sigma_i}|}} \tag{2.2.3}$$

4) Combining the conditional density with the a-priori probabilities, the B3 Bayes discriminants are calculated as

$$d_p(i) = P(i \mid \boldsymbol{x}_p) = \frac{P(i) f(\boldsymbol{x}_p \mid i)}{\sum_{j=1}^{N_c} P(j) f(\boldsymbol{x}_p \mid j)} \tag{2.2.4}$$

The class with the maximum value of the Bayes discriminants is the predicted class

$$i_p' = argmax_i\big(d_p(i)\big) \tag{2.2.5}$$

2.3 Structure of Gaussian Naïve Bayes classifiers

One major problem in the Bayes Gaussian classifier is that we need to calculate the inverse of the covariance matrix $\boldsymbol{\Sigma_i}$. This is difficult if the matrix is singular or close to singular.

Gaussian Naïve Bayes overcome this problem by making strong independence assumptions between the features. Since the features are independent, the covariance between them is 0. This assumption leads to a diagonal covariance matrix which is easy to invert. The diagonal terms represent the variance of each feature.

Following are the details for calculating the discriminants for a Gaussian Naïve Bayes classifier -

1) For each class i, the mean vector is estimated as

$$m_i(n) = \frac{1}{N_v(i)} \sum_{p:i_c(p)=i}^{N_v(i)} x_p(n) \tag{2.2.1}$$

for 1 ≤ n ≤ N.

2) The variances are then calculated as

$$\sigma_i^2(n) = \frac{1}{N_v(i)} \sum_{p:i_c(p)=i}^{N_v(i)} (x_p(n) - m_i(n))^2 \tag{2.2.2}$$

for 1 ≤ n ≤ N.

3) Once we have the mean vector and covariance matrix, the conditional density of $\boldsymbol{x}$ is given as:

$$f(\boldsymbol{x}_p \mid i) = \frac{\exp\left(\frac{-1}{2}\sum_{n=1}^{N}(x_p(n) - m_i(n))^2 v_i(n)\right)}{\sqrt{(2\pi)^n \prod_{n=1}^{N} \sigma_i^2(n)}} \tag{2.2.3}$$

where $v_i(n) = \sigma_i^2(n)^{-1}$.

4) Combining the conditional density with the a-priori probabilities, the B3 Bayes discriminants are calculated as

$$d_p(i) = P(i \mid \boldsymbol{x}_p) = \frac{P(i)\, f(\boldsymbol{x}_p \mid i)}{\sum_{j=1}^{Nc} P(j)\, f(\boldsymbol{x}_p \mid j)} \tag{2.2.4}$$

The class with the maximum value of the Bayes discriminants is the predicted class

$$i_p' = \mathrm{argmax}_i\big(d_p(i)\big) \tag{2.2.5}$$

2.4 Structure of Gaussian Mixture Models

A Gaussian mixture model [8] is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One hint that data might follow a mixture model is that the data looks multimodal, i.e. there is more than one "peak" in the distribution of data. Trying to fit a multimodal distribution with a unimodal (one "peak") model will generally give a poor fit, as shown in the example below. Since many simple distributions are unimodal, an obvious way to model a multimodal distribution would be to assume that it is generated by multiple unimodal distributions [17]. A Gaussian mixture model is parameterized by two types of values, the mixture component weights and the component means and covariances.

A Gaussian mixture model pdf with K components can be represented as follows:

$$f(\boldsymbol{x}) = \sum_{k=1}^{K} a_k \frac{exp\left(-\frac{1}{2}\left(\boldsymbol{x}-\boldsymbol{m}_k\right)^T \boldsymbol{\Sigma_k}^{-1}\left(\boldsymbol{x}-\boldsymbol{m}_k\right)\right)}{\sqrt{(2\pi)^N |\boldsymbol{\Sigma_k}|}} \tag{2.3.1}$$

The weights, a_k, represent the contribution of the corresponding component to the resulting density. The weights a_k are positive and have values between 0 and 1. The sum of these weights add up to 1 so that total probability distribution normalizes to 1.

$$\sum_{k=1}^{K} a_k = 1 \tag{2.3.2}$$

2.5 Training a Gaussian Mixture Model using Expectation Maximization

Training a Gaussian mixture model involves finding the values for the parameters $\boldsymbol{m_k}, \boldsymbol{\Sigma_k}, a_k$. A common approach is to use maximum likelihood estimation which seeks to maximize the likelihood of observing the data under the given model parameters. This can be done by differentiating the likelihood w.r.t. the model parameters and set it to 0. For mixture models, this approach turns out to be analytically impossible.

The technique most commonly used to train mixture models is Expectation Maximization (EM) [9] which is an iterative method for maximum likelihood estimation of parameters in a statistical model. EM iterations alternate between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E step . Following are the details of the algorithm:

The components are usually initialized using a clustering algorithm like K-means [13].
The initial values for the parameters can be calculated from these clusters.

The n^{th} element of mean vector for component k is given as

$$m_k(n) = \frac{1}{N_v(k)} \sum_{p:k_c(p)=k}^{N_v(k)} x_p(n) \tag{2.5.1}$$

for 1 ≤ k ≤ K, 1 ≤ n ≤ N, where $N_v(k)$ is the number of patterns belonging to cluster k, and $k_c(p)$ is the cluster membership of pattern p.

The covariance matrix can for component k is given as

$$\Sigma_k = \frac{1}{N_v(k)} \sum_{p:k_c(p)=k}^{N_v(k)} (\boldsymbol{x}_p - \boldsymbol{m}_k)(\boldsymbol{x}_p - \boldsymbol{m}_k)^T \tag{2.5.2}$$

The component weights are initialized as

$$a_k = \frac{N_v(k)}{N_v} \tag{2.5.3}$$

The algorithm alternates between an Expectation step and a Maximization step described as below:

Expectation Step:

Calculate for all values *p, k*

$$P(k \mid \boldsymbol{x}_p) = \frac{P(k) f(\boldsymbol{x}_p \mid k)}{\sum_{j=1}^{K} P(j) f(\boldsymbol{x}_p \mid j)} \tag{2.5.4}$$

where $P(k|\boldsymbol{x_p})$ is the probability that $\boldsymbol{x_p}$ is generated by component k.

Maximization Step:

Using $P(k|\boldsymbol{x_p})$ in the Expectation step, the parameters of the model are updated. First, the mean vectors for the component k is updated as a weighted average of all the patterns as per equation 2.4.2. The weight is the probability that the pattern was generated by that component. Thus, patterns with a large weight will tend to pull the mean vectors towards them.

$$\boldsymbol{m}_k = \frac{\sum_{p=1}^{Nv} P(k \mid \boldsymbol{x}_p)\boldsymbol{x}_p}{\sum_{p=1}^{Nv} P(k \mid \boldsymbol{x}_p)} \tag{2.5.5}$$

Similarly, the covariance matrix is updated as a weighted cross product

$$\boldsymbol{\Sigma}_k = \frac{\sum_{p=1}^{Nv} P(k \mid \boldsymbol{x}_p)(\boldsymbol{x}_p - \boldsymbol{m}_k)(\boldsymbol{x}_p - \boldsymbol{m}_k)^T}{\sum_{p=1}^{Nv} P(k \mid \boldsymbol{x}_p)} \tag{2.5.6}$$

The component weights are updated as the mean of $P(k|\boldsymbol{x}_p)$ for all the patterns. If $P(k|\boldsymbol{x}_p)$ is large for some component k, that component will have a greater contribution to the density.

$$a_k = \frac{1}{N_v}\sum_{p=1}^{N_v} P(k|\boldsymbol{x}_p) \tag{2.5.7}$$

The training iterates over these two steps until the parameters converge. The EM algorithm guarantees that the likelihood increases after each step. Usually, the training is stopped if the change in likelihood is smaller than a chosen tolerance value. EM can get stuck in a local maximum or a saddle point. Also, it is sensitive to the initial values of the parameters.

2.6 Gaussian Mixture Model classifiers

Gaussian mixture models can also be used for classification. We can think about this as an extension of the Bayes Gaussian classifier. Instead of using a single Gaussian to model the conditional density, we use a mixture of Gaussians. The densities are then

combined with the prior probabilities of each class and the discriminant is calculated using the Bayes rule.

GMM classifiers are initialized using K-means. Each class is clustered into K components. The initial values of the component weight are calculated as

$$a_{ki} = \frac{N_v(k,i)}{N_v(i)}$$

The i^{th} class density is given as:

$$f(\boldsymbol{x}|i) = \sum_{k=1}^{K} a_{ki} \frac{exp\left(-\frac{1}{2}\left(\boldsymbol{x}-\boldsymbol{m}_{ik}\right)^T \boldsymbol{\Sigma_{ik}}^{-1}\left(\boldsymbol{x}-\boldsymbol{m}_{ik}\right)\right)}{\sqrt{(2\pi)^N|\boldsymbol{\Sigma_{ik}}|}} \quad (2.6.1)$$

Calculate the discriminants as:

$$d_p(i) = \frac{P(i)\,f(\boldsymbol{x}_p\,|\,i)}{\sum_{j=1}^{Nc} P(j)\,f(\boldsymbol{x}_p\,|\,j)} \quad (2.6.2)$$

The predicted class is given

$$i_p{}' = \mathrm{argmax}_i\left(d_p(i)\right) \quad (2.6.3)$$

Training a Gaussian mixture model classifier involves modeling the probability density for each class separately. The number of components K for each mixture is usually chosen by evaluating the performance of the classifier on a validation set

Training algorithm for Gaussian mixture model classifiers using EM

(Initialize K = 2, separate training data into training and validation)

1. For each class i, cluster the patterns belonging to that class into K clusters using K-means ++ [14]. Calculate the component weights as

$$a_{ki} = \frac{N_v(k,i)}{N_v(i)}$$

2. Run EM on each class separately to get the estimated $\boldsymbol{m}_{ik}$, $\boldsymbol{\Sigma_{ki}}$, a_{ki}
3. Calculate $P(i)$ as $N_v(i)/N_v$
4. Use equation 2.5.2 to calculate the discriminants for each pattern
5. Use equation 2.5.3 to get the predicted class
6. Evaluate the performance on a validation set, save the model if the validation accuracy is improved
7. Increment K by 1 and go to 1
8. Pick the best value of K using the validation accuracy.

2.7 Problems with Gaussian Mixture Model classifiers

1. Mean vectors initialized using K-means may not be optimal for the task of classification. Mean vectors found using K-means are optimized for minimizing the Euclidean distance between them and the points in the cluster. They may be good for purpose of clustering, but not for classification. EM is used to improve the mean vectors $\boldsymbol{m}_{ik}$ initialized using K-means. But since EM is used for each class separately, there's no interaction between mean vectors for clusters belonging to different classes.
2. Similarly, $\boldsymbol{\Sigma_{ik}}$, a_{ki} that maximize the log likelihood in EM, are not optimal for classification

3. When using EM, each class is trained separately. Thus, EM needs to be run N_c times.

These problems are the motivation for the proposed algorithm described in the next chapter

Chapter 3

GRADIENT BASED TRAINING OF GAUSSIAN MIXTURE MODEL CLASSIFIERS

In this chapter, a new technique for training of Gaussian mixture model classifier is presented.

3.1 Problem formulation

We formulate the problem of training the classifier as a supervised learning problem. As mentioned before, in supervised learning we adjust the parameters of our model by using the ground truth information. In classification, the ground truth is the class number to which the pattern belongs. This ground truth, or the desired output, can be represented as a N_c-dimensional vector. This vector is now our target output $\boldsymbol{t_p}$. It is defined as below:

$$t_p(i) = \delta(i - i_c) \qquad for\ 1 \le i \le N_c \tag{3.1.1}$$

where δ is the Kronecker delta function [22]. The target output is 1 for the correct class and 0 elsewhere. The output of our classifier is same as in equation 2.5.2. It is represented by the vector $\boldsymbol{y_p}$.

$$y_p(i) = P(i \mid \boldsymbol{x}_p) = \frac{P(i)\, f(\boldsymbol{x}_p \mid i)}{\sum_{j=1}^{Nc} P(j)\, f(\boldsymbol{x}_p \mid j)} \tag{3.1.2}$$

where, $f(\boldsymbol{x_p}|i)$ is the value of the conditional density of $\boldsymbol{x_p}$ for class i

$$f(\boldsymbol{x}_p \mid i) = \sum_{k=1}^{K} a_{ki} \frac{\exp(-\sum_{n=1}^{N} (x_p(n) - m_{ki}(n))^2 v_{ki}(n))}{(2\pi)^{N/2} (\prod_{n=1}^{N} \sigma_{ki}{}^2(n))^{1/2}}$$

We define an error function *E* to quantify the difference between the desired output $\boldsymbol{t_p}$ and the actual output $\boldsymbol{y_p}$.

$$E = \frac{1}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{N_c}\left(t_p(i) - y_p(i)\right)^2 \tag{3.1.3}$$

We want to minimize this error function w.r.t the parameters of the mixture model classifier. There are three different types of parameters – the prior probabilities P(i), the elements of the mean vectors $\boldsymbol{m_{ik}}$, and the elements of the covariance matrices $\boldsymbol{\Sigma_{ki}}$**.** As mentioned before, we assume that the inputs to the classifier are independent. Thus, the covariance matrices are diagonal and the diagonal elements represent the variance of the corresponding input. The output of the classifier $\boldsymbol{y_p}$ is a function of these three types of parameters.

3.2 Proposed technique for updating the parameters

To minimize the error function, we use the gradient descent technique. Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function [18]. Finding the minimum of a function using gradient descent involves two steps - calculate the gradient of the function at the current point, take steps in the direction of the negative gradient to update the parameters. If a function has multiple minima, gradient descent can get stuck in a local minimum.

It was found during experiments that a variant of the gradient descent technique, called as Adam [19], performed much better. It stands for Adaptive Moment Estimation. Adam keeps an exponentially decaying average of the gradients from all the previous iterations. This quantity is known as the momentum term **p.** We also keep an exponentially decaying average of the squared gradients from all the previous iterations. This is denoted

by **q.** The **p** and **q** arrays have the same dimensions as $\boldsymbol{\theta}$ which denotes the sets of parameters. Both are initialized to 0.

$\boldsymbol{p_0} = 0, \boldsymbol{q_0} = 0$

Consider $\boldsymbol{\theta}$ as a 3-D array containing all the parameters**. F**or iteration t, $p_t(i,j,k)$**,** $q_t(i,j,k)$ can be calculated as follows

$$p_t(i,j,k) = \beta_1 p_{t-1}(i,j,k) + (1-\beta_1) g_t(i,j,k) \quad (3.2.1)$$

$$q_t(i,j,k) = \beta_2 q_{t-1}(i,j,k) + (1-\beta_2) q_t^2(i,j,k) \quad (3.2.2)$$

Here, $g_t(i,j,k)$ represents the gradient at iteration t. β_1, β_2 are numbers between 0 and 1. Typical values proposed are 0.9 for β_1, 0.999 $for\ \beta_2$**.** The parameters at iteration t are given by $\boldsymbol{\theta_t}$, the update equation for parameter $\theta(i,j,k)$ is given as

$$\theta_{t+1}(i,j,k) = \theta_t(i,j,k) - \frac{\alpha}{\sqrt{q_t(i,j,k)} + \in} p_t(i,j,k) \quad (3.2.3)$$

α is the learning rate and ∈ is a small number to prevent division by 0. The learning rate α is chosen heuristically.

The advantage of using Adam is two-fold. First, the momentum term is helpful for accelerating the training in directions where the gradient does not change sign. It also helps in preventing oscillations in steep directions where the gradient is high [20]. Second, the $\boldsymbol{v_t}$ term acts as an adaptive, per-parameter learning rate, similar to that in [21]. This makes it possible to use different learning rate for each parameter. The learning rate is inversely related to the gradient of that parameter.

3.3 Training the parameters

There are number of different ways to perform the training. Since we have three different types of parameters, the a-priori probabilities, the mean vectors, and the inverse covariance elements, number of permutations can be tried. Experiments were carried for the following methods:

i. Training all parameters together

   In this method, all three set of parameters are updated together for each iteration. For each iteration, we calculate the gradients of the error function defined in equation 3.1.1 w.r.t each parameter. The parameters are then updated using Adam as described in the previous section

ii. Training one parameter at a time

   Training in this method occurs in three steps. In each step, one parameter is trained while others are kept constant. For e.g.: In the first step, the mean vectors are trained. In the second step, the trained mean vectors are kept constant and the inverse variances are trained. In the third step, the trained mean vectors and inverse variances are kept constant and the prior probabilities are trained.

iii. Adding one parameter at each step

   This method starts by training a single parameter. At every subsequent step, a new parameter is added and training continues for the newly added parameter as well as for parameter added before.

Additionally, in ii and iii, the order of parameters can also change. During experiments, it was found that training all the parameters together performs the best. Also, it is faster since all parameters are trained together.

3.4 Training only one set of parameters at a time

In this sub-section, we perform training by updating only one out of the three set of parameters and evaluate the performance on different datasets

3.4.1 Training the prior probabilities

Prior probabilities for each are initialized as the fraction of patterns belonging to that class. For most classification datasets, the number of patterns in a class is the same for all classes. So, the prior probabilities indicate that all classes are equally probable. During experiments, it was found that training the prior probabilities does not make much difference in the classifier performance. Prior probabilities should always be positive. This is done by using the square of the square root of those elements while calculating the output. Additionally, the probabilities should also add up to 1. This is ensured by dividing the probabilities by the sum of all probabilities.

3.4.2 Training the mean vectors

For each class, we have K component clusters. As mentioned before, the cluster mean vectors are initialized using K-means++. The cluster centers are initialized to minimize the Euclidean distance between the patterns belonging to that class and the mean vectors. Thus, mean vectors obtained by K-means++ may not be optimal for our final task which is classification. We could optimize the center vectors so that it minimizes the error function defined in equation 3.1.3. Table 2 shows the improvement in accuracy after training the prior probabilities and the mean vectors

3.4.3 Training the inverse variances elements

Similar to the mean vectors, the inverse variances are calculated using clusters initialized by K-means. As reasoned before, they may not be optimal. We use the same technique to optimize the inverse variances. Since variances are always positive, it is important to ensure that they don't go negative while training. This is done by using the square of the square root of those elements while calculating the output.

Dataset	*Number of inputs*	*Number of classes*	*Accuracy before training prior probabilities*	*Accuracy after training prior probabilities*
Comf18	18	4	63.21	64.87
Grng	16	4	81.0	81.2
Gongtrn	16	10	89.2	89.4
F17C	17	39	96.2	97.2
Phoneme	5	2	78.9	80.6
M-feat	6	10	73.4	73.8

Table 1: The table shows the improvement in accuracy after training the prior probabilities

Dataset	*Number of inputs*	*Number of classes*	*Accuracy before training mean vectors*	*Accuracy after training mean vectors*
Comf18	18	4	63.21	71.6
Grng	16	4	80.0	85.0
Gongtrn	16	10	89.2	91.4
F17C	17	39	96.4	98.4
Phoneme	5	2	78.9	85.1
M-feat	6	10	73.4	75.0

Table 2: The table shows the improvement in accuracy after training the prior probabilities and mean vectors

Table 1 shows the improvement in accuracy after training the prior probabilities. As mentioned before, this doesn't result in much performance gain. The table shows the improvement in accuracy after training the prior probabilities and the mean vectors. We see significant improvement after training the mean vectors. This indicates that mean vectors initialized by K-means++ are not optimal for classification.

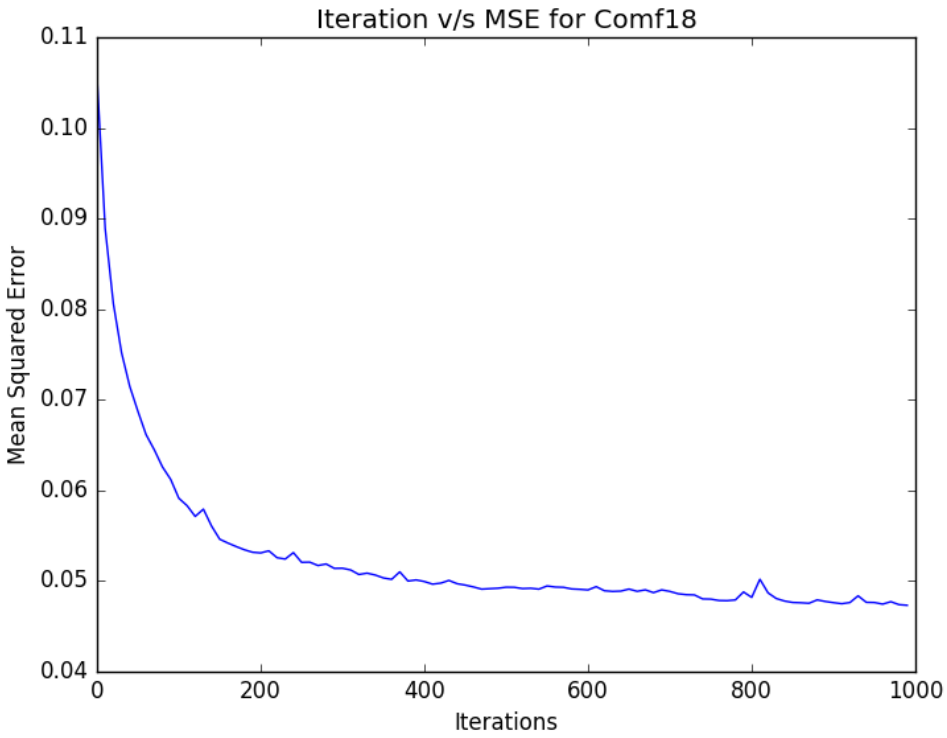

Figure 3.1: This figure shows the decrease in the error as the training progresses

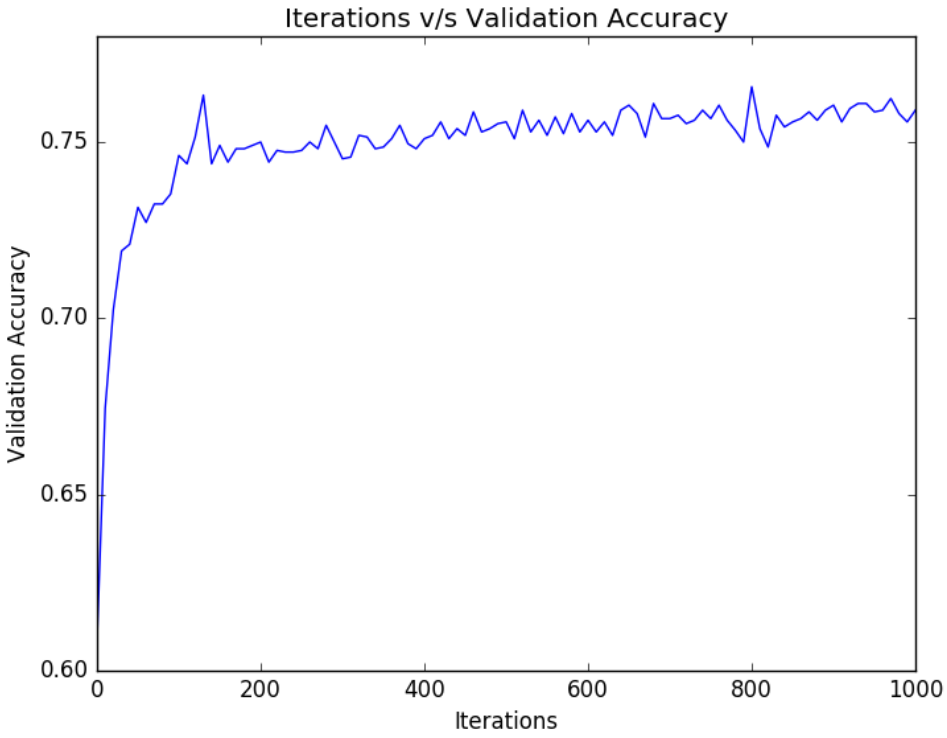

Figure 3.2 The figure shows how training the parameters (only the mean vector in this case) increases the validation accuracy

Chapter 4

FINAL ALGORITHM AND RESULTS

In this chapter, we present the final algorithm, and compare the results with EM.

4.1 Final algorithm

In chapter 3, section 3.5, we presented the results after training the prior probabilities and the mean vectors. For the final algorithm, we train all the parameters together. For each iteration, we jointly optimize all three sets of parameters. To prevent the prior probabilities from going negative, we initialize them as square root of the actual probabilities and use square of those values while calculating the output. Similar trick is used for the variances where they are initialized as standard deviations

The prior probabilities are stored in $\boldsymbol{\theta_p}$. It is a vector of size N_c with elements $\sqrt{P(i)}$. $P(i)$ is given as:

$$P(i) = \frac{N_v(i)}{N_v}$$

The mean vectors are stored in $\boldsymbol{\theta_m}$. It is 3-dimensional array of size $N_c x\ Kx\ N$ with the elements $m_{ki}(n)$.

$$m_{ki}(n) = \frac{1}{N_v(k,i)} \sum_{p:i_c(p)=i,k_c(p)=k}^{N_v(k,i)} x_p(n)$$

where, $i_c(p), k_c(p)$ are the class and cluster membership of pattern p respectively.

$\boldsymbol{\theta_v}$ is also 3-dimensional array of size $N_c x\ Kx\ N$ containing the elements $v_{ki}(n)$ which are the inverse standard deviations $\frac{1}{\sqrt{\sigma_{ki}^2(n)}}$.

$$\sigma_{ki}^2(n) = \frac{1}{N_v(k,i)} \sum_{p:i_c(p)=1,k_c(p)=k}^{N_v(k,i)} (x_p(n) - m_{ki}(n))^2$$

This describes the final training algorithm using the approach described in section 3.1-3.3.

(Initialize K = 2, separate training data into training and validation. Choose a maximum value for K as K_{max})

1. For each class i, cluster the patterns belonging to that class into K clusters using K-means ++ [14]
2. Calculate the mean vectors $\boldsymbol{m_{ki}}$, inverse variances $\boldsymbol{v_{ki}}$ for the clusters. Initialize the prior probabilities P(i) as $N_v(i)/N_v$. Initialize the component weights a_{ki} as $N_v(k,i)/N_v(i)$. Initialize $\boldsymbol{P_m}, \boldsymbol{P_v}, \boldsymbol{P_p}, \boldsymbol{Q_m}, \boldsymbol{Q_v}, \boldsymbol{Q_p}$ as 0
3. For iteration i_t = 1 to N_{it}, where N_{it} is the total number of iterations
4. Calculate $\boldsymbol{y}$ as

$$y_p(i) = P(i \mid \boldsymbol{x}_p) = \frac{P(i) f(\boldsymbol{x}_p \mid i)}{\sum_{j=1}^{Nc} P(j) f(\boldsymbol{x}_p \mid j)}$$

$$f(\boldsymbol{x}_p \mid i) = \sum_{k=1}^{K} a_{ki} \frac{\exp(-\sum_{n=1}^{N} (x_p(n) - m_{ki}(n))^2 v_{ki}{}^2(n))}{(2\pi)^{N/2} (\prod_{n=1}^{N} \sigma_{ki}{}^2(n))^{1/2}}$$

5. Calculate error E and gradients of E (Refer to Appendix A for complete equations). $\boldsymbol{G_m}, \boldsymbol{G_P}, \boldsymbol{G_v}$, denote the matrices of partial derivatives of E w.r.t the mean vectors, prior probabilities and inverse variances respectively. Elements of $\boldsymbol{G_m}$ are $\frac{\boldsymbol{\partial E}}{\boldsymbol{\partial} m_{ki}(n)}$

and are given in equation 6.1.5 of Appendix A. Elements of $\boldsymbol{G_v}$ are $\frac{\partial E}{\partial v_{ki}(n)}$ and are given in equation 6.1.6 of Appendix A. Elements of $\boldsymbol{G_p}$ are $\frac{\partial E}{\partial P(j)}$ and are given in equation 6.1.7 of Appendix A.

6. For all values of $i, k, n\ (1 \le i \le N_c,\ 1 \le k \le K,\ 1 \le n \le N)$ calculate $P_m(i,k,n), P_v(i,k,n), P_p(i), Q_m(i,k,n), Q_v(i,k,n), Q_p(i)$ as follows. β_1 = 0.99,β_2 = 0.999.

$$P_m(i,k,n) \leftarrow \beta_1 P_m(i,k,n) + (1-\beta_1)\, G_m(i,k,n)$$

$$P_p(i) \leftarrow \beta_1 P_p(i) + (1-\beta_1) G_p(i)$$

$$P_v(i,k,n) \leftarrow \beta_1 P_v(i,k,n) + (1-\beta_1)\, G_v(i,k,n)$$

$$Q_m(i,k,n) \leftarrow \beta_2 Q_m(i,k,n) + (1-\beta_2) G_m^2(i,k,n)$$

$$Q_p(i) \leftarrow \beta_2 Q_p(i) + (1-\beta_2) G_p^2(i)$$

$$Q_v(i,k,n) \leftarrow \beta_2\, Q_v(i,k,n) + (1-\beta_2)\, G_v^2(i,k,n)$$

7. Update parameters as

$$\theta_p(i) \leftarrow \theta_p(i) - \frac{\alpha}{\sqrt{Q_p(i)} + \in} P_p(i)$$

$$\theta_m(i,k,n) \leftarrow \theta_m(i,k,n) - \frac{\alpha}{\sqrt{Q_m(i,k,n)} + \in} P_m(i,k,n)$$

$$\theta_v(i,k,n) \leftarrow \theta_v(i,k,n) - \frac{\alpha}{\sqrt{\theta_v(i,k,n)} + \in} P_v(i,k,n)$$

8. Normalize the priors so that they sum up to 1.

$$P(i) \leftarrow \frac{P(i)}{\sum_{j=1}^{N_c} P(j)}$$

9. Check performance on validation data. If performance is better than the previous iteration, save the parameters.

10. Increase i_t by 1. If $i_t < N_{it}$, go to 4
11. End iterations
12. If K < K_{max}. Increase K by 1, go to step 1.

The value of K which performs the best on validation data is chosen as the final value of K. The final values for the prior probabilities and variances are found as:

$$P(i) \leftarrow P^2(i)$$

$$\sigma_{ki}^2 \leftarrow (\sigma_{ki})^2$$

This way the prior probabilities and inverse variances are prevented from going negative.

In the following the table, we show the improvement obtained after training the parameters using the algorithm described and compare it with training the parameters using EM as described in section 2.6.

Dataset	*Number of inputs*	*Number of classes*	*GMM initialized with K-means*	*GMM initialized with K-means and trained with gradient approach*	*GMM initialized with K-means and trained with EM*
Comf18	18	4	63.21 (K = 2)	77.94 (K = 4)	68.67 (K = 4)
Grng	16	4	80.0 (K = 3)	92.0 (K = 2)	83.0 (K = 2)
Gongtrn	16	10	89.2 (K = 4)	92.8 (K = 2)	88.8 (K = 2)
F17C	17	39	96.4 (K = 5)	99.2 (K = 5)	97.0 (K = 3)
Phoneme	5	2	78.96 (K = 2)	87.37 (K = 4)	79.97 (K = 4)
M-feat	6	10	73.4 (K = 2)	76.2 (K = 3)	68.0 (K = 3)

Table 3: This table compares performance for three different algorithms. The final value of K is given in brackets.

We also compare performance for methods described in section 3.5 with the final algorithm. In figure 4.1, we show that training all parameters performs better than training any individual parameter. Also, as mentioned before, training the prior probabilities gives

the least improvement in performance. Training the mean vectors and inverse variances do give a considerable improvement in performance, but it is still less than training all parameters together.

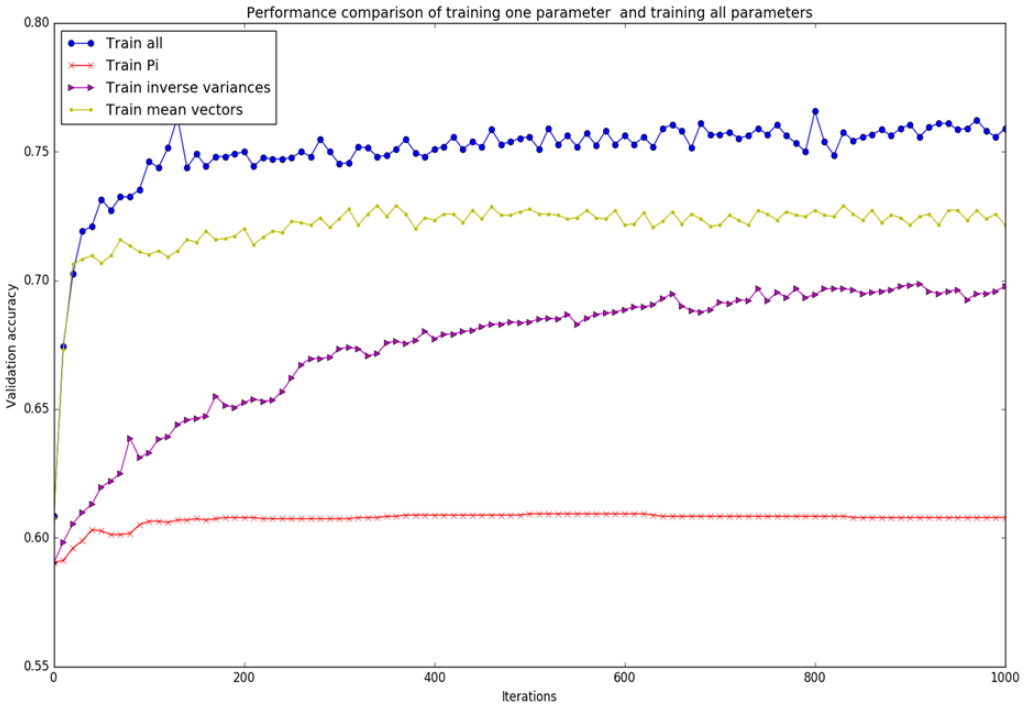

Figure 4.1: This figure shows how training all parameters together performs much better than training them individually.

In figure 4.2, we show how validation accuracy changes with number of components in each mixture model. There's an optimal value of K which gives the best classification performance. Also, the optimal value of K varies with the dataset.

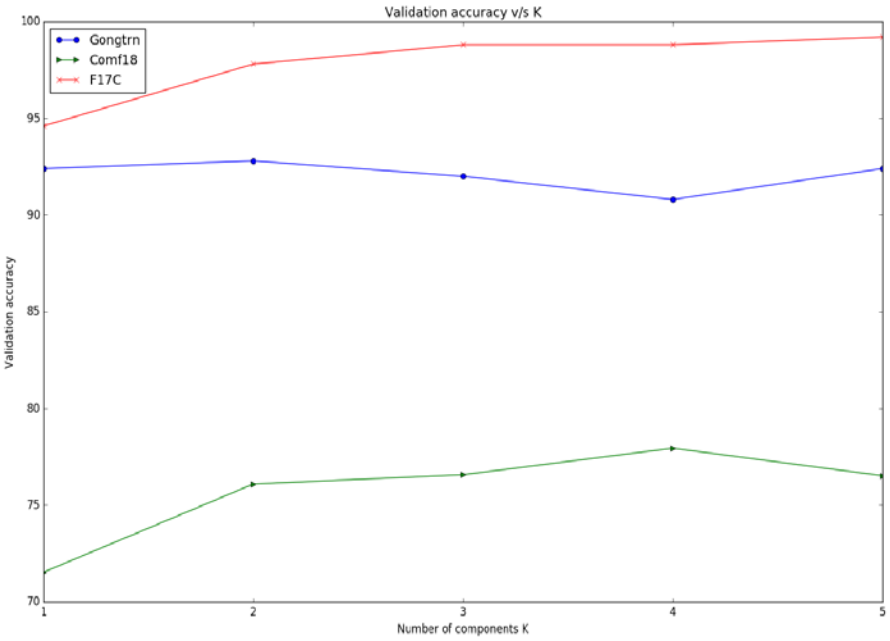

Figure 4.2. The figure shows how validation accuracy changes with K.

Chapter 5

CONCLUSIONS

In this thesis, a supervised learning approach is proposed to train Gaussian mixture model classifiers. The results show a considerable improvement in performance compared to EM. A possible reason for this could be that the final objective for EM is to find parameters that maximize the likelihood of the observed data. This is different than the objective of training classifiers which is minimizing the probability of error. Also, when using EM, each class is trained separately, so there is no interaction between the parameters of different classes. In the gradient approach, the error function ties all the parameters together. This leads to interaction between the classes which helps in improving the classification performance.

While training the parameters, we make sure that the prior probabilities add up to 1. Also, it is ensured that the inverse variances are positive after training. Thus, the resulting mixture model after training the parameters using a gradient approach is still a valid mixture model. After training, the mean vectors may not represent the actual centers of the corresponding cluster. The resulting model may not be a good estimate of the true conditional density, but it is good for classification.

Appendix A

Derivation of gradients

Here we derive the equations for gradients used in section 3.3. We need the gradient of the error function w.r.t the mean vector elements, inverse variance elements, and with the prior probabilities.

We first derive the gradient of the error function E w.r.t the conditional density of class u $f(\boldsymbol{x_p}|u)$.

The error function E and output $y_p(i)$ is given as

$$E = \sum_{p=1}^{N_v} \sum_{i=1}^{M} (t_p(i) - y_p(i))^2$$

$$y_p(i) = \frac{P(i) f(\boldsymbol{x}_p \mid i)}{\sum_{j=1}^{Nc} P(j) f(\boldsymbol{x}_p \mid j)}$$

$$\frac{\partial E}{\partial f(\boldsymbol{x}_p \mid u)} = -2 \sum_{p=1}^{N_v} \sum_{i=1}^{M} (t_p(i) - y_p(i)) \frac{\partial y_p(i)}{\partial f(\boldsymbol{x}_p \mid u)} \tag{6.1.1}$$

$$\begin{aligned} &for\ u = i \\ &\frac{\partial y_p(i)}{\partial f(\boldsymbol{x}_p \mid u)} = \frac{(\sum_{j=1}^{Nc} P_j f(\boldsymbol{x}_p \mid j)) P_u - P_u^{\,2} f(\boldsymbol{x}_p \mid u)}{(\sum_{j=1}^{Nc} P_j f(\boldsymbol{x}_p \mid j))^2} \\ &for\ u \neq i \\ &\frac{\partial y_p(i)}{\partial f(\boldsymbol{x}_p \mid u)} = \frac{-P_u f(\boldsymbol{x}_p \mid i) P_i}{(\sum_{j=1}^{Nc} P_j f(\boldsymbol{x}_p \mid j))^2} \end{aligned} \tag{6.1.2}$$

We now derive the gradient of the conditional density $f(\boldsymbol{x_p}|u)$. w.r.t the mean vector elements $m_{lu}(q)$ and inverse variances $v_{lu}(q)$. The conditional density is given as

$$f(\boldsymbol{x}_p \mid u) = \sum_{k=1}^{K} a_{ku} \frac{\exp(-\sum_{n=1}^{N} (x_p(n) - m_{ku}(n))^2 v_{ku}(n))}{(2\pi)^{N/2} \mid \sum_{ku} \mid^{1/2}}$$

The partial of conditional density of $\boldsymbol{x_p}$ for class u w.r.t mean vector element q of component l is given as

$$\frac{\partial f(\boldsymbol{x}_p \mid u)}{\partial m_{lu}(q)} = a_{lu} \frac{\exp(-\sum_{n=1}^{N} (x(n) - m_{lu}(n))^2 v_{lu}(n))}{(2\pi)^{N/2} \mid \sum_{lu} \mid^{1/2}} (2(x_p(q) - m_{lu}(q))$$

(6.1.3)

The partial of conditional density of $\boldsymbol{x_p}$ for class u w.r.t inverse variance of element q of component l is given as

$$\frac{\partial f(\boldsymbol{x}_p \mid u)}{\partial v_{lu}(q)} = a_{lu} \frac{\exp(-\sum_{n=1}^{N} (x_p(n) - m_{lu}(n))^2 v_{lu}(n))}{(2\pi)^{N/2} \mid \sum_{lu} \mid^{1/2}} (x_p(q) - m_{lu}(q))^2$$

(6.1.4)

Combining equations 6.1.1, 6.1.2, 6.1.3, 6.1.4, the gradient of the error function w.r.t the mean vector elements and inverse variances is

$$\frac{\partial E}{\partial m_{lu}(q)} = \frac{\partial E}{\partial f(\boldsymbol{x}_p \mid u)} \frac{\partial f(\boldsymbol{x}_p \mid u)}{\partial m_{lu}(q)} \tag{6.1.5}$$

$$\frac{\partial E}{\partial v_{lu}(q)} = \frac{\partial E}{\partial f(\boldsymbol{x}_p \mid u)} \frac{\partial f(\boldsymbol{x}_p \mid u)}{\partial v_{lu}(q)} \tag{6.1.6}$$

Finally, we derive the gradient of the error function w.r.t to the prior probabilities

$$\frac{\partial E}{\partial P(m)} = -2 \sum_{p=1}^{N_v} \sum_{i=1}^{M} (t_p(i) - y_p(i)) \frac{\partial y_p(i)}{\partial P(m)} \tag{6.1.7}$$

for m = i

$$\frac{\partial y_p(i)}{\partial P(m)} = \frac{(\sum_{j=1}^{Nc} P_j f(\boldsymbol{x}_p \mid j)) f(\boldsymbol{x}_p \mid m) - P(m) f(\boldsymbol{x}_p \mid m)^2}{(\sum_{j=1}^{Nc} P_j f(\boldsymbol{x}_p \mid j))^2}$$

for m $\neq$ i

$$\frac{\partial y_p(i)}{\partial P(m)} = \frac{-P_i f(\boldsymbol{x}_p \mid i) f(\boldsymbol{x}_p \mid m)}{(\sum_{j=1}^{Nc} P_j f(\boldsymbol{x}_p \mid j))^2}$$

Appendix B

Description of datasets

F17C

This file has 17 inputs, 39 classes and 4745 patterns. This data file consists of parameters that are available in the basic health usage monitoring system (HUMS), plus some others. The data was obtained from the M430 flight load level survey conducted in Mirabel Canada in early 1995. The input features include: (1) CG F/A load factor, (2) CG lateral load factor, (3) CG normal load factor, (4) pitch attitude, (5) pitch rate, (6) roll attitude, (7) roll rate, (8) yaw rate, (9) corrected airspeed, (10) rate of climb, (11) longitudinal cyclic stick position, (12) pedal position, (13) collective stick position, (14) lateral cyclic stick position, (15) main rotor mast torque, (16) main rotor mast pm, (17) density ratio. The 39 classes represent different maneuvers of the flight like taking off, landing, turning right or left etc. This is an application for prognostics or flight condition recognition

COMF18

This file has 18 inputs, 4 classes and 12392 patterns. The training data file is generated segmented images. Each segmented region is separately histogram equalized to 20 levels. Then the joint probability density of pairs of pixels separated by a given distance and a given direction is estimated. We use 0, 90, 180, 270 degrees for the directions and 1, 3, and 5 pixels for the separations. The density estimates are computed for each classification window. For each separation, the co-occurrences for the four directions are folded together to form a triangular matrix. From each of the resulting three matrices, six features are computed: angular second moment, contrast, entropy, correlation, and the sums of the main diagonal and the first off diagonal. This results in 18 features for each classification window. [28]

GONGTRN

This file has 16 inputs, 10 classes and 3000 patterns. The raw data consists of images from hand printed numerals collected from 3,000 people by the Internal Revenue Service. We randomly chose 300 characters from each class to generate 3,000-character training data. Images are 32 by 24 binary matrices. An image scaling algorithm is used to remove size variation in characters. The feature set contains 16 elements. The 10 classes correspond to 10 Arabic numerals [29]

GRNG

This file has 16 inputs, 4 classes and 800 patterns. The geometric shape recognition data file consists of four geometric shapes, ellipse, triangle, quadrilateral, and pentagon. Each shape consists of a matrix of size 64*64. For each shape, 200 training patterns were generated using different degrees of deformation. The deformations included rotation, scaling, translation, and oblique distortions. The feature set is ring-wedge energy (RNG), and has 16 features [30]

PHONEME

This file has 5 inputs, 2 classes and 5404 patterns. The aim of this dataset is to distinguish between nasal (class 0) and oral sounds (class 1). The class distribution is 3,818 samples in class 0 and 1,586 samples in class 1. The phonemes are transcribed as follows: sh as in she, dcl as in dark, iy as the vowel in she, aa as the vowel in dark, and ao as the first vowel in water. [31]

M-FEAT

This dataset consists of features of handwritten numerals (0 - 9) extracted from a collection of Dutch utility maps. 200 patterns per class (for a total of 2,000 patterns) have

been digitized in binary images. These digits are represented in terms of the following six feature sets (files): 1. mfeat-fou: 76 Fourier coefficients of the character shapes; 2. mfeat-fac: 216 profile correlations; 3. mfeat-kar: 64 Karhunen-Love coefficients; 4. mfeat-pix: 240 pixel averages in 2 x 3 windows; 5. mfeat-zer: 47 Zernike moments; 6. mfeat-mor: 6 morphological features. [32]

References

[1] Wikipedia contributors. "Statistical classification." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 18 Nov. 2016. Web. 18 Nov. 2016.

[2] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20.3 (1995): 273-297.

[3] Quinlan, J. R. (1987). "Simplifying decision trees". International Journal of Man-Machine Studies.

[4] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. No. ICS-8506. CALIFORNIA UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE, 1985.

[5] Freund, Y.; Schapire, R. E. (1999). "Large margin classification using the perceptron algorithm"

[6] Thomas Bayes, "An Essay towards solving a Problem in the Doctrine of Chances", Philosophical Transactions, 1763

[7] Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar (2012) Foundations of Machine Learning

[8] McLachlan, G., and D. Peel. Finite Mixture Models. Hoboken, NJ: John Wiley & Sons, Inc., 2000.

[9] Dempster, A.P.; Laird, N.M.; Rubin, D.B. (1977). "Maximum Likelihood from Incomplete Data via the EM Algorithm". Journal of the Royal Statistical Society

[10] Hand, D. J.; Yu, K. (2001). "Idiot's Bayes — not so stupid after all?". International Statistical Review.

[11] Rish, Irina (2001). An empirical study of the naive Bayes classifier . IJCAI Workshop on Empirical Methods in AI.

[12] Russell, Stuart; Norvig, Peter (2003) [1995]. Artificial Intelligence: A Modern Approach (2nd ed.). Prentice Hall

[13] Selim SZ, Ismail MA (1984) K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. IEEE Transactions on Pattern Analysis and Machine Intelligence 6(1):81

[14] Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2007.

[15] Deng, L. (2014). Automatic Speech Recognition- A Deep Learning Approach (pp. 6-8). Springer

[16] Santosh, D. (2013). Tracking Multiple Moving Objects Using Gaussian Mixture Model. International Journal of Soft Computing and Engineering, 3-2, 114-119.

[17] Gaussian Mixture Model. Brilliant.org. Retrieved 01:51, July 13, 2017, from https://brilliant.org/wiki/gaussian-mixture-model/

[18] Wikipedia contributors. "Gradient descent" Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia

[19] Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13

[20] Sutton, R. S. (1986). Two problems with backpropagation and other steepest-descent learning procedures for networks. Proc. 8th Annual Conf. Cognitive Science Society

[21] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12, 2121–2159.

[22] Wikipedia contributors. "Kronecker delta" Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia

[23] Wikipedia contributors. "Multivariate normal distribution" Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia

[24] R. Vedala and B. R. Kumar, "An application of Naive Bayes classification for credit scoring in e-lending platform," 2012 International Conference on Data Science & Engineering (ICDSE), Cochin, Kerala, 2012, pp. 81-84.

[25] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.

[26] George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Large vocabulary continuous speech recognition with context-dependent DBN-HMMS. In ICASSP, pages 4688–4691. IEEE, 2011. ISBN 978-1-4577- 0539-7

[27] M. N. Marsono, M. W. El-Kharashi, and F. Gebali, “Binary LNS-based naïve Bayes inference engine for spam control: Noise analysis and FPGA synthesis”, IET Computers & Digital Techniques, 2008

[28] R.R. Bailey, E. J. Pettit, R. T. Borochoff, M. T. Manry, and X. Jiang, "Automatic Recognition of USGS Land Use/Cover Categories Using Statistical and Neural Network Classifiers," Proceedings of SPIE OE/Aerospace and Remote Sensing, April 12-16, 1993, Orlando Florid

[29] W. Gong, H. C. Yau, and M. T. Manry, "Non-Gaussian Feature Analyses Using a Neural Network," Progress in Neural Networks, vol. 2, 1994, pp. 253-269.

[30] H. C. Yau, M. T. Manry, "Iterative Improvement of a Nearest Neighbor Classifier", Neural Networks, Vol. 4, pp. 517-524, 1991

[31] Openml.org “Phoneme dataset”, https://www.openml.org/d/1489

[32] M. van Breukelen, R.P.W. Duin, D.M.J. Tax, and J.E. den Hartog, Handwritten digit recognition by combined classifiers, Kybernetika, vol. 34, no. 4, 1998, 381-386.