Hybrid Generative-Discriminative Learning

João Pedro Valdeira Caetano
joao.p.v.caetano@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

July 2018

Abstract

This work presents a hybrid generative-discriminative algorithm for supervised or semi-supervised learning, inspired by radial basis function network classifiers. The network is composed of two layers: (1) a layer with mixture models that creates an embedding of the input; (2) an output layer with a log-linear model that classifies the embedded data. The mixtures may include Gaussian and/or categorical components (also called *multinoullis*), thus being able to handle real, categorical, and mixed data. It can also learn from datasets containing missing entries, where the mixture layer can estimate the missing values at each iteration. The proposed learning scheme intertwines the (unsupervised) expectation-maximization algorithm for finite mixtures (running in the first layer) with (supervised) logistic regression (running in the output layer); moreover, it allows the output layer to influence the weights of the components in the first layer (according to their importance/weight for the classifier), thus being interpretable as a form of back-propagation (without gradients). The algorithm handles semi-supervised learning and, as a by-product, performs density estimation, clustering, and dimensionality reduction of the input data. The experimental results on some classical datasets (involving real-valued, categorical, and mixed data) show competitive and promising results.

Keywords: embeddings, radial basis function networks, classification, density estimation, semi-supervised learning

1. Introduction

In this work, we propose a new learning algorithm that learns a classifier with a structure similar to a *radial basis function* network (RBF network) [4]. Our approach extends that of [9], which proposes a training algorithm for RBF networks, incorporating both generative (mixture-based) and discriminative (logistic) criteria.

Our classifier is composed of two main layers, a mixture layer that embeds the data and is able to work with real and/or categorical data, also filling missing values, and a log-linear model layer, similar to a multinomial logistic regression that classifies the embedded data. The learning algorithm gives the possibility of sharing parameters from the log-linear model layer to the mixtures layer, to be used in the upcoming fitting of the mixtures. This makes the (usually unsupervised) task of fitting mixtures partially supervised, as it is being influenced also by the supervised learning task.

Our learning algorithm can work in a supervised or semi-supervised mode, where the unlabeled data is used only in the mixture fitting part. Fully unsupervised learning is also an option of the algorithm.

As a side product of the learning algorithm, at the end of training, we have a distribution of the training data, and also its clustering. By using a smaller number of components in the mixtures than the original data dimension, the data can be embedded into a lower dimension representation, thus performing dimensionality reduction.

In summary, the main objective of the algorithm is to create an RBF network that by learning an embedding of the data that is able to adapt, not only to the density of the input data, but also in a way that improves the classification accuracy.

1.1. Feature Learning and Embeddings

Some algorithms have been proved to automatically extract new features and perform *smart* embedding during training. This is the case of deep neural networks (DNNs) [11], which have been the focus of much recent work, but also of classical techniques such as Fisher discriminant analysis [12].

1.1.1 Supervised feature learning

A visualization technique for convolutional neural networks proposed in [31] gives insight into intermediate feature layers where we can see how intermediate constructed features are far from random,

showing many intuitively desirable properties, such as compositionality, increasing invariance, and class discrimination as we proceed through the layers.

In speech recognition, [29] argues that the improved accuracy achieved by DNNs is the result of their ability to extract discriminative internal representations that are robust to the many sources of variability in speech signals. They show that these representations become increasingly insensitive to small perturbations in the input with increasing network depth, which leads to better speech recognition performance with deeper networks.

This previous examples illustrate supervised feature learning via back propagation, which means that labels are used in the feature learning process.

There are some drawbacks with DNNs. Small datasets, with only a few samples, are not enough for training, specially if they include strong nonlinearities. Also, in the case of data that is hard or expensive to label, semi-supervised learning is not an option. Finally, DNNs do not directly handle data with missing features. As explained bellow, our approach overcomes all those problems.

1.1.2 Unsupervised feature learning

Learning generative models from data and using them to create features for classification is widely used. In [6], the authors exploit the fact that MRI data is well described by Rice distributions, and use Rician mixtures as the underlying generative model, based on which several generative embeddings are built, yielding vectorial representations used for kernel-based support vector machines. Generative models are combined in [14], such as hidden Markov Models, providing a principled way of treating missing information and dealing with variable-length sequences, and discriminative models such as support vector machines, which enables to construct flexible decision boundaries and often results in superior classification performance. Other works where generative models are used to represent/embed the data and then use this representation (embedding) in a discriminative classifier are for instance [23, 25, 2, 18, 15, 3].

As shown bellow learning features in a fully unsupervised way is also an option of our proposed algorithm.

1.2. Outline

The work developed mainly builds upon two blocks: finite mixture models and log-linear models. The models needed to understand and describe our algorithm are reviewed in Section 2, where a brief overview of those topics can be found. In Section 3, we explain how the algorithm works and is structured. Also some notes about it's implementation are given.

Experiments designed to provide intuition about some aspects of the algorithm such as its ability to estimate the number of Gaussians to be used and the sensitivity of the mixture layer to *linking* both layers, are presented in Section 4. Experimental results in comparison with other algorithms on some classical datasets are also shown in Section 4.

Finally, some conclusions and directions for future work are presented in Section 6.

2. Background

2.1. Radial Basis Function Networks

Since the structure of our classifier is based on RBF networks (RBFN), we begin by briefly reviewing this class of methods. Let us start by formulating the problem of learning a RBF network. Given a set of K centers $z_1, ..., z_K \in \mathbb{R}^d$, an RBF network is a function $f : \mathbb{R}^d \to \mathbb{R}$ that takes the form

$$f(x) = \sum_{i=1}^{K} w_i h(\|x - z_i\|), \tag{1}$$

where the $w_i \in \mathbb{R}$ are called the weights, and h is a radial kernel. One of the most popular choices for this kernel is a Gaussian function, defined as $h(\|x - z\|) = K(x, z) = \exp(-\frac{\|x-z\|^2}{2t})$.

The goal of RBFN learning is to estimate a set of centers $z_1, ..., z_K$ and weights $w_1, ..., w_K$ from N given training points $\{(x_1, y_1), ..., (x_N, y_N)\}$. Given a loss function L, learning may be formulated as an optimization problem,

$$\underset{\boldsymbol{w}\in\mathbb{R}^K, \boldsymbol{z}\in\mathbb{R}^K}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N} L(f(x_i), y_i) + R(\boldsymbol{w}, \boldsymbol{z}),$$

where R is a regularization function that may or not be present.

Recently, some of the older approaches to training the RBF networks were revisited from a more modern perspective [24]. They analyze two common regularization procedures, one based on the square norm of the coefficients in the network and another one using centers obtained by k-means clustering, and provide a theoretical analysis of these methods as well as a number of experimental results, where they show some competitive experimental performance, as well as advantages over the standard kernel methods in terms of both flexibility (incorporating of unlabeled data) and computational complexity.

2.2. Mixture Models and the EM algorithm

Mixture models and the expectation maximization (EM) algorithm are important tools in statistical inference and machine learning, with a long and rich theory and countless applications [19, 21].

2.2.1 Mixture models: Generic formulation

Let $\boldsymbol{X} = [X_1, ..., X_d]^T \in \mathbb{R}^d$ be a d-dimensional random variable and $\boldsymbol{x} = [x_1, ..., x_d]^T \in \mathbb{R}^d$ one particular outcome of $\boldsymbol{X}$. We say that $\boldsymbol{X}$ follows a K-component **finite mixture distribution** if its probability density function can be written as [1]

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k p(\boldsymbol{x}|\boldsymbol{\theta}_k), \tag{2}$$

where $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_K, \pi_1, ..., \pi_K\}$ is the complete set of parameters. Each $\boldsymbol{\theta}_m$ is the set of parameters defining the mth component and $\pi_1, ..., \pi_K$ are the **mixing weights**, which satisfy $\pi_k \geq 0$, for $k = 1, ..., K$, together with $\sum_{k=1}^{K} \pi_k = 1$, thus $p(\boldsymbol{x}|\boldsymbol{\theta})$ is a **convex combination** of the component's distributions. The π_k can be interpreted as the component probabilities.

2.2.2 Hybrid mixtures of Gaussians and multinoullis

Data containing both real and categorical components, i.e. $\boldsymbol{x} = (\boldsymbol{x}^r, \boldsymbol{x}^c)$ where $\boldsymbol{x}^r \in \mathbb{R}^{d_r}$ and $\boldsymbol{x}^c \in \{0,1\}^{C_1} \times ... \times \{0,1\}^{C_{d_c}}$, can be modeled by a mixture of Gaussians and multinoullis with the following form:

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}^r|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \prod_{j=1}^{d_c} \mathrm{Mu}(\boldsymbol{x}_j^c|\boldsymbol{\theta}_k), \tag{3}$$

where

$$\mathcal{N}(\boldsymbol{x}^r|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{\exp\left(-\frac{1}{2}(\boldsymbol{x}^r - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}^r - \boldsymbol{\mu})\right)}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}}, \tag{4}$$

and

$$p(\boldsymbol{x}^c|\boldsymbol{\theta}) = \prod_{j=1}^{d_c} \mathrm{Mu}(\boldsymbol{x}_j^c|\boldsymbol{\theta}) = \prod_{j=1}^{d_c} \prod_{l=1}^{C_j} \theta_{jl}^{x_{jl}}, \tag{5}$$

with $\boldsymbol{x_j}$ denoting the one-hot representation of component j, that is, $x_{jl} = 1$ if and only if $\boldsymbol{x_j}$ is in category l. We admit that, by knowing $\boldsymbol{z}$, which component of the mixture is responsible for a particular data sample, there is (conditional) independence between the real and categorical part of the data.

2.2.3 EM algorithm for hybrid mixtures

EM (see [21]), is an iterative algorithm with often closed-form updates at each step, which automatically enforces the required constraints. It exploits the fact that if the data were fully observed, the ML estimate could be easily computed. It can be seen as alternating between inferring the missing values, given the current parameter estimates (**expectation**, or **E step**) and then optimizing the parameters given the filled-in log-likelihood function (**maximization**, or **M step**).

As is standard in EM for mixtures, the E-step is

$$r_{ki} = \frac{\pi_k p(\boldsymbol{x}_i|\boldsymbol{\theta}_k^{t-1})}{\sum_j \pi_j p(\boldsymbol{x}_i|\boldsymbol{\theta}_j^{t-1})} \tag{6}$$

Regarding the M step, the MAP estimate of $\boldsymbol{\pi}$, with a Dirichlet prior [21], is

$$\pi_k = \frac{r_k + \alpha_k - 1}{N + \sum_k \alpha_k - K}. \tag{7}$$

With $\alpha_k = 1$, the prior is **uniform** and the MAP estimate reduces to the ML estimate. The Gaussian parameters estimates are

$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} r_{ki} \boldsymbol{x}_i}{r_k}, \tag{8}$$

and,

$$\boldsymbol{\Sigma}_k = \frac{\sum_{i=1}^{N} r_{ki} (\boldsymbol{x}_i - \boldsymbol{\mu}_k)(\boldsymbol{x}_i - \boldsymbol{\mu}_k)^T}{r_k}. \tag{9}$$

Finally, the MAP estimate of the multinoulli parameters, again with a Dirichlet prior, is

$$\theta_{kdc} = \frac{N_{kdc} + \alpha_{kdc} - 1}{\sum_{c=1}^{C_d} N_{kdc} + \alpha_{kdc} - 1}. \tag{10}$$

2.2.4 EM for missing observations

Suppose that we have "holes in our data matrix, due to missing observations. Let $O_{ij} = 1$ if component j of sample i is observed, and let $O_{ij} = 0$, otherwise. Let $\boldsymbol{X}_v = \{x_{ij} : O_{ij} = 1\}$ be the visible data, and $\boldsymbol{X}_h = \{x_{ij} : O_{ij} = 0\}$ be the missing or hidden data. Following [8], the expected values of the missing data for the k-th Gaussian are

$$\tilde{\boldsymbol{\mu}}_{hik} = \boldsymbol{\mu}_{hk} + \boldsymbol{\Sigma}_{hvk} (\boldsymbol{\Sigma}_{vk})^{-1} (\boldsymbol{x}_{vi} - \boldsymbol{\mu}_{vi}), \tag{11}$$

where $\boldsymbol{\mu}_{hk}$ is the mean of the k-th Gaussian component for the hidden values indexes, $\boldsymbol{\Sigma}_{hvk}$ the cross-covariance matrix of the hidden and visible variables, $\boldsymbol{\Sigma}_{vk}$ the covariance matrix of the visible variables, $\boldsymbol{x}_{vi}$ the visible variables of sample i and $\boldsymbol{\mu}_{vi}$ the visible part of the Gaussian component mean.

The inputed missing values are weighted by the weights of each mixture component:

$$\tilde{\boldsymbol{x}}_{hi} = \sum_{k=1}^{K} \pi_k \tilde{\boldsymbol{\mu}}_{hik}. \tag{12}$$

[1] Subscript notation as in $p_{\boldsymbol{X}}(\boldsymbol{x})$ will be dropped and replaced with $p(\boldsymbol{x})$, unless strictly needed.

For the multinoulli part assuming that the data samples $\boldsymbol{x}$ are one-hot encoded, the expected value of the missing variables is equal to the probabilities of them being 1, which are the multinoulli parameters $\boldsymbol{\theta}$. The components estimates are then weighted by the their weights $\boldsymbol{\pi}_k$. Thus,

$$\tilde{\boldsymbol{x}}_{hi} = \sum_{k=1}^{K} \pi_k \boldsymbol{\theta}_{hik}. \tag{13}$$

2.3. Log-Linear Models

In this subsection, we introduce a family of classifiers known as log-linear models. A classifier is a mathematical function that maps input data $\boldsymbol{x} \in \mathbb{R}^d$ to a category $y \in \{1, ..., C\}$. Log-linear models are general classes of probabilistic discriminative classifiers. A log-linear model has the form

$$p(y = c|\boldsymbol{x}; \boldsymbol{\omega}) = \frac{\exp(\boldsymbol{w}_c^T f(\boldsymbol{x}, y = c))}{\sum_{c'=1}^{C} \exp(\boldsymbol{w}_{c'}^T f(\boldsymbol{x}, y = c'))}, \tag{14}$$

where $\boldsymbol{\omega}$ is a ordered tuple containing the parameter weights for each class, $\boldsymbol{\omega} = (\boldsymbol{w}_1, ..., \boldsymbol{w}_C)$ where $\boldsymbol{w}_c \in \mathbb{R}^{p_c}$. Expression 14 should be read as the probability of $y = c$, conditioned on the feature values $\boldsymbol{x}$, under parameter weights $\boldsymbol{\omega}$. The quantity $\boldsymbol{w}_c^T f(\boldsymbol{x}, y = c)$ is the inner product between $\boldsymbol{w}_c \in \mathbb{R}^{p_c}$ and $f : \mathbb{R}^d \times \{1, ..., C\} \to \mathbb{R}^{p_c}$. The last component of $f(\cdot, y = c)$ is always 1, thus the last component of $\boldsymbol{w}_c$ is the bias term of class c.

In the context of this work, the function f embeds the feature vectors in a pre-established way for each class. Let $f(\boldsymbol{x}, y = c) = \boldsymbol{x}_c$ where, for each class c, there is a corresponding parameter vector $\boldsymbol{w}_c$, which is responsible for its classification.

We now introduce some notation. Let $\eta_{ic} = p(y_i = c|\boldsymbol{x}_i; \boldsymbol{\omega})$ and $y_{ic} = \mathbb{I}(y_i = c)$ be the one-hot encoding of y_i, where the c-th bit is 1 if and only if $y_i = c$. With this notation the negative log-likelihood can be written as

$$\begin{aligned} l(\boldsymbol{\omega}) &= -\log \prod_{i=1}^{N} \prod_{c=1}^{C} \eta_{ic}^{y_{ic}} \\ &= -\sum_{i=1}^{N} \left[\left(\sum_{c=1}^{C} y_{ic} \boldsymbol{w}_c^T \boldsymbol{x}_{ic} \right) - \log \sum_{c'=1}^{C} \exp(\boldsymbol{w}_{c'}^T \boldsymbol{x}_{ic'}) \right]. \end{aligned}$$

2.3.1 Elastic log-linear model

It is common to add regularization to the cost function, since it is well known that regularization tends to improve the generalization performance of the classifier, and because some types of regularizers can also lead to variable selection [12]. An elastic net regularizer combines ℓ_1 and ℓ_2 regularization, leading to the cost function:

$$l_{elastic}(\boldsymbol{\omega}) = l(\boldsymbol{\omega}) + \lambda_1 \|\boldsymbol{\omega}\|_1 + \lambda_2 \|\boldsymbol{\omega}\|_2^2, \tag{15}$$

where $\|\boldsymbol{\omega}\|_1 = \sum_i \sum_j |\boldsymbol{w}_{ij}|$ and $\|\boldsymbol{\omega}\|_2 = \sum_i \sum_j \boldsymbol{w}_{ij}^2$. The ℓ_1 terms originates from a Laplacian prior and promotes sparsity in $\boldsymbol{\omega}$, while the the ℓ_2 term corresponds to a Gaussian prior and tends to shrink the magnitude of the parameters in $\boldsymbol{\omega}$.

2.3.2 Parameter estimation

To learn the parameter vector $\boldsymbol{\omega}$, we can use the maximum likelihood criterion, though it has no closed form, so we need to use an optimization algorithm to obtain it. For that, we usually require the gradient of the NLL. Though, the gradient of this new function is not well defined because the ℓ_1 norm is not differentiable at 0. To overcome this problem, one can reformulate the log-likelihood cost or modify the optimization algorithm to deal with it. Some of this modified optimization algorithms were analyzed and compared in [30].

In this work we chose to reformulate the optimization problem as done in [10]. Introducing two new variables, $\boldsymbol{\omega}_+$ and $\boldsymbol{\omega}_-$, which have the following properties:

$$\begin{aligned} \boldsymbol{\omega}_+ &= (\boldsymbol{w}_{+1}, ..., \boldsymbol{w}_{+C}), \\ \boldsymbol{\omega}_- &= (\boldsymbol{w}_{-1}, ..., \boldsymbol{w}_{-C}), \end{aligned} \tag{16}$$

$$\boldsymbol{\omega}_+ \geq 0,\ \boldsymbol{\omega}_- \geq 0,\ \boldsymbol{\omega} = \boldsymbol{\omega}_+ - \boldsymbol{\omega}_-. \tag{17}$$

The optimization problem is therefore reformulated as

$$\begin{aligned} \min_{\boldsymbol{\omega}} \quad & l(\boldsymbol{\omega}_+ - \boldsymbol{\omega}_-) + \lambda_1 \sum_{c=1}^{C} \|\boldsymbol{w}_{+c}\|_1 + \lambda_1 \sum_{c=1}^{C} \|\boldsymbol{w}_{-c}\|_1 \\ & + \frac{\lambda_2}{2} \sum_{c=1}^{C} \|\boldsymbol{w}_{+c}\|_2^2 + \frac{\lambda_2}{2} \sum_{c=1}^{C} \|\boldsymbol{w}_{-c}\|_2^2 \\ \text{s. t.} \quad & \boldsymbol{\omega}_+ \geq 0,\ \boldsymbol{\omega}_- \geq 0. \end{aligned} \tag{18}$$

This is a convex optimization problem with bound constraints. The gradient of the objective function assumes a similar form:

$$\frac{\partial l(\boldsymbol{\omega})}{\partial \boldsymbol{w}_{\pm c}} = \pm \sum_{i=1}^{N} \left(\eta_{ic} - y_{ic} \right) \boldsymbol{x}_{ic} + \lambda_1 + \lambda_2 \boldsymbol{w}_{\pm c} \tag{19}$$

To solve this optimization problem, we use the L-BFGS-B algorithm proposed in [5], based on the limited-memory BFGS (L-BFGS) [16]. The L-BFGS-B algorithm extends L-BFGS to handle simple box constraints, such as those in 18.

3. Proposed Algorithm and Implementation

3.1. Classifier Structure

In this work, we propose a probabilistic model for classification, i.e., a prediction of class posterior probability $p(y|\boldsymbol{x})$, for all classes $y \in \{1, ..., C\}$, where $\boldsymbol{x} = (\boldsymbol{x}^r, \boldsymbol{x}^c)$ is the input vector that can be composed by real data $\boldsymbol{x}^r \in \mathbb{R}^{d_r}$ and categorical data $\boldsymbol{x}^c \in \{0,1\}^{C_1} \times ... \times \{0,1\}^{C_{d_c}}$, where C_j is the number of categories in the j-th component of $\boldsymbol{x}^c$. The structure of the classifier is shown in Figure 1:

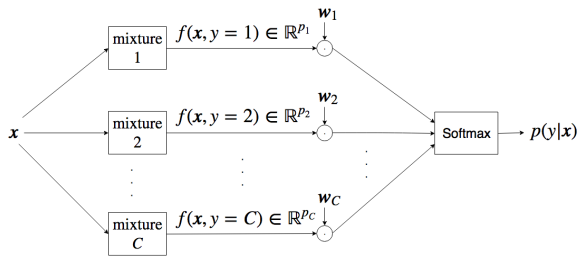

Figure 1: Structure of the proposed classifier.

The corresponding mathematical expression of this scheme is in equation 14.

We now explain how the mixtures are used to create the class-specific embeddings. The data sample $\boldsymbol{x}$ enters each of the C different mixtures (C is the number of classes in the classification problem), which create a different embedding each. The reasoning is to have a different embedding for each class. In the case of $\boldsymbol{x}$ having missing values they are inputed as explained in subsection 2.2.4. If the data samples are real, the mixtures are purely Gaussian, if they are categorical, the mixtures are purely multinoulli, and if they are an hybrid of real and categorical, the mixtures are an hybrid of Gaussians and mutinoullis (subsection 2.2.2). The embedding created by mixture c, that is $f(\boldsymbol{x}, y = c)$, is some characteristic that each of its components extracts from $\boldsymbol{x}$, except for the last position which is always 1, to account for the coming bias weight. Function f maps $\boldsymbol{x}$ to $\mathbb{R}^{p_c}$, where p_c is the number of components in the mixture of class c, plus 1. There can be many different characteristics to be taken by each component from $\boldsymbol{x}$, tough we found that posterior probability is what usually leads to the best empirical performance. The j-th component of $f(\boldsymbol{x}, y = c)$ is thus given by: $\big(f(\boldsymbol{x}, y = c)\big)_{p_c} = 1$; for $j = 1, ..., p_c - 1$,

$$\big(f(\boldsymbol{x}, y = c)\big)_j = \frac{\mathcal{N}(\boldsymbol{x}^r|\boldsymbol{\mu}_{cj}, \boldsymbol{\Sigma}_{cj}) \prod_{l=1}^{d_c} \mathrm{Mu}(\boldsymbol{x}_l^c|\boldsymbol{\theta}_{cj})\pi_{cj}}{\sum_{i=1}^{p_c-1} \mathcal{N}(\boldsymbol{x}^r|\boldsymbol{\mu}_{ci}, \boldsymbol{\Sigma}_{ci}) \prod_{l=1}^{d_c} \mathrm{Mu}(\boldsymbol{x}_l^c|\boldsymbol{\theta}_{ci})\pi_{ci}}, \tag{20}$$

where $\boldsymbol{\mu}_{cj}, \boldsymbol{\Sigma}_{cj}, \boldsymbol{\theta}_{cj}, \pi_{cj}$ are the parameters of the j-th component of the mixture of class c. The circles with a dot in the scheme at Figure 1 represent an inner product operation. Finally, the results $\boldsymbol{w}_c^T f(\boldsymbol{x}, y = c)$ enter a softmax block, which exponentiates its entries and then normalizes them, expressed by equation 14.

3.2. Classifier Training

Our classifier can be seen as a two-layer network: an input layer that creates the mixture-based embedding, and an output layer with a log-linear model that classifies the embedded data. We propose an approach where this network can be trained in a hybrid generative/discriminative way: the training of the log-linear model at the output layer influences the fitting of the mixtures in the input layer, thus effectively *linking* the layers. Optionally, the *link* between the layers can be cut making the mixtures layer work in an unsupervised way.

An iteration in our learning algorithm starts with the imputation of the missing values, if there are any (subsection 2.2.4). Then, a step of the EM algorithm (subsection 2.2.3) that fits the mixtures to the whole data, even the unlabeled samples. The log-sum-exp trick [21], is used so that the numerical errors are minimized.

The EM algorithm uses a MAP estimate for the calculation of the component weights $\boldsymbol{\pi}$ (see equation 7). The priors used, which are the αs in equation 7, are calculated with help of the log-linear model weights $\boldsymbol{\omega}$. First, we exclude the biases of the log-linear model weights and then we take their absolute value and normalize them by class, multiplying them by a constant γ, that we call *link* which defines the importance this prior has in the calculation of the mixture weights.

The prior (α) of mixture c and component i is therefore calculated as:

$$\alpha_{ci} = \gamma \frac{|w_{ci}|}{\sum_{i'=1}^{K_c} |w_{ci'}|} + 1, \tag{21}$$

where K_c is the number of components of the mixture associated with class c, and $\gamma \geq 0$ is a parameter that controls the intensity of the link between the output and the input layers. The addition of 1 guarantees that, if $\gamma = 0$ (no link), the MAP estimate falls down to the ML estimate. In the first iteration, since no log-linear weights have been calculated, the ML estimate is used.

The reasoning is that if a weight is large in magnitude, the corresponding feature is important in the classification process, thus the mixture component responsible for that feature should be given more importance. The idea is to give more weight to the components that are creating features that are important in the classification process, so that the mixture produces a better embedding for the classification of that class. Because each mixture uses

the weights of a specific class, it tends to *specialize* in embedding data from that class.

After the EM step, each mixture embeds the labeled data which is used by the log-linear model for training. We use an elastic log-linear model with both ℓ_1 and ℓ_2 regularization, trained as explained in subsection 2.3.2. Every iteration of the algorithm is initialized with the parameters achieved in the last iteration (warm starting).

One additional mode of regularization can be added, which works together with the sparsity imposed by the ℓ_1 regularizer: when a weight of the log-linear model is set to 0 in training, the mixture component responsible for creating the feature associated with that weight is erased from the mixture, thus decreasing the complexity of the classifier.

The high-level pseudo-code of our RBF network training is shown in Algorithm 1. What is referred to as algorithm specific parameters are the following: γ, $K_1, ..., K_C$, λ_1, λ_2, the type of covariance used for the GMM. The algorithm is stopped when the relative change of the mixtures' parameters falls below some threshold.

Input: Training data (X), labels (y) and algorithm specific parameters
Output: RBF network fitted object
initialization of mixtures and log-linear models;
$\alpha_{ci} = 1$, for all c and i;
while *stopping criterium not met* **do**
    new_dataset = [ ];
    for *each class* **do**
        mixture = mixture.EM_iteration(X, $\boldsymbol{\alpha}$);
        embedded = mixture.embed_data(X);
        new_dataset.concatenate(embedded)
    end
    log-linear_model = log-linear_model.fit(new_dataset);
    $\boldsymbol{\alpha}$ = calculate_new_priors(log-linear_model.weights)
end

Algorithm 1: RBF network training algorithm.

3.3. Implementation

A Python package implementing the algorithm is available at `github.com/jpvcaetano/Radial-Basis-Function-Networks`. The code follows the scikit-learn library structure (see `scikit-learn.org/stable/developers`). The scikit-learn library includes a function designed to test if a classifier is compatible with its framework. If the package passes this test, it is ready to work in combination with all the scikit-learn tools. Our package is compatible with the scikit-learn library.

4. Illustrative Experiments

4.1. Effect of the Link Parameter

We begin with a toy experiment to provide intuition on the role of the link parameter (Section 3.2). We use the first two dimensions of the classical dataset Iris. Because this dataset has 3 classes, 3 mixtures are created, each one embedding the data for each class. We compare how the mixtures are fitted, for different values of γ ($\gamma = 0$ in Fig. 2 and $\gamma = 10^{10}$ in Fig. 3). Notice that for $\gamma = 0$, no parameters were shared from the log-linear model and the three mixtures converge to the same final configuration. With a very high link parameter, the mixture weights estimates are overloaded by the log-linear parameters and the responsibilities of the points are ignored.

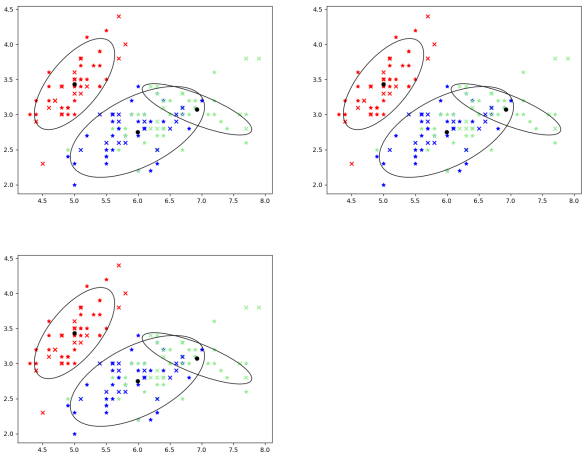

Figure 2: Mixtures of the red, blue and green class respectively, $\gamma = 0$

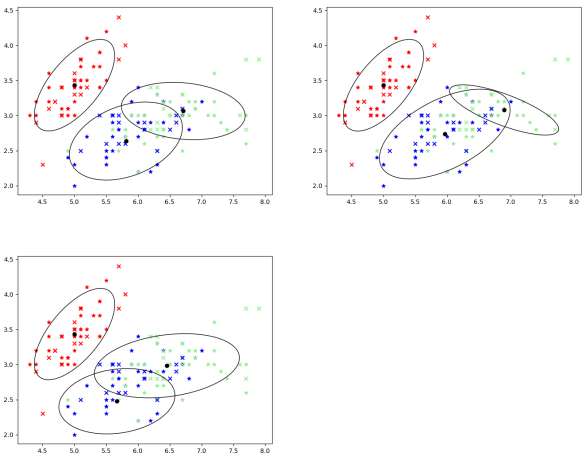

Figure 3: Mixtures of the red, blue, and green classes respectively, $\gamma = 10^{10}$

In a second example, we consider a two-class synthetic dataset with three Gaussians, to further illustrate the effect of the link. For $\gamma = 0$, the mixtures and decision boundaries are those shown in Fig. 4. Obviously, this is not the best mixture for the classi-

fication task since the decision boundary is far from optimal (accuracy 91.1%). For $\gamma = 10^4$, the mixtures and decision boundaries converge to those in Fig. 5, with accuracy of 98.8%.

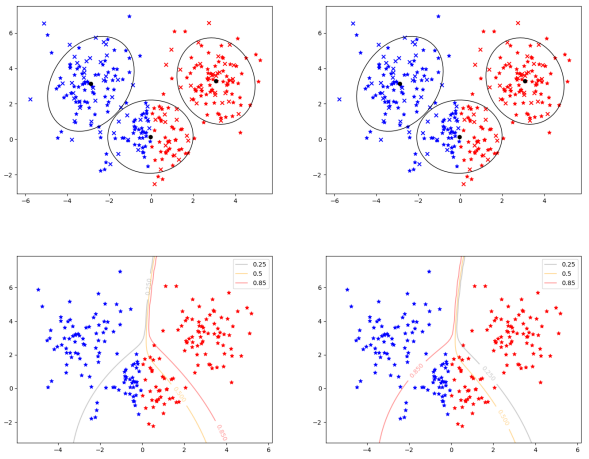

Figure 4: Mixtures and decision boundaries of the red and blue classes respectively, $\gamma = 0$

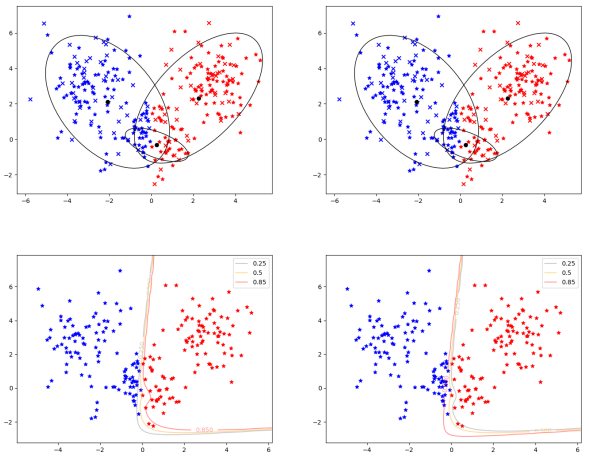

Figure 5: Mixtures and decision boundaries of the red and blue classes respectively, $\gamma = 10^4$

With $\gamma = 0$, the features extracted by the bottom Gaussians do not help in the classification task. The estimated left mixture weights are very similar $(0.372, 0.323, 0.305)$. For $\gamma = 10^4$, because the features associated to this Gaussians are not relevant (thus its corresponding linear-model weights are are small in magnitude) its weights are forced to become small. The final weigh estimates for the left mixture are $(0.037, 0.494, 0.469)$, where the smallest weight corresponds to the bottom Gaussian.

4.2. Component Annihilation

We now show how the ℓ_1 weight (λ_1, see equation 15) affects the final number of components (doing model selection), when using component annihilation as described in subsection 3.2. We fit our estimator to a synthetic dataset with two classes, each sampled from a Gaussian. We start with 6 components in the mixture and vary λ_1 to see how the model can learn the correct number of Gaussians. In Fig. 6, we observe that for $\lambda_1 = 0.001$, after fitting 6 Gaussians to the data, they all persist in the end, while for $\lambda_1 = 0.1$ the algorithm eliminates 4 components, thus yielding the correct distribution model. Component annihilation can therefore lead to better density estimates of the input data, and naturally speeds up the algorithm through the iterations, because the algorithm deals with fewer components.

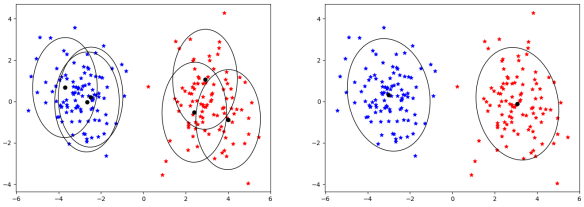

Figure 6: Mixture components after fitting a synthetic dataset for $\gamma_1 = 0.001$ and $\gamma_1 = 0.1$.

4.3. Visualization of the Embeddings

To illustrate the effect of the embedding, Fig. 7 shows the Iris, Wine, and Leukemia datasets, before and after the embedding, and mapped into a 2-dimensional space by t-SNE (*t-distributed stochastic neighbor embedding* [27]). This technique maps data into lower dimensions while trying to preserve its original neighborhood structure, and is widely used for visualization. Clearly, after the embeddings, the classes are easier to separate with some decision boundaries.

5. Experiments on Benchmark Datasets

5.1. Tuning the hyper-parameters

The hyper-parameters in our method that need to be adjusted are: γ, $K_1, ..., K_C$, λ_1, λ_2, and the type of covariance used for the GMM. We use the randomized search method from scikit-learn, which works as follows. First, we define the number of parameter configurations to be tested. Then, for each configuration, a K-fold cross validation scheme is used to assess the quality of that configuration.

5.2. Real datasets

We compare our results with those reported in [13], obtained with the following algorithms: k-nearest neighbors (KNN), linear regression for classification (LRC), regularized orthogonal least squares algorithm for RBF networks (ROLS) [7], RBF network from Netlab toolbox (RBFNnl) [22], normalized radial basis function network (nRBFN) [20], nRBFN without regularization, using the same basis as nRBFN (nRNwr), nRBFN with basis chosen

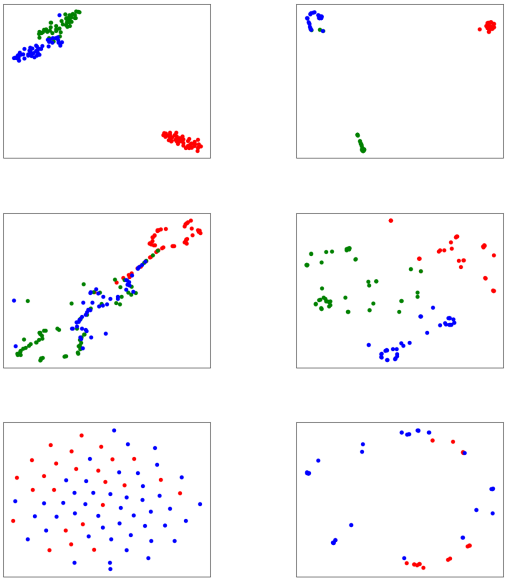

Figure 7: Original (left) and embedded (right) Iris (top), Wine (middle), and Leukemia (bottom) datasets mapped to 2-dimension with T-SNE.

randomly from the training set and basis size following nRBFN (nRNrb), and support vector machine (SVM). We also a trained a standard multinomial logistic regression model.

We use classical small datasets of the UCI Machine Learning Repository [1] (Iris, WDBC, Glass, Sonar, Wine) and high-dimensional and small-sample-size gene data (Colon, Leukemia). For each dataset, we perform 10 tests, each with a different seed. Table 1 shows the results.

The proposed method achieves the best results in the Iris and the Leukemia datasets, and shows competitive results in the rest except in the Wdbc, Glass, and the Sonar where performs poorly.

5.3. Semi-supervised learning

In this section, we illustrate the ability of the proposed algorithm to operate in semi-supervised mode. We use the same hyper-parameter configurations obtained for the same datasets in fully supervised mode, although some points are unlabeled and therefore are only used for the mixture estimation, not for the log-linear model fitting. Some percentage of the dataset is unlabeled in a random way and the results are compared for different percentages of labeled data.

(Note: missing results correspond to cases where there were not enough classes in the training data for training.)

Table 2: Mean test error (%) and standard deviation of our method with different percentages of labeled data for different datasets.

Dataset	85%	50%	25%
Iris	3.2(±1.8)	3.2(±1.4)	7.2(±7.7)
wbdc	7.2(±1.4)	7.6(±1.6)	8.8(±1.8)
Glass	57.5(±2.6)	57.7(±3.4)	57.9(±3.5)
Sonar	32.4(±8.2)	35.7(±6.6)	40.2(±5.5)
Wine	10.1(±9)	21.0(±14.7)	36.4(±22.8)
Colon	20.6(±9.9)	23.9(±7.5)	29.4(±14.5)
Leukemia	10.9(±6.9)	12.6(±10.6)	14.4(±12.2)

The results show that the method is able to handle missing labels, with a slow degradation of the accuracy, as is naturally expectable.

5.3.1 Categorical and mixed real and categorical dataset

Here we test our method on the Mushroom dataset (purely categorical) and on the Census income dataset (mixed real and categorical), which have missing data (both).

In [17] the authors have achieved 0% error on the Mushroom dataset and Steves [2], who tested several algorithms on the Census income dataset, and obtained 14, 1% error with the best performing one.

6. Conclusions

In this final chapter, we discuss some of the achievements of our work and mention future research directions that it could stimulate.

6.1. Achievements

We have developed an algorithm capable of learning a multi-class classifier, which is able to handle missing values and missing labels, as well as real valued, categorical, or mixed data.

We show in Section 4.1 how sharing the parameters between the layers can be advantageous in finding a mixture that better produces an embedding for the classification process. Another confirmation that the embeddings work was given in Section 4.3, where we show that after the application of the embedding, the data is better separated in the embedding space than in the original one.

In Chapter 4.2, we saw that estimating the number of Gaussians with the log-linear models weights also works well as a regularization technique.

In the tested datasets our algorithm performs well, showing competitive and promising results on some real-valued datasets, and also in the categorical and mixed ones, even with missing data.

[2] knowm.org/census-income-classification-benchmark/

Table 1: Mean test error (%) and standard deviation of our estimator, compared with some algorithms reported in [13] on real dataset.

Dataset	Proposed	logistic	KNN	LRC	ROLS	RBFNnl	nRNwr	nRNrb	SVM	nRBFN
Iris	**3.3(±1.8)**	7.9(±0.0)	5.3	18.7	4.0	5.3	6.7	5.3	5.3	5.3
Wdbc	7.6(±1.4)	4.9(±0.1)	6.3	6.3	8.5	4.9	7.7	5.6	**4.6**	4.9
Glass	56.8(±2.6)	45.8(±0.0)	39.0	43.8	36.2	67.6	50.5	36.2	**31.4**	38.1
Sonar	33.6(±6.1)	25.2(±0.0)	33	23.3	**18.4**	22.3	21.4	**18.4**	**18.4**	**18.4**
Wine	7.1(±4.3)	7.3(±0.9)	33.0	3.4	**1.1**	5.7	6.8	**1.1**	2.3	**1.1**
Colon	20.9(±7.5)	25.2(±4.9)	35.5	**16.1**	**16.1**	**16.1**	**16.1**	**16.1**	**16.1**	**16.1**
Leuk.	**10.9(±6.5)**	32.9(±3.5)	41.2	17.6	20.6	17.6	20.6	20.6	23.5	20.6

Table 3: Mean test error (%) and standard deviation of our method for different percentages of labeled data on the Mushroom dataset.

Dataset	100%	85%	50%	25%	5%
Mushroom	0.8(±0.6)	0.9(±0.7)	1.5(±0.5)	2.3(±1.4)	6.8(±3.2)
Census Income	18.6(±0.7)	18.2(±1.0)	18.7(±1.1)	18.9(±1.0)	18.8(±1.1)

The semi-supervised learning experiments also showed promising results, showing that the accuracy degrades slowly with the decrease of the number of labeled points.

Finally, we implemented this method in a publicly available package, compatible with scikit-learn.

6.2. Future Work

The extension of this method for regression problems, by using only one mixture, instead of one mixture per class (since in this case there are no classes), would be the next natural step for future work. The log-linear model would be replaced by linear regression and the algorithm would be able to perform RBF-type non-linear regression, with simultaneous learning of the basis functions.

An interesting research direction would be to add more layers, so the embeddings could be richer; for example, this may be achieved by using deep Gaussian mixture models [26, 28], rather than standard Gaussian mixtures. Also, a more complex classification algorithm in the last layer, such as a neural network, can be considered.

Finally, another research path is to devise other means by which the output layer may influence the estimates of the parameters of the mixtures in the input layer. In addition to influencing the component weights via a Dirichlet prior, as proposed in this work, other parameter estimates, such as the covariance matrices and means of the Gaussians, could also be influenced, but it is not yet clear how that should be done.

Acknowledgements

I would like to thank Professor Mário Figueiredo for the opportunity of working with him, for everything he taught me, and the help provided to conclude this work.

References

[1] UCI Machine Learning Repository.

[2] K. Aida-zade, A. Xocayev, and S. Rustamov. Speech recognition using support vector machines. In *2016 IEEE 10th International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–4, 2016.

[3] M. Bicego, C. Acosta-Munoz, and M. Orozco-Alzate. Classification of seismic volcanic signals using hidden-markov-model-based generative embeddings. *IEEE Transactions on Geoscience and Remote Sensing*, 2013.

[4] D. S. Broomhead and D. Lowe. Multivariable Functional Interpolation and Adaptive Networks. *Complex Systems*, 2:321–355, 1988.

[5] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, 1995.

[6] A. C. Carli, M. A. T. Figueiredo, M. Bicego, and V. Murino. Generative embeddings based on Rician mixtures for kernel-based classification of magnetic resonance images. *Neurocomputing*, 2014.

[7] S. Chen, E. S. Chng, and K. Alkadhimi. Regularized orthogonal least squares algorithm for constructing radial basis function networks. *International Journal of Control*, 64:829–837, 1996.

[8] E. Eirola, A. Lendasse, V. Vandewalle, and C. Biernacki. Mixture of Gaussians for distance estimation with missing data. *Neurocomputing*, 131:32–42, 2014.

[9] A. J. Ferreira and M. A. T. Figueiredo. Hybrid generative/discriminative training of radial basis function networks. In *ESANN*, pages 599–604, 2006.

[10] M. A. T. Figueiredo, R. D. Nowak, and S. J. Wright. Gradient Projection for Sparse Reconstruction: Application to Compressed Sensing and Other Inverse Problems. *IEEE Jornal of selected topics in Signal Processing*, 2007.

[11] I. J. Goodfellow, Q. V. Le, A. M. Saxe, H. Lee, and A. Y. Ng. Measuring Invariances in Deep Networks. *Advances in Neural Information Processing Systems 22*, 22:646–654, 2009.

[12] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[13] Z. Hu, G. Pan, and Z. Wu. Spectral-graph Based Classifications: Linear Regression for Classification and Normalized Radial Basis Function Network. *ArXiv e-prints*, 2017.

[14] T. S. Jaakkola and D. Haussler. Exploiting generative models discriminative classifiers. *Advances in Neural Information Processing Systems*, 11, 1999.

[15] M. Layton. Maximum margin training of generative kernels. Technical report, Engineering Department, Cambridge University, 2004.

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[17] J. Li, G. Dong, K. Ramamohanarao, and L. Wong. Deeps: A new instance-based discovery and classification system. *Machine Learning*, 2001.

[18] X. Li, T. S. Lee, and Y. Liu. Hybrid generative-discriminative classification using posterior divergence. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2011.

[19] G. McLachlan and D. Peel. *Finite Mixture Models*. Wiley, 2000.

[20] J. Moody and C. J. Darken. Fast Learning in Networks of Locally-Tuned Processing Units. *Neural Computation*, 1(2):281–294, jun 1989.

[21] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[22] I. T. Nabney. *NETLAB Algorithms for Pattern Recognition*. Springer, 2003.

[23] A. Perina, M. Cristani, U. Castellani, V. Murino, and N. Jojic. Free energy score space. In *Advances in Neural Information Processing Systems 22*, pages 1428–1436. Curran Associates, Inc., 2009.

[24] Q. Que and M. Belkin. Back to the future: Radial basis function networks revisited. In A. Gretton and C. C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 1375–1383. PMLR, 2016.

[25] K. Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K. Müller. A new discriminative kernel from probabilistic models. *Neural Computation*, 14:2397–2414, 2002.

[26] A. Van Den Oord and B. Schrauwen. Factoring Variations in Natural Images with Deep Gaussian Mixture Models. *Advances in Neural Information Processing Systems*, 27, 2014.

[27] L. Van Der Maaten and G. Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[28] C. Viroli and G. J. McLachlan. Deep Gaussian Mixture Models. *ArXiv e-prints*, Nov. 2017.

[29] D. Yu, M. L. Seltzer, J. Li, J.-T. Huang, and F. Seide. Feature Learning in Deep Neural Networks - Studies on Speech Recognition Tasks. *ArXiv e-prints*, Jan. 2013.

[30] G. Yuan, C. Ho, and C. Lin. An improved GLMNET for l1-regularized logistic regression. *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '11*, 13:33, 2011.

[31] M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. In *Computer Vision – ECCV 2014*, pages 818–833. Springer International Publishing, 2014.