Wolfgang Roth, BSc

Hybrid Generative-Discriminative Training of Gaussian Mixture Models

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Assoc.Prof. Dipl.-Ing. Dr.mont. Franz Pernkopf

Signal Processing and Speech Communications Laboratory

Graz, March 2015

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Abstract (English)

Gaussian mixture models are used in many applications to capture the underlying distribution of the given data. Gaussian mixture models are usually learned according to the generative paradigm, for instance in a maximum likelihood sense, such that the available data is described well by the model. Nevertheless, when generatively learned models are used to perform classification, the resulting classification errors are in practice typically higher than the classification errors of discriminatively trained models. Recent work presented several ways to learn Gaussian mixture models discriminatively to solve this problem. However, none of these approaches takes the likelihood into account. Consequently, the likelihood is typically small and the probabilistic interpretation of the model suffers. A good probabilistic interpretation of the model enables classification of data with missing features in a mathematical sound way whereas discriminative models often rely on heuristics such as imputation techniques. In this thesis we present a hybrid generative-discriminative approach to learn Gaussian mixture models. This is achieved by optimizing an objective that trades off between a generative likelihood term and a discriminative margin term. We show the classification performance on synthetic and real world data. We demonstrate the capabilities of hybrid Gaussian mixture models when classifying data with missing features and show how unlabeled data can be used to improve the accuracy of the classifier, i.e. semi-supervised learning. The resulting models improve the performance of purely generatively learned Gaussian mixture models and achieve a high accuracy in the presence of missing data. The hybrid model is compared to support vector machines and other state of the art Gaussian mixture model classifiers.

Keywords: Machine Learning, Gaussian Mixture Models, Semi-supervised Learning, Missing Features, Hybrid Generative-Discriminative Learning

Abstract (German)

Gaußsche Mischverteilungen werden in vielen Anwendungsgebieten eingesetzt um die zugrundeliegende Verteilung der gegebenen Daten zu modellieren. Üblicherweise werden Gaußsche Mischverteilungen nach dem generativen Prinzip gelernt, wie etwa der Maximum-Likelihood-Methode, sodass die zugrundeliegenden Daten durch das Modell möglichst gut beschrieben werden. Da die Fehlerraten von generativen Modellen bei Klassifikationsanwendungen in der Praxis typischerweise höher sind als jene von diskriminativen Modellen, wurden in einigen Arbeiten Möglichkeiten präsentiert, um Gaußsche Mischverteilung auch diskriminativ zu lernen. Keine der bisherigen Ansätze berücksichtigt jedoch, dass die vorhandenen Daten unter dem Modell eine hohe Likelihood aufweisen, sodass die Interpretation als Wahrscheinlichkeitsmodell verloren geht. Eine gute Interpretation als Wahrscheinlichkeitsmodell ermöglicht es allerdings, Daten mit fehlenden Merkmalen auf natürliche Weise mit mathematischen Methoden zu klassifizieren, wohingegen sich diskriminative Techniken weitestgehend auf Heuristiken wie Merkmalsimputation stützen müssen. In dieser Arbeit stellen wir einen hybriden diskriminativen-generativen Ansatz vor, um Gaußsche Mischverteilungen zu lernen. Dies wird durch Optimierung einer Zielfunktion, die zwischen einem generativen Likelihood-Kriterium und einem diskriminativen Margin-Kriterium abwägt, erreicht. Wir zeigen die Klassifikationsleistung auf echten und künstlichen Daten. Wir demonstrieren die Fähigkeiten von hybriden Gaußschen Mischverteilungen beim Klassifizieren von Daten mit fehlenden Merkmalen. Weiters zeigen wir, wie Daten, von denen man die Klassenzugehörigkeit nicht kennt, die Genauigkeit des Klassifikators verbessern können, sprich halbüberwachtes Lernen. Wir verbessern die Fehlerraten von rein generativen Gaußschen Mischverteilungen und erzielen gleichzeitig eine hohe Genauigkeit, wenn die zu klassifizierenden Daten fehlende Merkmale enthalten. Das hybride Modell wird mit Stützvektormaschinen und anderen Klassifikatoren, die auf Gaußschen Mischverteilungen basieren, verglichen.

Schlüsselwörter: Maschinelles Lernen, Gaußsche Mischverteilungen, Halbüberwachtes Lernen, Fehlende Merkmale, Hybrides Generatives-Diskriminatives Lernen

Contents

1 Introduction

Many technical systems involve decision making procedures where for some given input $\boldsymbol{x}$ an output $\boldsymbol{y}$ needs to be computed. The number of applications and problems which have to deal with decision making is huge and we will give some motivating examples here. In health care, doctors can be assisted by a system that predicts whether a patient suffers from a particular disease or not, provided that some measurements from a medical examination are available. Nowadays it is common that the postal service distributes the deliveries by automatically analyzing their destination address. This process involves handwritten character recognition systems where the goal is to decide the correct letter or digit which is depicted on a digital image. The rapidly evolving field of speech processing is concerned with computing the spoken words for some audio signal. E-mail clients typically provide a spam detection system where new mails are classified as spam or non-spam messages.

To formalize the notion of decision making, we introduce a function f which maps the input $\boldsymbol{x} \in \mathcal{X}$ to an output $\boldsymbol{y} \in \mathcal{Y}$. The spaces $\mathcal{X}$ and $\mathcal{Y}$ vary for each application and can be as diverse as the examples described above. In the medical example the input might be seen as a vector of values, one for each possible measurement. Since typically not every possible measurement is necessary in order to make a reliable diagnosis, some values of this vector could also be missing. The output could be a binary value indicating whether the patient is sick or not, or might just as well be a categorical value indicating the particular disease. In the character recognition task the input can be represented by a matrix of pixel values while the output is a categorical value representing the detected character. In speech processing the input could be a variable length vector containing sound frequencies whereas the output can be phonemes, words or even sentences. In case of the spam application the function f is used to map a whole document to a binary vector indicating whether it is spam or not.

The question is now how to implement a function f that is capable of making the desired decisions. The field of *machine learning* is mainly concerned with automatically finding structures and dependencies in a set of given data. Especially its subfield of *supervised learning* provides a large set of tools that can be used to construct the function f. The idea of supervised learning is simple: Given a set of data examples $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$ representing input values $\boldsymbol{x}_n$ and corresponding output values $\boldsymbol{y}_n$ of the specific applications, the goal is to infer a function f that maps the inputs to the outputs. The term *learning* stems from the interpretation of the construction of f as learning the dependencies of inputs and outputs from some given real world examples. A supervised learning procedure typically starts with the acquisition of input and corresponding output values which are then used to *learn* or *train* the function f. It is espe-

cially the acquisition of the output values, that usually incorporates human expertise and which is typically time-consuming or expensive to perform. The ultimate goal is that the function generalizes well, i.e. its accuracy in predicting the outputs for previously unseen data is high.

The function f is often constructed by first building a model that describes the given data. This model is subsequently used to compute the function f. Since the models are usually determined by a set of parameters $\boldsymbol{\theta}$, the problem of learning the model is often cast as an optimization problem. The goal is to find some parameters $\boldsymbol{\theta}$ such that the corresponding model fits the data well. There are basically two important paradigms that are used to fit a model to some given data. On the one hand, *generative learning* is concerned with modeling the underlying joint probability distribution $p(\boldsymbol{x}, \boldsymbol{y})$ that has generated the data examples at hand. Using Bayes' theorem, the function f then outputs the value $\boldsymbol{y}$ with the highest posterior probability, i.e.

$$f(\boldsymbol{x}) = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{Y}} p(\boldsymbol{y}|\boldsymbol{x}) = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{Y}} \frac{p(\boldsymbol{x}|\boldsymbol{y})\, p(\boldsymbol{y})}{p(\boldsymbol{x})}. \tag{1.1}$$

On the other hand, *discriminative learning* models the output posterior probability $p(\boldsymbol{y}|\boldsymbol{x})$ directly rather than indirectly via the joint probability. There are aspects that favor each of the two approaches. Generative learning can usually be performed cheaper in terms of computation time and memory requirements than discriminative learning. Furthermore, given that certain assumptions hold, in the generative approach the joint probability provides a mathematical sound way to deal with missing values in the data. For instance, in the medical example above, a missing value could be a measurement that has not been conducted in order to make a diagnosis. However, this does not mean that this value does not exist. It merely means that we do not know it. Discriminative learning strategies usually have to rely on heuristics to cope with missing data. The advantage of discriminative learning is that the resulting functions usually provide a high accuracy since they operate on the output criterion directly.

1.1 Aim and Contributions

The aim of this thesis is to combine the ideas of generative and discriminative strategies into a hybrid approach to get the best of both worlds [1]. Specifically, we incorporate the ideas developed in support vector machines (SVMs) to learn a joint distribution $p(\boldsymbol{x}, \boldsymbol{y})$ in a hybrid way. A SVM is a discriminative strategy, that can be used to compute binary outputs. Loosely speaking, SVMs try to separate the data with different output values by a hyperplane, such that the data examples are in some sense far away from this hyperplane. Hence, they fall into the category of large margin techniques. Large margin techniques have been adapted to generative strategies by the concept of the probabilistic margin. Recent work has shown how to use this techniques to learn probabilistic graphical models in a hybrid generative-discriminative way [2].

In this thesis we apply this approach to Gaussian mixture models (GMMs). We will show how to learn a function f with GMMs for real-valued fixed-length input values $\boldsymbol{x}$ and categorical output values $\boldsymbol{y}$. In recent years, different approaches to learn GMMs discriminatively have been proposed. However, none of these techniques takes the likelihood into account. We show how to learn GMMs by simultaneously optimizing the likelihood and the margin of the data. The goal is to improve the classification performance of purely generatively learned models and to be competitive with state of the art algorithms such as SVMs. Furthermore, we aim to achieve good results in the presence of missing data in the inputs as well as in the outputs. Missing data is essentially the motivation for learning a model in a hybrid generative-discriminative way.

We conduct experiments on both synthetic and real world data. The experiments on synthetic data are used to illustrate several properties of hybrid GMMs whereas real world data is used for

comparisons with other classifiers. We show classification experiments on real world data where we randomly removed different amounts of missing features. We show that, if the parameters of the model are carefully tuned, the performance of hybrid GMMs increases for data with and without missing features compared to the generative solution. Furthermore, we show how unlabeled data can be used to improve the performance of the classifier, i.e. semi-supervised learning.

1.2 Outline of the Thesis

In Chapter 2 we explain the necessary background and review the related work. This includes elementary topics of supervised and unsupervised learning and the generative and discriminative paradigm for learning classifiers. We show important properties of GMMs and how to learn its parameters. Since the aim of this thesis is to train a model that can handle missing features, we give a short introduction to the theory of missing data. The chapter concludes with general numerical optimization techniques that we use to optimize the hybrid objective.

Chapter 3 shows how to apply large margin techniques to learn GMMs in a hybrid generative-discriminative way. We derive the hybrid objective from soft-margin SVMs and demonstrate how learning hybrid GMMs can be cast as a smooth unconstrained optimization problem. We show how to handle missing features and present an approach to perform semi-supervised learning. We provide detailed interpretations of several model parameters.

In Chapter 4 we present results of several experiments on real world and synthetic data. We classify data with and without missing features and demonstrate how unlabeled data can be used to improve the performance of hybrid GMMs. The model is compared to other GMM classifiers and SVMs.

Chapter 5 concludes the thesis and gives possible directions for future work.

2 Background and Related Work

In this chapter we present the necessary background needed in the subsequent chapters. Section 2.1 starts with a short introduction to the relevant topics of supervised and unsupervised learning. Section 2.2 describes the generative and discriminative paradigm for training models where each comes with its own advantages. Section 2.3 introduces GMMs, the model that we will use to make decisions. Since the goal of this thesis is to build a model that is capable of dealing with missing data, Section 2.4 gives a brief introduction to the theory of missing data. Section 2.5 concludes the chapter with numerical optimization techniques relevant for this thesis.

2.1 Supervised and Unsupervised Learning

The field of machine learning can be divided into three categories:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

The first two categories are relevant for this thesis and will be explained in more detail in this section. The goal of supervised learning is to model the dependencies of given input and output values, i.e. one aims to infer a function f from some given input-output combinations. Many problems arising in practice, as already mentioned in Chapter 1, can be cast as a supervised learning problem. After introducing the necessary terminology, we explain several concepts of supervised learning by the problem of fitting a polynomial to some given data. We illustrate the problem of overfitting and show common ways to avoid it. Overfitting is a problem common to many supervised algorithms which, loosely speaking, occurs when we model the given data *too* accurate. The section ends with a brief discussion about unsupervised techniques which aim to find different kinds of structures in the given data. Unsupervised techniques are often used in conjunction with supervised techniques.

2.1.1 Supervised learning

Given a set of N data examples $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$ with $\boldsymbol{x}_n \in \mathcal{X}$ and $\boldsymbol{y}_n \in \mathcal{Y}$, the goal of supervised learning is to construct a *decision function* $f : \mathcal{X} \mapsto \mathcal{Y}$ that maps the inputs $\boldsymbol{x}_n$

to the corresponding outputs $\boldsymbol{y}_n$. Although the spaces $\mathcal{X}$ and $\mathcal{Y}$ can be of any possible kind, we will restrict ourselves in this thesis to real valued vector inputs, i.e. $\mathcal{X} = \mathbb{R}^D$, and real or categorical scalar outputs, i.e. $\mathcal{Y} = \mathbb{R}$ or $\mathcal{Y} = \{1, \dots, C\}$. The inputs are sometimes called *feature vectors* and each of their entries is called a *feature*. The output values are called *target values* or *labels*. Hence, the data in supervised learning is also called *labeled data*. If the outputs are real valued, the task of constructing the function f is called a *regression problem*. In case of categorical output values, it is called a *classification problem* and the function f is called a *classifier*. In case of a classification problem, the output values are also called classes. To make a clear distinction between regression and classification problems, we write the output values of classification problems as c rather than y. Since the goal of this thesis is to build a classifier, most of the discussions will be restricted to classification problems.

The process of constructing the function f from some given data examples is also called *learning* or *training* due to its similarity to the learning process of humans. In case of a classification task, one can think of the function f as dividing the space $\mathcal{X}$ into C distinct decision regions and data points are predicted according to the region they fall into. Note that the decision region of a class does not need to be connected. Since in practice the decision function is typically used to predict the target values of previously unseen inputs, an important property to consider is that the function f *generalizes* well. To formalize the notion of generalization, we assume that the data is generated according to a joint distribution $p(\boldsymbol{x}, y)$. Furthermore, we assume that there is a *loss* or *error function* $l(y, \hat{y})$ that quantifies the quality of the prediction $\hat{y}$ if the true target value is y. Then the *generalization error* or *expected loss* of a function f is given by

$$l(f) = \int_{\mathcal{X}} \int_{\mathcal{Y}} p(\boldsymbol{x}, y)\, l(y, f(\boldsymbol{x}))\; dy\; d\boldsymbol{x}. \tag{2.1}$$

A function is said to have a high *generalization performance* if the expected loss is small. The loss function can differ from application to application. A typical choice for classification is the zero-one loss function that incurs a loss of one for incorrectly classified examples and zero loss for correctly classified examples. In this case the generalization error reduces to the probability of misclassification which we simply refer to as the *classification error*. In a medical application it would be much worse to classify a person as healthy if it is sick than vice versa. In such cases the particular type of misclassification must be weighted accordingly. In regression tasks the so called squared loss is a common loss function which simply outputs the square of the difference of the target value y and the prediction y', i.e.

$$l(y, y') = (y - y')^2. \tag{2.2}$$

This discussion shows that it is not only important to predict the targets well for the data that is used to construct f, but also to achieve a high performance on previously unseen data. Since the true underlying distribution is usually not known, the generalization performance is typically estimated using a separate held-out data set that is not used during the training process. Hence, there are some distinctions between the available data to be made. The data which is directly used to learn the function f is called the *training set*. A distinct *test set* is then used to estimate the generalization error in (2.1). There exists yet another type of data that is used during the training process. Many learning algorithms involve the choice of one or more *hyperparameters* which govern the complexity of a model. An appropriate choice for these hyperparameters is crucial in order to achieve a good generalization performance. These parameters usually need to be hand tuned and it is most often not possible to give optimal values for them. To find hyperparameters that result in a good generalization performance, a separate *validation set* is used. The function f is constructed for a set of different values of these hyperparameters and the one with the best performance on the validation set is selected.

At this point one might ask why there is a distinction between the validation and the test set

to be made and why the estimated performance on the validation set is not a valid approximation to the generalization error. The answer is quite simple. When choosing a function that achieves the best performance on the validation set, the hyperparameters, and therefore the model, are implicitly tuned to the validation set. The resulting performance would be an optimistic estimate of the true loss. It is therefore necessary to assess the true performance using a separate test set.

2.1.2 Fitting a polynomial to given data

We want to explain several concepts with a simple regression task based on an example from [3]. Assume that we are given data examples $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ with $x_n \in \mathbb{R}$ and $y_n \in \mathbb{R}$ and we want to find a function f such that $f(x_n) \approx y_n$ for all n. The first choice one has to make is to decide which family of functions should be used to model the data. We postulate that our data was generated by a polynomial with some added noise. Thus, the function f will be of the form

$$f_{\boldsymbol{\theta}}(x) = \sum_{d=0}^{D} \theta_d x^d. \tag{2.3}$$

Even if the data was not really generated by a polynomial, they often provide a good approximation to many problems that are encountered in practice. Since a polynomial of order D is entirely defined by a weight vector $\boldsymbol{\theta} \in \mathbb{R}^{D+1}$, we can associate each vector in $\mathbb{R}^{D+1}$ with a polynomial. We denote the corresponding function by $f_{\boldsymbol{\theta}}$. In this case the weight vector $\boldsymbol{\theta}$ are called the parameters and $\mathbb{R}^{D+1}$ is the parameter space. The goal is now to find a vector $\boldsymbol{\theta} \in \mathbb{R}^{D+1}$ such that the corresponding function $f_{\boldsymbol{\theta}}$ fits to the observed training data and furthermore generalizes well. The task of fitting the training data is usually achieved by formulating an objective function that takes the parameters $\boldsymbol{\theta}$ as input and outputs a numerical value that describes how well the polynomial fits the data. By using the squared loss function (2.2), we obtain the mean squared error function

$$l_{MSE}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} (f_{\boldsymbol{\theta}}(x_n) - y_n)^2. \tag{2.4}$$

The problem of finding a function f is now reduced to finding parameters which minimize this loss function. There is an important point in this task: How does one choose the order of the polynomial D? There is no general answer to this question. It is known that N data points can be fit exactly by a $(N-1)$-order polynomial, i.e. there exists a $\boldsymbol{\theta} \in \mathbb{R}^{N-1}$ such that $l_{MSE}(\boldsymbol{\theta}) = 0$. Figure 2.1(b) shows why choosing the order of the polynomial too large is generally not preferable. The data points lie exactly on the polynomial but, intuitively, the function does not appear to be very natural and the generalization performance of this model is most certainly poor. Hence, the fact that lower order polynomials are a special case of higher order polynomials is not exploited automatically and restrictions must be made by hand. From our previous discussion we can interpret the order of the polynomial as a hyperparameter which needs to be tuned using a validation set. One simple but effective way to attain a good generalization performance is to learn a model for several different polynomial orders and choose the one with the best validation performance.

Figure 2.1(d) shows the validation performance for functions with different polynomial orders D. The graph is reminiscent of a bathtub function with high validation errors for small and higher polynomial orders and smaller errors for intermediate polynomial orders. The regime of higher errors due to too small polynomial orders is called *underfitting* since the model is too simple to model the data well. On the other side of the bathtub we have to deal with *overfitting*.

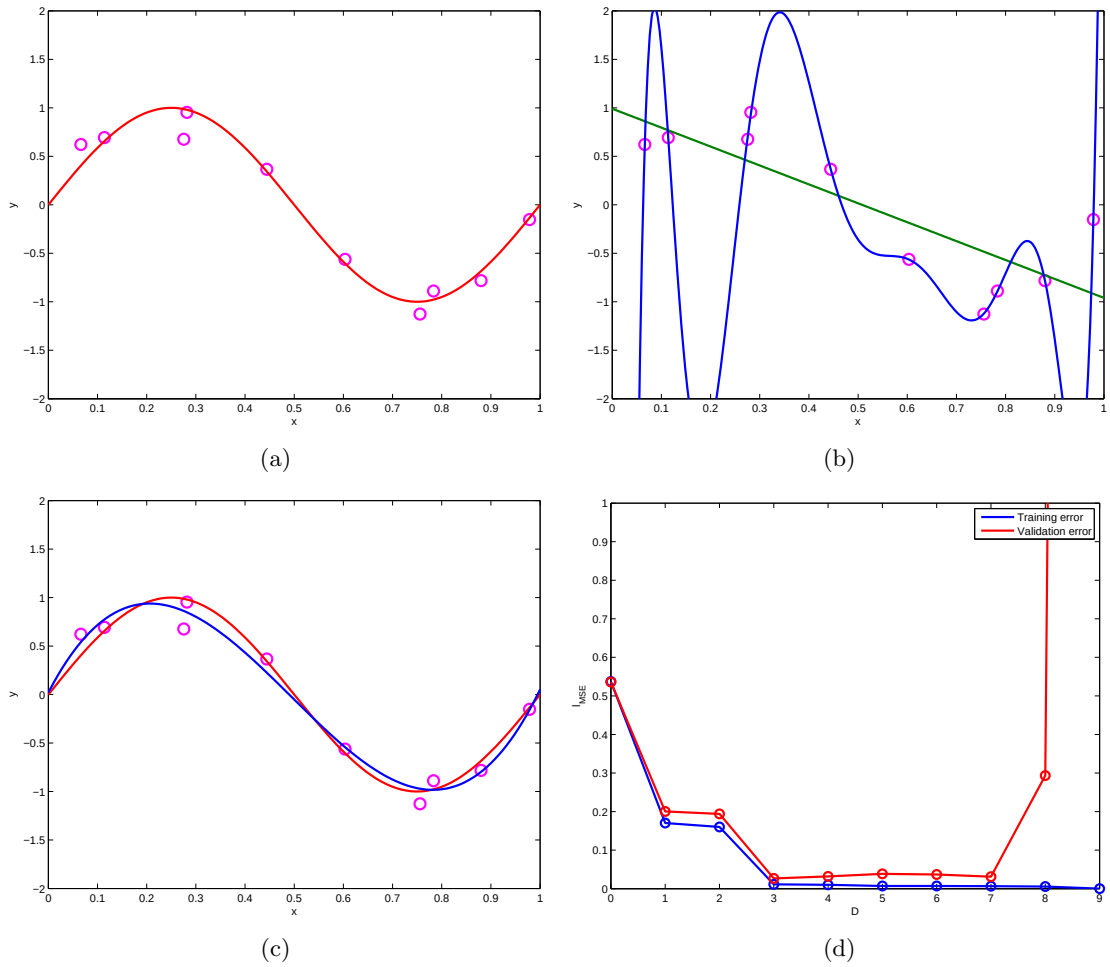


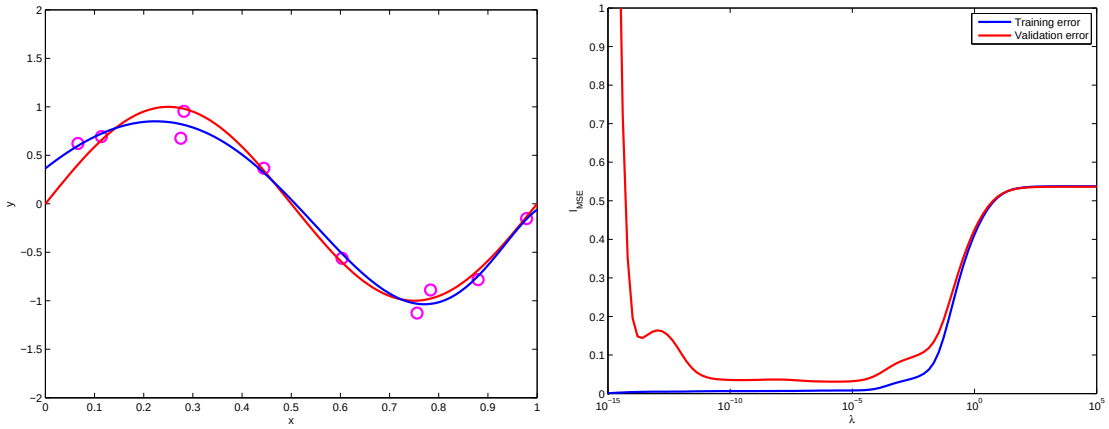
Figure 2.1: Different plots of the polynomial example. (a) shows the training data and the function that was used to generate the data. The blue and the green lines in (b) show an overfitting and an underfitting function respectively. (c) shows the function with the least validation error in blue next to the true function in red. The training and the validation errors are shown in (d) for different values of D. The validation set comprises 50 separate examples.

In this case the model is so expressive that it also fits the inherent noise in the data. Thus, it is desired to find a model somewhere in between that performs well.

2.1.3 Avoiding overfitting

It turns out that overfitting is usually the greater problem. Overfitting occurs especially when the model class is complex while at the same time training data is rare. The *complexity* of a model roughly corresponds to the number of parameters. However, in some cases a model is also said to be complex if the parameters take on large values. Models with many parameters are typically more expressive than models with only a few parameters, meaning that they are capable of representing structures that simpler models cannot. However, in the case of limited training data it is often better to stick to a less expressive model.

A natural way to measure the complexity of a polynomial is its order. This measure alone would imply that all polynomials of the same order are equally complex. By taking a closer look at the weights of an overfitting polynomial, it can be observed that the magnitudes of its weights are typically much larger than the magnitudes of the parameters of non-overfitting models. Consequently it is often desired to compute a model with small parameter values. The



(a) (b)

Figure 2.2: Regression using a polynomial of order 10 with weight regularization. The blue line in (a) shows the polynomial for the λ which resulted in the best validation error. The red line is the true function. The training and the validation errors for different values of λ are show in (b).

norm of the weight vector appears to be another good complexity measure. This discussion leads to another common way to avoid overfitting. A loss function is extended with a so called *regularization term* that penalizes larger parameter values and, therefore, the complexity of the model. The regularized loss function for the regression example is given by

$$l_{REG}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}(f_{\boldsymbol{\theta}}(x_n) - y_n)^2 + \lambda\|\boldsymbol{\theta}\|_2^2. \tag{2.5}$$

The variable λ can be seen as a trade-off hyperparameter between the data fit and the complexity of the model. With $\lambda = 0$ the mean squared error formulation is recovered. As λ goes to infinity one obtains the simplest possible model, namely the zero function. It seems now that we have added another hyperparameter that needs to be tuned, but the regularization term enables us to optimize over a class of models containing rather complicated functions. We can fix a polynomial order that would clearly be too large for the given amount of data examples and the result will still be simple, provided that λ takes on a reasonable value for the given problem. Therefore λ remains the only hyperparameter left to tune. Regularization techniques are used in combination with a wide range of different models in order to help reducing their complexity.

Many models use more than one hyperparameter. In the case of a single hyperparameter one chooses several values and evaluates the model for each of them. In case of two or more hyperparameters one chooses several values for each hyperparameter and typically performs this step exhaustively for each parameter combination. This procedure is also known as *grid search*. Since the number of combinations to evaluate grows exponentially in the number of hyperparameters, it is important to incorporate as few hyperparameters as possible when devising new models. A strategy that has proven useful in practice is to use logarithmically spaced values for the hyperparameters such as $\{2^i\}_{i\in\mathcal{I}}$ for some integer interval $\mathcal{I}$. Another good technique is to start with a coarser grid and to subsequently refine the search around parameter values that seem promising. For some models and applications a *random search* over the hyperparameter space can outperform grid search [4]. Bergstra et al. [5] have also shown how previously searched hyperparameters can be used in order to systematically determine configurations to try next. This is especially useful if the number of hyperparameters is large where grid search becomes intractable.

In any reasonable setting, machine learning algorithms achieve better performance the more

data there is available. In practice it is often the case that labeled data is in some sense difficult or expensive to obtain. As a consequence, the amount of data at hand is small. The classical approach now tells us to divide the data into three different sets where each has its own purpose when learning a model and estimating its performance. If data is scarce, it could happen that every possible partition into these sets has severe drawbacks. If the number of training data is high, the accuracy of the estimated performance can be very poor. On the other side, if the performance should be measured precisely, the learning process will suffer from a small training set. The solution to this problem is called *cross validation*. The idea is to divide the available data in many ways into a training and a test set. For each of these partitions the algorithm is run on the training set and the performance is assessed with the test set. The true error is then estimated to be the average of the individual performances. There is plenty of freedom to partition the data and the most widely used strategy is called *k-fold cross validation*. This approach divides the data into k equally sized partitions. Each partition is then used exactly once as test set while all the others are used as training set. The accuracy of the estimated performance usually grows with the number of partitions. The special case of $k = N$ is called *leave-one-out cross validation*. The drawback of this procedure is that it is typically very time-consuming. In practice a single training run can already take a long time, rendering k-fold cross validation infeasible for larger values of k. If cross validation is used for selecting hyperparameters, it is common to train the model using the best parameters again on the whole data set.

Another method to avoid overfitting is called *early stopping*. Many algorithms in machine learning operate in iterations. For instance most of the optimization algorithms such as gradient descent are iterative procedures. When plotting the performance of the training and validation set against the number of iterations, the result often looks as in Figure 2.3. On the one hand, the performance on the training set is always decreasing since it is incorporated in the objective function that is minimized.[1] On the other hand, the validation error decreases at the beginning until it reaches a minimum. Afterwards it starts to increase again at which point the model starts to overfit. Since the goal is to achieve a good classifier rather than solving the optimization problem optimally, it is often not necessary to continue and one can stop the iterative procedure and return the model that produced the highest validation performance so far. Of course this approach would fail if the validation error would drop again after the algorithm was stopped. Hence, it is not easy to say when it is safe to stop computation. There are many ways to perform early stopping. A very simple and effective heuristic is to stop if the validation performance has not increased for a fixed number of iterations. Prechelt [6] describes other ways to implement early stopping, including the concept of *generalization loss*. Let the validation error in iteration t be $l_{va}(t)$. Then the generalization loss in iteration t is defined as

$$gl(t) = \frac{l_{va}(t)}{\min_{t' \in \{1,\dots,t\}} l_{va}(t')} - 1. \tag{2.6}$$

The generalization loss describes how much the validation performance of the current solution degraded from the best validation performance so far. Early stopping can then be applied if the generalization loss exceeds a fixed prespecified value.

One drawback of early stopping is that it becomes difficult to reason about how much the quality of a solution stems from the problem formulation or the early stopping heuristic. From a theoretic viewpoint, it is often desired to find the hyperparameters which resulted in the best model. Early stopping should also not be combined with cross validation. The number of iterations at which early stopping is performed can vary for each partition. Thus, the found hyperparameters can not be justified to be appropriate for the particular problem.

[1] Depending on the specific task, it can happen that the error occasionally increases for some iterations. Nevertheless, the training error should always decrease in the long term.

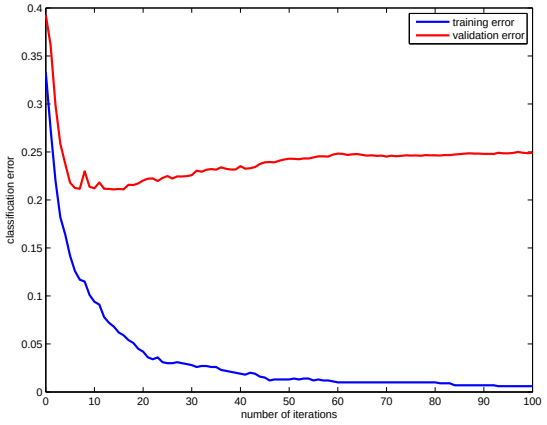


Figure 2.3: A characteristic behavior of the training and the validation error over the number of iterations. While the training error tends to decrease over all iterations, the validation error only drops at the beginning and starts to increase again. It is often not likely that the validation error would decrease again. The plot was generated using hybrid generative-discriminative GMMs on the MNIST data set which was reduced to 50 dimensions with PCA. For details see Section 4.3.1.

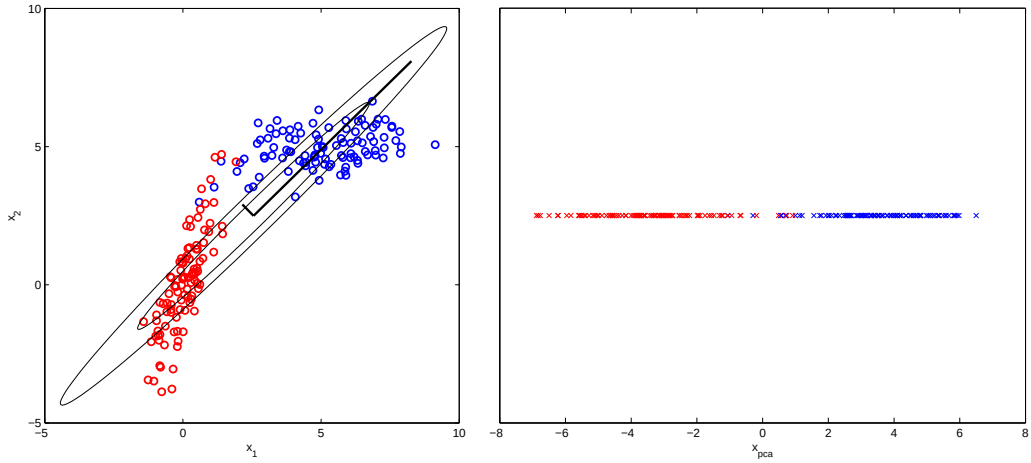
2.1.4 Unsupervised learning

Another main field of machine learning is *unsupervised learning*. In unsupervised learning one has only given a data set $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ without corresponding target values. Hence the data in unsupervised learning is also called *unlabeled data*. Unlike in supervised learning, where one seeks to find a mapping from the data vectors to their target values, the goal of unsupervised learning cannot be so easily identified. Loosely speaking, one tries to find some kind of structures within the given data. In the following we consider instances of unsupervised learning relevant to this thesis, i.e. clustering, density estimation and dimensionality reduction.

An important unsupervised application is *clustering*. A cluster is a subset of the given data that is in some sense similar. An obvious way to define similarity is by using a distance metric such as the Euclidean distance. The task is then to partition the data into several clusters such that the distances between examples from the same clusters are in some sense small. The number of clusters k is usually chosen by hand. In cases where the data can be visualized, i.e. when the dimension is low, the number of the clusters can often be estimated by inspection from a plot. In larger dimensions the choice of the number of clusters is often highly nontrivial.

Another important unsupervised task is *density estimation*. It is assumed that the given data was drawn from an underlying probability distribution which one aims to find. Many probability distribution are described by some parameters $\boldsymbol{\theta}$. For instance a Gaussian distribution can be identified using its mean $\boldsymbol{\mu}$ and its covariance matrix $\boldsymbol{\Sigma}$, i.e. $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The problem of finding the underlying distribution is often cast as an optimization problem where one seeks to find the parameters that optimize an objective which describes how well the corresponding distribution fits the data. This approach will be discussed in more detail in Section 2.2.

Many tasks involve high dimensional data and it is often a useful step to reduce the number of features to ease the problem or simply gain more insights into the data. The field of *dimensionality reduction* is also considered an unsupervised task. It is often desired to perform dimensionality reduction such that as much variability in the data as possible is preserved. *Principal component analysis (PCA)* is a widely used tool to achieve this. The idea of PCA is as follows: Assume that the original space has D dimensions. First one tries to find the direction in the data with the highest variance. In the following steps the direction with the highest variance constrained to be orthogonal to all directions found so far is computed. The result is a set of D orthogonal vectors called the principal components. The data is then reduced by projecting the original data onto the space spanned by the first $\tilde{D}$ dimensions. It turns out that



(a) (b)

Figure 2.4: Illustration of PCA on a simple example. (a) shows the original data together with the two directions found by PCA. (b) shows the same data projected onto the first principal component.

this problem can be solved by performing an eigendecomposition of the empirical covariance matrix of the data. The resulting eigenvalues describe the variability in the direction of the corresponding eigenvector. Assume that the eigenvalues are ordered, i.e. $\lambda_1 \geq \ldots \geq \lambda_D$. The original data can now be projected onto the space spanned by the first $\tilde{D}$ principal components. Given a reasonable amount of principal components $\tilde{D}$, which may in practice be much smaller than the original number of dimensions D, the projected data will only differ slightly from the original data. The amount of preserved variability in the projected data set by using the first $\tilde{D}$ principal components can be quantified by

$$\frac{\sum_{d=1}^{\tilde{D}} \lambda_d}{\sum_{d=1}^{D} \lambda_d}. \tag{2.7}$$

(2.7) simply computes the fraction of variability in the first $\tilde{D}$ principal components. If the retained variability is high, say more than 95%, this can be interpreted to mean that the original data lies close to a $\tilde{D}$-dimensional subspace. This approach is very useful since in many applications the number of parameters to estimate significantly grows with the number of dimensions in the data. Consequently, PCA can often be used as a preprocessing step to speed up training and even increase the performance of classifiers. Furthermore, learning algorithms can become less prone to overfitting due to the reduced amount of parameters to estimate.

2.2 Generative and Discriminative Models

The generative and the discriminative paradigm provide the foundation for many supervised algorithms. On the one hand, generative strategies aim to model the underlying joint distribution $p(\boldsymbol{x}, c)$ of the data and the classes to infer a classifier using Bayes' rule. On the other hand, discriminative strategies try to model the class posterior probability $p(c|\boldsymbol{x})$. Discriminative models tackle the classification criterion directly and thus typically achieve a better classification performance than generative algorithms, especially when the model does not represent the underlying distribution well. However, there are certain aspects that also favor the generative paradigm when performing classification. This section provides an overview of common generative and discriminative strategies. In Chapter 3 we show how to learn a hybrid generative-discriminative classifier.

2.2.1 The Bayes classifier

Probability theory provides natural ways for designing classifiers. As mentioned in Section 2.1, one typically assumes that the data is drawn from an underlying fixed and unknown joint distribution $p(\boldsymbol{x}, c)$ of the data and the class. Using Bayes' rule, the class posterior probability of class c for a given data example $\boldsymbol{x}$ is given by

$$p(c|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|c)\,p(c)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}, c)}{p(\boldsymbol{x})}. \tag{2.8}$$

Since the true underlying distribution $p(\boldsymbol{x}, c)$ is usually not known, one typically sticks to a parameterized approximation $p(\boldsymbol{x}, c|\boldsymbol{\theta})$ thereof. It is then straightforward to build the *Bayes classifier* which outputs the class with the largest posterior probability, i.e.

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \operatorname*{argmax}_{c \in \mathcal{C}} p(c|\boldsymbol{x}, \boldsymbol{\theta}) = \operatorname*{argmax}_{c \in \mathcal{C}} p(\boldsymbol{x}, c|\boldsymbol{\theta}). \tag{2.9}$$

Generative models try to estimate the true underlying joint distribution $p(\boldsymbol{x}, c)$ of the data and the labels and make their predictions according to (2.9). Note that the denominator $p(\boldsymbol{x})$ is not relevant for classification, since it is independent of the class c. It can be shown that the best results in terms of the expected loss (2.1) with respect to a zero-one error function are achieved when the true underlying distribution $p(\boldsymbol{x}, c)$ is used to classify according to (2.9). There are two main problems that arise in practice: First, we often do not know a model class which contains the true underlying distribution. In this case, one has to select a model class that can at least approximate the true distribution reasonably well. Second, even if a model class containing the true distribution is known, the problem of estimating its true parameters can be difficult when data is rare.

Using the product rule, the distribution $p(\boldsymbol{x}, c)$ can be factorized as $p(\boldsymbol{x}|c)\,p(c)$, i.e. the joint distribution can be represented by a prior probability $p(c)$ over the classes and a class conditional distribution $p(\boldsymbol{x}|c)$ for each class. We will assume that each class conditional distribution can be parametrized by a vector $\boldsymbol{\theta}_c \in \boldsymbol{\Theta}_c$ and that the parameters for each class are pairwise disjoint, i.e. $\boldsymbol{\theta}_c \cap \boldsymbol{\theta}_{c'} = \emptyset$ for $c \neq c'$. The parameters of the whole joint distribution are denoted as $\boldsymbol{\theta} = (\pi_1, \ldots, \pi_C, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_C)$ where $p(c|\boldsymbol{\theta}) = \pi_c$. The parameter space is denoted as $\boldsymbol{\Theta}$. The class conditional distribution of class c is written as $p(\boldsymbol{x}|\boldsymbol{\theta}_c)$.

2.2.2 ML estimatation

A common way to fit a model to a given data set is by computing the *maximum likelihood (ML)* estimate of its parameters. Hereby one seeks for parameters $\boldsymbol{\theta}_{ML}$ that maximize the joint probability of the data and the corresponding class labels. If the samples are assumed to be independently and identically distributed, the ML estimator is given by

$$\boldsymbol{\theta}_{ML} = \operatorname*{argmax}_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, c_1, \ldots, c_N|\boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \prod_{n=1}^{N} p(\boldsymbol{x}_n, c_n|\boldsymbol{\theta}). \tag{2.10}$$

By taking the logarithm of (2.10) the product is turned into a sum, making the problem easier to handle. The resulting objective is called the *log-likelihood* function. Note that the maxima are not changed, since the logarithm is a strictly monotonically increasing function. Hence, the ML estimator in terms of the log-likelihood function can be written as

$$\boldsymbol{\theta}_{ML} = \operatorname*{argmax}_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \sum_{n=1}^{N} \log p(\boldsymbol{x}_n, c_n|\boldsymbol{\theta}). \tag{2.11}$$

Using the product rule of probability, the log-likelihood function can also be written as

$$\boldsymbol{\theta}_{ML} = \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmax}} \sum_{c=1}^{C} N_c \; \pi_{c_n} + \sum_{n=1}^{N} \log p\left(\boldsymbol{x}_n|\boldsymbol{\theta}_{c_n}\right), \tag{2.12}$$

where N_c denotes the number of examples of class c. By introducing a Lagrangian multiplier for the sum-to-one constraint of the class priors and by setting the derivative of this function to zero, it turns out that the optimal class priors π_c are given by the fraction of examples that belong to this class, i.e.

$$\pi_{c,ML} = \frac{N_c}{N}. \tag{2.13}$$

We refer to (2.13) as the *empirical prior*. Equation (2.12) shows that the log-likelihood objective decomposes into a sum of terms which only contain the parameters of a single model. Therefore, the optimal parameters can be found by optimizing the parameters of each class conditional distribution separately without taking examples from other classes into account. Furthermore, if the log-likelihood objective is concave for each class, the log-likelihood objective of the full joint distribution will also be concave. The tractability of optimization problems often stems from the fact that the corresponding objective function is concave or convex [7].

The ML solution usually approximates the true underlying distribution more accurately as the number of available training data increases. Given that the true underlying distribution is in the chosen model class, one can show that the ML solution converges under mild assumptions to the true distribution as the number of data examples goes to infinity [8].[2] On the other side, if data is rare, ML often produces poor results. Figure 2.5(a) illustrates ML estimation in case of a Gaussian distribution.

2.2.3 MAP estimation and the posterior predictive distribution

Another widely used approach to learn a distribution for some given data is known as *maximum a-posteriori (MAP)* estimation. Here the key idea is to assume that the model parameters are random variables which are distributed according to a prior distribution $p\left(\boldsymbol{\theta}\right)$. The goal is then to find parameters with maximal posterior probability after observing the data. Hence, the MAP solution is given by

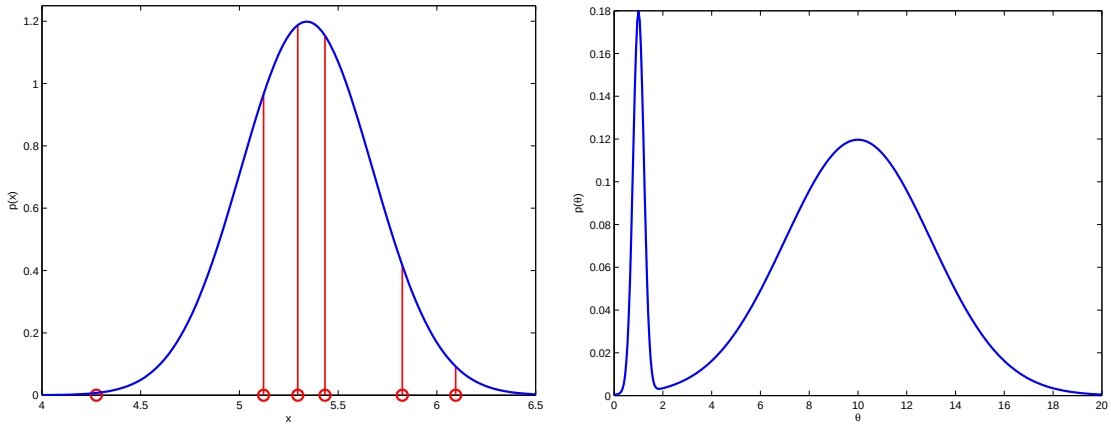
$$\boldsymbol{\theta}_{MAP} = \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmax}}\, p\left(\boldsymbol{\theta}|\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N\right) = \underset{\theta\in\boldsymbol{\Theta}}{\operatorname{argmax}}\, p\left(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N|\boldsymbol{\theta}\right) p\left(\boldsymbol{\theta}\right). \tag{2.14}$$

Using this approach, it is possible to incorporate prior knowledge about the model parameters in the density estimation procedure. Especially in the case of limited data MAP estimation usually outperforms the ML solution. Hence, MAP estimation can be seen as a way to regularize ML in order to avoid overfitting [8].

Nevertheless, like the ML estimate, the MAP solution is also only a point estimate which can result in poor performance if the posterior mode is not in a high density region of the posterior distribution. This scenario is depicted in Figure 2.5(b). The solution to this problem is referred to as the *posterior predictive distribution* where the target value is computed as the class with the highest probability given the data, i.e.

$$f(\boldsymbol{x}) = \underset{c\in\mathcal{C}}{\operatorname{argmax}}\, p\left(c|\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N\right). \tag{2.15}$$

[2] It suffices that the parameters of a probabilistic model are *identifiable* such that the convergence statement holds. Informally speaking, identifiability of the parameters means that different parameters result in different distributions.



(a) (b)

Figure 2.5: (a) illustrates the ML principle with a Gaussian. ML estimation tries to find the model for which the product of the red bars is maximized. (b) shows a possible posterior distribution for the parameter $\boldsymbol{\theta}$. The MAP estimator would be the peak on the left whereas the true parameter is much more likely to lie in the high density region on the right.

The parameters $\boldsymbol{\theta}$ are then incorporated and integrated out again. The posterior predictive distribution can then be rewritten as

$$f(\boldsymbol{x}) = \underset{c \in \mathcal{C}}{\operatorname{argmax}} \int_{\boldsymbol{\Theta}} p\left(c|\boldsymbol{\theta}\right) p\left(\boldsymbol{\theta}|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\right) d\boldsymbol{\theta}. \tag{2.16}$$

A useful interpretation of this method is that all possible parameters are used to classify an example and the results are weighted according to their posterior probability. The drawback of this method is the integration over the whole parameter space, which can not always be computed analytically.

2.2.4 Discriminative learning and MCL estimation

In many practical cases the use of a generative model is restricted to perform classification. A famous quote of Vapnik states that one should never solve a more general problem as an intermediate step than the problem one really intends to solve [9]. When classifying examples according to the class with the maximum posterior probability, the problem of modeling the full joint distribution of the data and the class is clearly a more general one. The field of *discriminative learning* is concerned with modeling the class posterior $p\left(c|\boldsymbol{x}\right)$ directly.

A widely used discriminative training criterion is called *maximum conditional likelihood (MCL)*. This technique is similar to ML estimation. The goal is to find the parameters which result in the highest posterior probability of the true classes. Note that this approach directly operates on the prediction criterion. Assuming that the data examples are independently and identically distributed, the MCL estimator is given by

$$\boldsymbol{\theta}_{MCL} = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\operatorname{argmax}}\, p\left(c_1, \ldots, c_N|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \boldsymbol{\theta}\right) = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\operatorname{argmax}} \prod_{n=1}^{N} p\left(c_n|\boldsymbol{x}_n, \boldsymbol{\theta}\right). \tag{2.17}$$

Taking the logarithm again simplifies the objective:

$$\boldsymbol{\theta}_{MCL} = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\operatorname{argmax}} \sum_{n=1}^{N} \log p\left(c_n|\boldsymbol{x}_n, \boldsymbol{\theta}\right). \tag{2.18}$$

Using Bayes' rule and properties of the logarithm yields

$$\boldsymbol{\theta}_{MCL} = \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmax}} \sum_{n=1}^{N} \log \frac{p\left(\boldsymbol{x}_n|\boldsymbol{\theta}_{c_n}\right)\pi_{c_n}}{\sum_{c'\in\mathcal{C}} p\left(\boldsymbol{x}_n|\boldsymbol{\theta}_{c'}\right)\pi_{c'}} \tag{2.19}$$

$$= \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmax}} \sum_{n=1}^{N} \left(\log p\left(\boldsymbol{x}_n|\boldsymbol{\theta}_{c_n}\right) + \log \pi_{c_n} - \log \sum_{c'\in\mathcal{C}} p\left(\boldsymbol{x}_n|\boldsymbol{\theta}_{c'}\right)\pi_{c'} \right). \tag{2.20}$$

The difference to the joint log-likelihood function lies in the denominator of (2.19), which is simply $p\left(\boldsymbol{x}|\boldsymbol{\theta}\right)$. It is exactly this term which ties all the model parameters together and prevents us from optimizing each model separately as in the ML approach. From a different viewpoint this behavior becomes immediately apparent. Assume that the parameters of a single class model are changed in such a way that the posterior probability of the corresponding class for a certain training example increases. Due to the sum-to-one constraint of the class posteriors, there must be at least one other class for which the posterior probability of this example drops. Thus, changing the parameters of a single class model also affects the objective values for examples of other classes. The objective is in general not concave and globally optimal solutions can less frequently be guaranteed than in the ML approach. Methods which solve this problem are usually more involved. For example in [10] GMMs have been trained according to the MCL criterion using the extended Baum-Welch algorithm.

2.2.5 Large margin learning

Another frequently used discriminative criterion is large margin learning which became particularly important in the context of *support vector machines (SVMs)* [11]. Assume that a given data set with two classes is linearly separable, i.e. there exists a hyperplane such that the examples of the two classes lie on different sides. Such a hyperplane can be seen as the decision boundary that is used to perform classification by checking on which side of the hyperplane a given example lies. In most scenarios this hyperplane is not unique and there is plenty of freedom in how to choose it. SVMs aim to maximize the margin of the hyperplane which is defined as the shortest distance between the decision boundary and any of the points in the data set. The set of support vectors, which are those data points that define the margin, typically contain only a fraction of the original data set. It can be shown that these points determine the whole model and all the other points have effectively no impact on the choice of the parameters. This task can be cast as a convex optimization problem for which efficient algorithms exist.

However, there are several problems that arise in practice. SVMs are inherently designed to solve binary classification tasks. As a consequence, generalizations for multiclass problems are not straightforward. The *one-versus-all* classifier trains a SVM for each class where all examples of this class are separated from all other examples. The predicted class is the one for which the data point lies the farthest away from the corresponding hyperplane. On the other side, the *one-versus-one* classifier learns a SVM for each pair of classes. The output is computed as the class that was predicted most often when using all the pairwise SVMs. Crammer and Singer [12] proposed a way to learn multiclass SVMs directly rather than solving multiple independent SVMs.

Since the hyperplane is solely defined by the support vectors, the model is prone to bad results due to outliers in the set of support vectors. Furthermore, linear separable data is rarely encountered in practice. One solution to these problems is to introduce a non-negative *slack variable* for each example. These slacks measure how much a point lies on the wrong side of the hyperplane. The sum of these slacks can be seen as a measure of data misclassification. The convex optimization problem is then relaxed to allow data points to lie on the wrong side of the

hyperplane. However, these data points increase the penalization term represented by the sum of all slack variables in the objective. This technique is called *soft-margin SVMs*.[3]

SVMs can also be used to construct non-linear decision boundaries using the so called kernel trick. A kernel function is used to implicitly transform the data into a higher dimensional feature space in which a separating hyperplane exists. This way the induced decision boundary in the original feature space becomes non-linear. The technical details and illustrations of SVMs are given in Section 3.1.1.

The technique of margin based learning has also been adopted to probabilistic models. The *probabilistic margin* of a data example $\boldsymbol{x}$ of class c is defined as

$$\delta_c(\boldsymbol{x}) = \frac{p(c|\boldsymbol{x})}{\max_{c' \neq c} p(c'|\boldsymbol{x})}. \tag{2.21}$$

A data example is correctly classified if $\delta_c(\boldsymbol{x})$ is larger than one. A probabilistic model can be learned in a discriminative way by maximizing the margin of the data set. Since all the classes are considered in the definition of the probabilistic margin, the multiclass case is handled more naturally than by usual SVMs. This is similar to the way SVMs have been generalized to the multiclass case by Crammer and Singer [12]. The margin criterion has been used to train probabilistic graphical models [2, 13] and GMMs [10] in a discriminative way. In principle, this approach can be extended to any generative model.

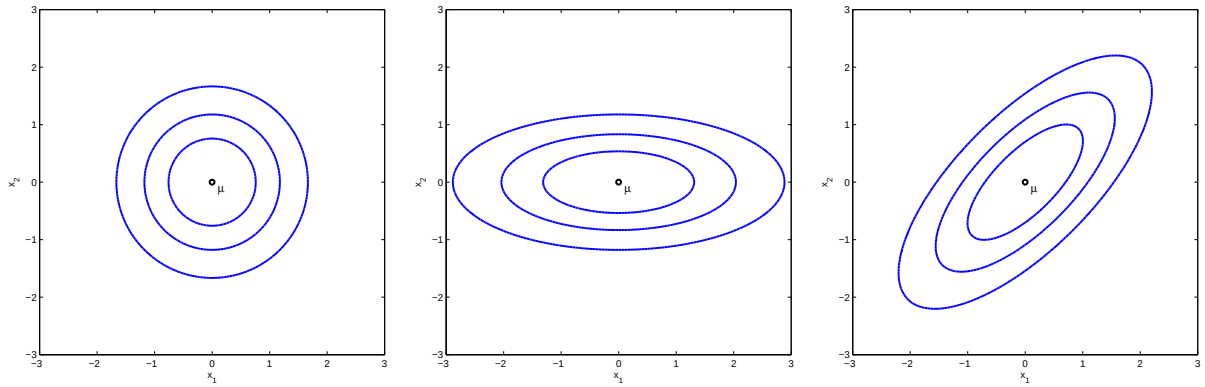
2.2.6 Discussion

Discriminative classifiers usually outperform generative models substantially. However, Ng and Jordan [14] have shown that generative models can yield better results in the regime of rare data. In particular, they have shown that in the case of a generative naive Bayes model and its discriminative analogon logistic regression, the asymptotic classification error with respect to the number of training examples of the generative model is higher than in the discriminative approach. Nevertheless, the generative model achieves its asymptotic performance much faster such that it might be preferable in case of a small data set. On the other side, generative models provide other benefits such as their natural ability to deal with missing data as we will see in Section 2.4. The aim of this thesis is to get the best of both worlds, a discriminative classifier with a generative interpretation. In Chapter 3 we will show how the probabilistic margin can be used to learn GMMs in a hybrid generative-discriminative way by incorporating both a likelihood term and a weighted large-margin term in the objective.

2.3 Gaussian Mixture Models

Gaussian mixture models (GMMs), as a superposition of several weighted Gaussian distributions, provide a powerful way to model the data in many applications. However, the common way to fit probabilistic models, namely ML estimation, comes with several difficulties that do not occur when using only a single Gaussian distribution. We present the EM algorithm, the common way to compute ML parameters for GMMs, and demonstrate its correctness. Furthermore, we show techniques to avoid the inherent shortcomings of ML estimation applied go GMMs.

[3] The classical SVM, where no data example is allowed to be misclassified, is therefore also known as hard margin SVM.



(a) (b) (c)

Figure 2.6: Illustration of how the covariance matrix influences the contours of a Gaussian distribution. (a) depicts the contours of a diagonal covariance matrix with equal entries, i.e. $\boldsymbol{\Sigma} = \sigma \boldsymbol{I}$ where $\boldsymbol{I}$ is the identity matrix. In this case the contours are ball-shaped. (b) shows the contours of a general diagonal covariance matrix. The contours are ellipsoidal and axis-aligned. (c) depicts the contours of a general full covariance matrix. The contours are ellipsoidal but not axis-aligned.

2.3.1 The Gaussian distribution

The data encountered in numerous applications is distributed according to a Gaussian or can at least be approximated by it. The density of a Gaussian in $\mathbb{R}^D$, which is defined by its mean $\boldsymbol{\mu} \in \mathbb{R}^D$ and its symmetric positive semidefinite covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{D \times D}$, is given by

$$\mathcal{N}\left(\boldsymbol{x} | \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \frac{1}{\sqrt{(2\pi)^D \det \boldsymbol{\Sigma}}} \exp\left(-\frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{\mu}\right)^T \boldsymbol{\Sigma}^{-1} \left(\boldsymbol{x} - \boldsymbol{\mu}\right)\right). \tag{2.22}$$

The unique ML parameters of the Gaussian distribution are given by

$$\boldsymbol{\mu}_{ML} = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}_n, \tag{2.23}$$

and

$$\boldsymbol{\Sigma}_{ML} = \frac{1}{N} \sum_{n=1}^{N} \left(\boldsymbol{x}_n - \boldsymbol{\mu}_{ML}\right)\left(\boldsymbol{x}_n - \boldsymbol{\mu}_{ML}\right)^T. \tag{2.24}$$

The parameter $\boldsymbol{\mu}_{ML}$ is the empirical sample mean and $\boldsymbol{\Sigma}_{ML}$ is the empirical covariance matrix or scatter matrix. The density of a Gaussian has an elliptic shape around the mean which is determined by the covariance matrix. The shapes for different types of covariance matrices are depicted in Figure 2.6. The term inside the exponential function is related to the Mahalanobis distance with respect to the covariance matrix $\boldsymbol{\Sigma}$, which is given by

$$d(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(\boldsymbol{x} - \boldsymbol{x}')^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{x}')}. \tag{2.25}$$

The Mahalanobis distance is a metric which takes the variability of the data into account. If the two data examples $\boldsymbol{x}$ and $\boldsymbol{x}'$ vary in a direction of high variability, their Mahalanobis distance will still be somewhat small.

2.3.2 Mixture models

The Gaussian is a unimodal distribution, and the assumption that the given data is also distributed according to a unimodal distribution might be too strong for many applications, especially when the data lies in distinct clusters. *Mixture models* provide a richer class of distributions to overcome shortcomings such as the unimodality assumption of the Gaussian. A mixture model is basically a superposition of several weighted densities which are called *components*. We denote the number of components by K. We assume that each component can be parametrized by a parameter vector $\boldsymbol{\theta}_k$ where k is the component index. The parameter vectors of each component are assumed to be pairwise disjoint, i.e. $\boldsymbol{\theta}_k \cap \boldsymbol{\theta}_{k'} = \emptyset$ for $k \neq k'$. Furthermore, each component is assigned a mixture weight or component prior α_k. The component priors of a mixture model are constrained to be non-negative and sum up to one. The density of a mixture model is then defined as

$$p\left(\boldsymbol{x}|\boldsymbol{\theta}\right) = \sum_{k=1}^{K} \alpha_k p\left(\boldsymbol{x}|\boldsymbol{\theta}_k\right). \tag{2.26}$$

Given that all the components are valid, i.e. their densities are non-negative and integrate to one, and that all the component priors are non-negative and sum up to one, it is straightforward to show that the resulting density is also valid. In the following we will only consider a special kind of mixture model, namely the *Gaussian mixture model (GMM)*, which is obtained by assuming that every component density is a Gaussian. The GMM is a powerful model since it can be used to approximate any distribution, given enough components [8]. The parameters of a GMM can be collected in a vector $\boldsymbol{\theta} = (\alpha_1, \ldots, \alpha_K, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K)$ with $\boldsymbol{\theta}_k = (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. Its density is defined as

$$p\left(\boldsymbol{x}|\boldsymbol{\theta}\right) = \sum_{k=1}^{K} \alpha_k \mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right). \tag{2.27}$$

2.3.3 ML estimation applied to GMMs

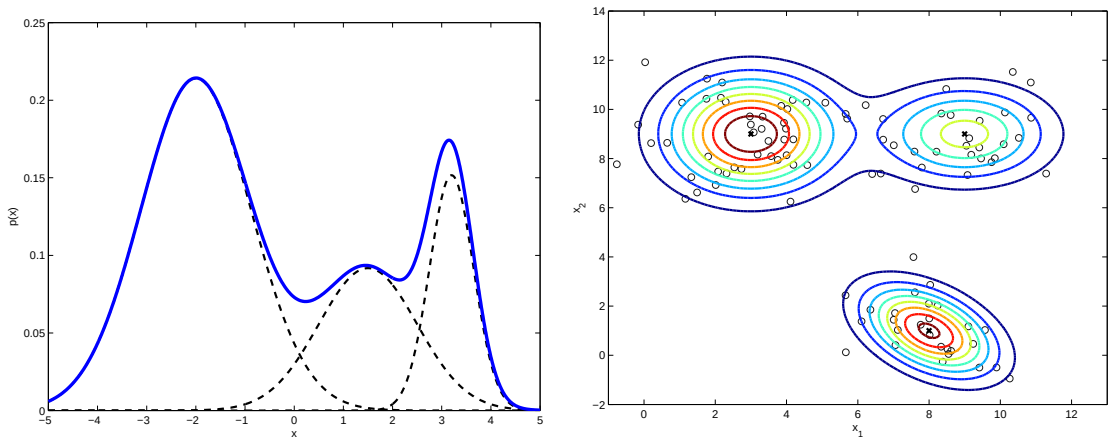
It appears now natural to fit GMMs according to the ML principle. However, there are several difficulties that arise in doing so as we will explain below. The log-likelihood function of a general mixture model is given by

$$p\left(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N|\boldsymbol{\theta}\right) = \sum_{n=1}^{N} \log \sum_{k=1}^{K} \alpha_k p\left(\boldsymbol{x}_n|k\right). \tag{2.28}$$

Unfortunately, this function is in general no longer concave, even if the components belong to the exponential family [3].[4] The reason for this is the sum inside the logarithm. The logarithm does not act on the exponential function directly which makes the log-likelihood function complicated and possibly highly multimodal. As a consequence, there exist in general no closed form solutions to the ML problem and globally optimal solutions in iterative procedures cannot be guaranteed. One can still use general gradient based optimization algorithms and try to solve the problem approximately to obtain reasonable locally optimal parameters.

A problem that comes with GMMs are singularities [3]. Consider the case of a single Gaussian whose mean falls exactly onto a data point. If the covariance matrix of this Gaussian goes to zero, the probability goes to infinity on this point. However, the probability simultaneously tends

[4] Distributions of the exponential family have the property that their log-likelihood function is concave. Many important distributions, such as the Gaussian, the binomial or the gamma distribution, belong to this family.



(a) (b)

Figure 2.7: Illustration of two different GMMs. (a) shows the density of a GMM with three components in one dimension. (b) shows the density of a GMM with three components in two dimensions together with some data examples generated by it.

to zero on all other points. Since the data likelihood is a product of the individual probabilities, it will go to zero rather than infinity. In the case of a GMM this behavior becomes a severe problem. If a component collapses on a single data point, the contribution of this point to the likelihood function becomes infinity. Unlike in the case of a single Gaussian, the other points will have a non-zero contribution because of other components that have not collapsed. The overall likelihood in this case would be infinity. Hence the ML problem is ill-posed. Instead of globally optimizing the likelihood function one tries to find parameters that are locally optimal and do not have singularities.

Another problem of GMMs is that the parameters are only allowed to take specific values. Concretely, the component priors must be non-negative and sum to one and the covariance matrices need to be positive semidefinite.[5] Such constraints often make the optimization procedure significantly harder.

Furthermore, mixture models inherently involve the choice of the number of components to be used. Unless the data is low-dimensional and can be visualized, it is usually difficult to choose the number of components. The number of components K can be seen as a hyperparameter that needs to be tuned, for instance by choosing K to maximize the performance on a separate validation set. It is worth noting that in the case of two or more components there will never be unique globally optimal parameters for the ML problem. Since all components are equal, interchanging the component parameters produces the same distribution.

2.3.4 The EM algorithm

There exists a special purpose algorithm, called the *expectation maximization (EM)* algorithm, which can be used to compute ML parameters in the presence of latent variables. To elicit the latent variable representation of mixture models, we have to view them from a different perspective. Assume that we have an underlying joint distribution $p(\boldsymbol{x}, k)$ which is factorized as $p(\boldsymbol{x}|k)\,p(k)$. Using the sum rule and writing $p(k)$ as α_k, we can immediately see that the marginal distribution $p(\boldsymbol{x})$ is given by (2.26). We call $\boldsymbol{x}$ the observed data, k the latent or hidden data and $(\boldsymbol{x}, k)$ the complete data. It is best to think of each data example being generating by

[5] The covariance matrix of a single Gaussian is also constrained to be positive semidefinite. However the closed-form ML solution guarantees that the covariance matrix is positive semidefinite.

a particular component. The responsible component for generating a data point is denoted by the variable k.

In Section 2.2 we have described that a generative classifier tries to model the joint distribution $p(\boldsymbol{x}, c)$ of the data and the class labels. Now suppose that the class labels c correspond to the hidden variables k and that all the class conditional densities $p(\boldsymbol{x}|c)$ are Gaussians. In this case, the marginal data distribution $p(\boldsymbol{x})$ is exactly a GMM. Since we have established that this task is tractable, we can conclude that optimizing the likelihood of a GMM is also easy, given that we know the hidden variables. The second step of the EM algorithm makes use of this fact.

The following derivation of the EM algorithm and its correctness is based on [3]. To derive the EM algorithm and its correctness, we introduce an auxiliary distribution $q(k)$ and note that the observed data log-likelihood can be written as

$$\log p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} q(k) \log p(\boldsymbol{x}|\boldsymbol{\theta}) \tag{2.29}$$

$$= \sum_{k=1}^{K} q(k) \log \frac{p(\boldsymbol{x}, k|\boldsymbol{\theta})\, q(k)}{p(k|\boldsymbol{x}, \boldsymbol{\theta})\, q(k)} \tag{2.30}$$

$$= \sum_{k=1}^{K} q(k) \log \frac{q(k)}{p(k|\boldsymbol{x}, \boldsymbol{\theta})} + \sum_{k=1}^{K} q(k) \log p(\boldsymbol{x}, k|\boldsymbol{\theta}) - \sum_{k=1}^{K} q(k) \log q(k) \tag{2.31}$$

$$= \mathrm{KL}(q||p) + \mathrm{E}_q\{\log p(\boldsymbol{x}, k|\boldsymbol{\theta})\} + \mathrm{H}(q). \tag{2.32}$$

The first term is the *Kullback-Leibler (KL) divergence*[6] between the auxiliary distribution $q(k)$ and the component posterior $p(k|\boldsymbol{x}, \boldsymbol{\theta})$. The second term is the expected complete data log-likelihood with respect to q and the last term is the entropy of q. Since the KL divergence is always non-negative, we refer to the sum of the latter two terms as the lower bound of the observed data log-likelihood.

The EM algorithm is an iterative procedure that alternates between two steps. Given a parameter configuration $\boldsymbol{\theta}$, the first step, called the expectation step or E-step, seeks for a distribution $q(k)$ which minimizes the KL divergence term. Alternatively, one can think of this step as maximizing the lower bound. This is achieved if $q(k)$ is equal to the component posteriors $p(k|\boldsymbol{x}, \boldsymbol{\theta})$ in which case the KL divergence term vanishes. Using Bayes' theorem, the posterior probabilities can be computed as

$$p(k|\boldsymbol{x}, \boldsymbol{\theta}) = \frac{p(\boldsymbol{x}|k, \boldsymbol{\theta})\, p(k)}{\sum_{k'=1}^{K} p(\boldsymbol{x}|k', \boldsymbol{\theta})\, p(k')}. \tag{2.33}$$

The posterior probabilities can be seen as our belief of a certain data example being generated by a particular component under the current model parameters.

The second step, called the maximization step or M-step, now holds this choice for the auxiliary distribution $q(k)$ fixed and optimizes the lower bound with respect to the parameters $\boldsymbol{\theta}$. It suffices to optimize the expected complete data log-likelihood, since the entropy of $q(k)$ is independent of the parameters. The result of this optimization are new parameters $\boldsymbol{\theta}'$. Given that the component densities are in the exponential family, optimizing this term is typically much easier than optimizing the observed data log-likelihood. This is because the logarithm acts directly on the complete data distribution. The tractability of this computation can also be explained with the correspondence between GMMs and a generative model with known class labels and Gaussian class conditional densities which we have established above. This step simultaneously optimizes the observed data log-likelihood because of the equality which we have

[6] The KL divergence can be seen as a measure of distance between two distributions. However, it is not a symmetric function and therefore not a true metric.

shown before. Unless we are already at a stationary point, the KL divergence term will become positive after this step due to the change in the parameters. The increase in the observed data log-likelihood will therefore even be larger than the increase in the lower bound. This shows that the EM algorithm increases the observed data log-likelihood in each step. These two steps are iterated until some convergence criterion is met. It is common to stop the EM algorithm if the increase in the observed data log-likelihood falls below a given threshold or the norm of the parameter change becomes small.

Since we have assumed that we are already given parameters $\boldsymbol{\theta}$ which were used in the subsequent E-step and M-step respectively to compute new parameters $\boldsymbol{\theta}'$, the questions arises with which parameters one should start. We emphasize here that the EM algorithm is only suited to find locally optimal points rather than global solutions. Both the E-step and the M-step are usually deterministic and hence the starting point is the only way to change the outcome of the algorithm. It turns out that good initial values for the parameters are crucial in order to achieve good model parameters. There are several different ways to accomplish this. A generic approach in the case of GMMs is to initialize the component priors uniformly and the means by some randomly chosen data points. The covariance matrices can be chosen to be identity matrices or by the empirical covariance matrix of all data examples. Since a bad guess of the randomly chosen data points can cause bad results, it is common to restart the procedure several times with different initializations and choose the solution with the highest likelihood. This common technique for iterative optimization procedures is called *random restart*. The EM algorithm for maximizing the likelihood of a GMM is given by Algorithm 1.

Algorithm 1 The EM algorithm for maximizing the likelihood of GMMs

(1) Choose $\boldsymbol{\theta}_0$

(2) E-step: Compute $r_{n,k} = \frac{p(\boldsymbol{x}_n|k,\boldsymbol{\theta})p(k)}{\sum_{k'=1}^{K} p(\boldsymbol{x}_n|k',\boldsymbol{\theta})p(k')} \quad \forall n,k$

(3) M-step: Compute
$$\begin{cases} N_k = \sum_{n=1}^{N} r_{n,k} \\ \alpha_k = \frac{N_k}{N} \\ \boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} r_{n,k} \boldsymbol{x}_n \\ \boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} r_{n,k} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T \end{cases} \quad \forall k$$

(4) If converged, return $\boldsymbol{\theta}$, else goto (2)

As described above the E-step computes the posterior probabilities of the components for all data examples. We can see that the update formulas in the M-step have an intuitive interpretation. The values N_k can be seen as the effective sample counts of the components. For the component priors we simply compute the fraction of the effective sample counts belonging to that component. The component means and the covariance matrices are computed similar to the ML solution. However, each data example is weighted according to its current probability of being generated by that particular component. The EM algorithm ensures that all parameters take values that satisfy the constraints. Since the component posteriors sum up to one, it can be shown that the effective sample counts N_k sum up to N. Therefore, the component priors satisfy the non-negativity and the sum-to-one constraint. The covariance matrices are always positive semidefinite since they are given by a sum of outer products. Furthermore, it turns out that in practice the EM algorithm converges typically much faster than general gradient based algorithms that do not make use of the special problem structure.

2.3.5 Avoiding problems of ML estimation

As mentioned in Section 2.1, many models tend to overfit if the number of parameters is high compared to the amount of given data examples. Consider the case of a GMM with K compo-

nents in a D-dimensional space. We need K component priors[7], K D-dimensional vectors to represent the means and K covariance matrices which are defined by $n(n+1)/2$ parameters. Especially if D is large, the number of parameters becomes huge, since it depends quadratically on the number of dimensions. There exist several techniques which help reducing the number of parameters at the cost of a less expressive model. Instead of using full covariance matrices, one can simply use diagonal covariance matrices to get rid of the quadratic dependency on the number of dimensions. Only D parameters are needed and it suffices that each of these values is non-negative in order to get a positive semidefinite matrix. In addition, non-negativity constraints are much easier to handle than ensuring that a matrix is positive semidefinite. A single Gaussian with a diagonal covariance matrix comes with the inherent assumption that all features are independent from each other, i.e. it factorizes into a product of one-dimensional Gaussians for each dimension. Note however that the distribution of a GMM is still given by a sum over several such Gaussians which in general does not factorize into a product for each dimension. As a result we are still able to capture dependencies between the variables. It is just the number of components which needs to be increased in order to get an approximation at similar quality as with full covariance matrices. The number of parameters can be decreased further by introducing equality constraints on the diagonal elements. In this case, the covariance matrices can be written as $\sigma_k \boldsymbol{I}$. Another approach to reduce the number of parameters is by using a shared covariance matrix for all components. This helps removing the dependency on the number of components. It is also possible to combine both approaches.

The EM algorithm might produce models with ill-conditioned covariance matrices which are either non-invertible or yield a singularity. This happens especially if the number of dimensions is high compared to the amount of observed data examples or the number of components is chosen too large. The EM algorithm can be extended to detect ill-conditioned covariance matrices and simply remove the corresponding components and possibly replace them by a newly initialized one. In some cases it suffices to use one of the techniques described above to reduce the number of parameters. Another simple way is to add a small positive constant to the diagonal of the covariance matrix after the M-step. In the case of diagonal covariance matrices, one can also set a minimal value for the entries on the diagonal and assign these value to entries which drop below it.

2.3.6 Discriminative GMMs

GMMs can also be learned in a discriminative way. Sha [15] learned GMMs with large margin techniques using the Mahalanobis distance with respect to the covariance matrices. In the case of a single Gaussian per class, he was able to formulate a convex semidefinite program to solve this problem. In the case of more than one component per class, he computes ML solutions for each GMM and assigns each data example a proxy label according to the most probable component having generated it. This way, he was able to get rid of latent data and derive a convex semidefinite program for multiple components as well.

As already mentioned, Pernkopf and Wohlmayr [10] proposed to train GMMs using the probabilistic margin. However, they do not consider the likelihood at all which diminishes the interpretation of the trained model as a good approximation of the underlying density. In this thesis we incorporate the ideas developed in [2] where the objective trades off between a large margin term and a likelihood term and thus maintains a better probabilistic interpretation.

[7] Strictly speaking, K-1 parameters would suffice given the non-negativity and the sum-to-one constraints of the component priors.

2.4 Missing Data

Missing data is encountered in many fields. For instance, in health care a data vector could comprise every possible measurements that can be made in the course of a medical examination. However, a doctor can often make a plausible diagnosis even if not all measurements are available. Another example are sensor networks where typically some of the sensors fail to produce measurements. In some cases the amount of observed data is actually rare compared to unobserved data. Consider an online shopping system where users can rate products. The available data can be represented as a matrix where one dimension corresponds to the users and the other dimension corresponds to the products. Such a matrix is typically sparse since users rate only a small fraction of the available products. A data vector can therefore be written as $\boldsymbol{x} = (\boldsymbol{x}^o, \boldsymbol{x}^m)$ where $\boldsymbol{x}^o$ refers to the observed values and $\boldsymbol{x}^m$ refers to the missing values. These examples already show that there is a need for algorithms and models which are able to cope with missing data at both training and testing time.

On the other side, there is also the possibility that some of the target values are missing. Such additional unlabeled data might seem to provide no gain at all. However, it turns out that there exist algorithms which can make use of the unlabeled data, given that certain assumptions hold in the underlying distribution. As we will see, there are numerous applications that can make use of such techniques. Since these algorithms make use of both labeled and unlabeled data, they belong to the field of semi-supervised learning.

2.4.1 Missing features

We start our discussion with the process that renders features hidden. In many cases the actual values of the data can be responsible for making the data unobserved. Suppose we have some sensors producing temperature values. It is more likely that they fail producing a value if the temperature is very high or very low. We now introduce the response indicator vector $\boldsymbol{r} \in \{0, 1\}^D$ that determines which features are observed. To capture the process that makes data hidden, we define a joint distribution $p(\boldsymbol{x}, \boldsymbol{r}|\boldsymbol{\psi}, \boldsymbol{\theta})$ over the data and the response indicator. It is common to factorize this distribution as $p(\boldsymbol{r}|\boldsymbol{x}, \boldsymbol{\psi})\, p(\boldsymbol{x}|\boldsymbol{\theta})$. The first term is called the missing data model and the second term is called the complete data model [16]. We assume that the missing data model can be parametrized by a vector $\boldsymbol{\psi}$. Little and Rubin [17] divided the types of the missing data processes into three groups:

1. The *missing completely at random (MCAR)* assumption is the strongest we can make about the missing data model. It assumes that the underlying process that decides which features are observed is independent from the actual values of the data, i.e. $p(\boldsymbol{r}|\boldsymbol{x}, \boldsymbol{\psi}) = p(\boldsymbol{r}|\boldsymbol{\psi})$.
2. The *missing at random (MAR)* assumption is less restrictive than the MCAR assumption. It assumes that the missing data model only depends on the observed data, i.e. $p(\boldsymbol{r}|\boldsymbol{x}, \boldsymbol{\psi}) = p(\boldsymbol{r}|\boldsymbol{x}^o, \boldsymbol{\psi})$. The MAR assumption can also be seen as imposing the constraint that the probability of a particular missing data pattern must be equal for all possible values of the unobserved features [16].
3. If neither of these assumptions hold, we refer to the *not missing at random (NMAR)* regime.

The following calculations will show that the MAR assumption suffices in order that we can ignore the missing data process when performing ML estimation. The log-likelihood function is given by

$$\log p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_N|\boldsymbol{\theta}) = \sum_{n=1}^{N} \log \int p(\boldsymbol{x}_n, \boldsymbol{r}_n|\boldsymbol{\psi}, \boldsymbol{\theta})\, d\boldsymbol{x}_n^m. \tag{2.34}$$

Using the factorization from above and applying the definition of the MAR assumption yields

$$\log p\left(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_N | \boldsymbol{\theta}\right) = \sum_{n=1}^{N} \log \int p\left(\boldsymbol{r}_n | \boldsymbol{x}_n^o, \boldsymbol{\psi}\right) p\left(\boldsymbol{x}_n | \boldsymbol{\theta}\right) d\boldsymbol{x}_n^m \tag{2.35}$$

$$= \sum_{n=1}^{N} \log p\left(\boldsymbol{r}_n | \boldsymbol{x}_n^o, \boldsymbol{\psi}\right) + \sum_{n=1}^{N} \log p\left(\boldsymbol{x}_n^o | \boldsymbol{\theta}\right). \tag{2.36}$$

Hence, we can essentially ignore the missing data process and simply maximize the likelihood. If we know the true underlying distribution of some data, a similar reasoning shows that the Bayesian classifier, which predicts according to the class with the highest posterior probability, is also optimal and does not have to take care of the missing data process. This already reveals an important feature of probabilistic classifiers: One only needs to integrate out the missing data and then learn or classify according to the marginal distribution of the observed variables.

Nevertheless, there are many discriminative classifiers such as SVMs where missing data cannot be handled merely with marginalization. Different ways to handle missing data have been proposed for these algorithms. *Case deletion* simply discards data examples containing missing features. This method is typically not recommended since the available data might shrink significantly. The discarded data often contains useful information that could help in learning a better model. Furthermore, this method does not provide a way to handle missing data at classification time. Therefore, it is only useful when all features are known when predicting future examples. One of the most famous methods are *imputation* techniques where one tries to find plausible values for the missing features and then ignore the fact that these values were not given at first.

- *Unconditional mean imputation* computes the mean for all observed values of each feature and simply replaces missing entries with it. This method is often not suited, since it chooses the values for the hidden data without taking the observed values into account.
- *Conditional mean imputation* estimates the mean and the sample covariance matrix based on the fully observed data examples which are then used to perform regression for the missing values.
- *K-nearest neighbor imputation* stores the whole training set and computes the K nearest data vectors of this set for new instances. These are then used to impute values according to their mean or using a majority vote in case of categorical values. Nearest neighbor techniques involve the computation of a distance metric. Note that the distance between vectors with missing entries is not well defined. One typically considers only those features which are observed for both vectors when computing the distance between them [18].

The imputation methods described so far generally only produce valid results if the MCAR assumption is given. An advantage of generative models over the techniques described here is that they produce valid results even if the weaker MAR assumption holds. This makes them more suitable in different missing data applications.

Finally, we mention that the joint distribution $p\left(\boldsymbol{x}, \boldsymbol{r} | \boldsymbol{\theta}, \boldsymbol{\psi}\right)$ can just as well be factorized into $p\left(\boldsymbol{x} | \boldsymbol{r}, \boldsymbol{\theta}\right) p\left(\boldsymbol{r} | \boldsymbol{\psi}\right)$. This suggests that each pattern of missing features determines its own distribution. If the number of different patterns in the data is small, one can learn a model for each of these patterns. This has the advantage that for each pattern standard supervised techniques can be applied.

2.4.2 Semi-supervised learning

The task of learning a classifier with partly missing labels leads to the emerging field of *semi-supervised learning*. In addition to a set of data examples with known class labels there is

also a set of unlabeled data. Given a set of labeled data $\{(\boldsymbol{x}_1^l, c_1), \ldots, (\boldsymbol{x}_{N_l}^l, c_{N_l})\}$ and a set of unlabeled data $\{\boldsymbol{x}_1^u, \ldots, \boldsymbol{x}_{N_u}^u\}$, the goal of semi-supervised learning is to improve the performance of a classifier which would use the labeled data alone and discard the unlabeled data.

There are numerous applications in which large amounts of unlabeled data are available but the labeling process is difficult or expensive. For example, in the field of speech classification it is cheap to record hours of speech while labeling this data is very time-consuming and requires experts. Similarly, in a digit recognition task it is easy to produce image data but manually labeling them is a time-consuming task. Hence, the number of unlabeled samples N_u is typically much larger than the number of labeled samples N_l, i.e. $N_u \gg N_l$.

The question is how the given unlabeled data can be used to improve the performance. For semi-supervised algorithms to work, it is important that there exist certain structures in the underlying data distribution. Chapelle et al. [19] describe three assumption out of which at least one must hold in order to benefit from unlabeled data:

Semi-supervised smoothness assumption If two points are close in a high-density region it is likely that their labels are also close.[8] Note that the unlabeled data can help identifying high-density regions.

Cluster assumption If points lie in the same cluster, they are likely to be of the same class. A direct consequence of this assumption is that the decision boundaries are more likely to lie in low-density regions. Hence, this is often referred to as the *low density separation* assumption.

Manifold assumption The high-dimensional data lies on a low-dimensional manifold. It is often easier to learn a model on this lower dimensional space.

We emphasize that these assumptions are not distinct and the latter two can more or less be seen as a special case of the first. Nevertheless, the different assumptions have inspired different algorithms and the way how existing supervised or unsupervised algorithms have been extended to make use of unlabeled data. Especially regularization techniques are based on these assumptions which penalize solutions that do not conform with the particular assumption. The semi-supervised regime lies between supervised and unsupervised learning. On the one hand, semi-supervised algorithms build on supervised techniques and treat the unlabeled data as additional information about the marginal distribution $p(\boldsymbol{x})$. On the other hand, the unsupervised task of clustering can be extended to impose certain constraints on the resulting clusters based on the information contained in the given class labels.

In the following we will present several algorithms to tackle the semi-supervised learning problem. One of the most basic semi-supervised schemes, which can be used on top of many supervised algorithms, is called *self-learning* [19]. The idea is to first learn a classifier using only the labeled data points and then use this classifier to predict the classes for unlabeled points. It is important that the supervised algorithm produces some kind of confidence score which tells us how certain we can be about the predicted class. In the case of a generative classifier this score would be the class posterior probability according to which the samples are predicted. Those unlabeled points for which the most certain outputs can be made are subsequently assigned the corresponding class labels and then included in the labeled data. This is iterated until either all unlabeled data is labeled or some other stopping criterion is met. Sometimes it is also possible to assign soft class labels to the unlabeled data and afterwards weight their impact accordingly.

SVMs have been extended to so called *transductive SVMs* which not only seek to maximize the margin of the labeled data but also maximize the margin of the unlabeled data [20]. Here unlabeled data is allowed to take any class label. It is just important that the points lie far away from the decision boundary. Therefore, transductive SVMs are implementing the low-density separation assumption, since they try to put the decision boundary into a region far away from

[8] For classification problems the term *'being close'* essentially means *'being equal'*.

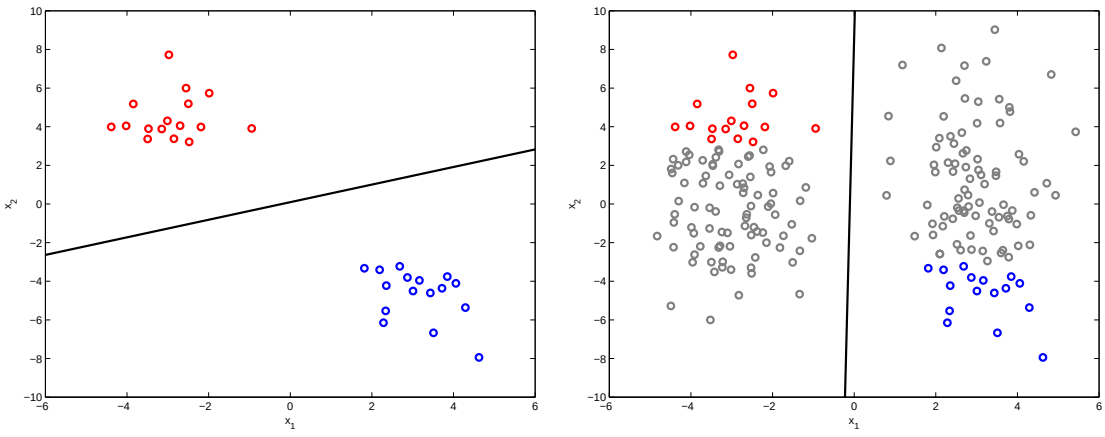


Figure 2.8: (a) shows a decision boundary that was trained using the labeled data alone. In (b) the same data is extended with some unlabeled data. It seems now more natural that the decision boundary lies in the low density region between the two clusters. Note that the decision boundary from the left picture would cut directly through the high density region.

all given points.

The manifold assumption is utilized by *graph based* approaches. These approaches use the given data to build a combinatorial graph, consisting of nodes and edges between these nodes, on which different algorithms can then be applied. The data examples are represented by the nodes. Edges with their corresponding weights typically represent some similarity measure between the data examples whereas missing edges mean no similarity at all. This implicitly assumes that adjacent nodes are more likely to be of the same class. Methods applied to the resulting graph include minimum cut problems in binary classification tasks, where the points of one class are assumed to be sources while the other points are treated as sinks [20]. Other frequently used approaches are label propagation algorithms [19].

Generative classifiers which model the joint distribution $p(\boldsymbol{x}, c)$ can make naturally use of the hidden data. The ML problem can be generalized to maximize the *observed data* log-likelihood by maximizing

$$\sum_{n=1}^{N_l} \log p\left(\boldsymbol{x}_n^l, c_n | \boldsymbol{\theta}\right) + \sum_{n=1}^{N_u} \log p\left(\boldsymbol{x}_n^u | \boldsymbol{\theta}\right). \tag{2.37}$$

The joint distribution can be interpreted as a mixture model with class variables that are only hidden for the unlabeled data. One can then use the EM algorithm or general gradient based algorithms to optimize the likelihood of this model.

It is important to understand that semi-supervised algorithms do not always provide an improvement in the performance of a classifier. It is rather the case that the use of unlabeled data results in a degradation of the performance if the expected properties in the data distribution are not fulfilled. For instance, when applying self-learning and unlabeled data is incorrectly classified, it will lead to a wrong signal for the learning procedure. In [21] it is shown that unlabeled data can degrade the performance of a generative classifier when the true underlying distribution does not lie in the model space being optimized. In such cases one better sticks to the labeled data only. A simple example of semi-supervised learning is shown in Figure 2.8.

2.5 Numerical Optimization

This section introduces general first and second-order algorithms for minimization based on [22]. Numerical optimization is important in many fields of machine learning, since a common way to train models is to optimize an objective that describes how well the model fits the data. We begin our discussion with general first and second-order algorithms. These algorithms typically only provide a search direction and do not tell us how far to move in the given direction. Line-search algorithms formulate a one-dimensional optimization problem along the given search direction in order to find a *good* step size. A good step size can be defined in several ways. We illustrate the widely used Wolfe conditions and describe informally how to satisfy them.

2.5.1 General first and second-order algorithms

First-order algorithms have access to a black box which evaluates the function value and the gradient at a given point. Second-order algorithms additionally have access to the Hessian. We emphasize that in the case of non-convex objective functions any of the following algorithms is generally only capable of finding locally optimal solutions and globally optimal solutions can usually not be guaranteed. The simplest first-order algorithm is the method of *gradient descent* which updates the current solution $\boldsymbol{x}_t$ iteratively according to the rule

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta_t \nabla f(\boldsymbol{x}_t). \tag{2.38}$$

The algorithm maintains a solution $\boldsymbol{x}_t$ and moves from this point into the direction of the steepest descent which is given by the negative gradient. The length of this step is determined by a step size parameter η_t which may vary in every iteration. The advantage of the steepest descent direction is that it is a descent direction for which a sufficiently small η exists such that $f(\boldsymbol{x}_{t+1}) < f(\boldsymbol{x}_t)$ holds, given that $\boldsymbol{x}_t$ is not already a local minimum. This ensures that, with an appropriate choice of the step size η_t, the function value can be reduced in every iteration. This greedy strategy is not always efficient as illustrated in Figure 2.9(a). If the curvature along one direction is much larger than the curvature along other directions, the steepest descent algorithm may oscillate wildly and converge only slowly towards a local minimum. Second-order algorithms make use of the Hessian which provides information about the curvature at the current point. Recall that a twice differentiable function f can be approximated in the vicinity of $\boldsymbol{x}_t$ by the second-order Taylor polynomial $g(\boldsymbol{x})$ which is given by

$$f(\boldsymbol{x}) \approx g(\boldsymbol{x}) = f(\boldsymbol{x}_t) + \nabla f(\boldsymbol{x}_t)^T(\boldsymbol{x} - \boldsymbol{x}_t) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_t)^T \nabla^2 f(\boldsymbol{x}_t)(\boldsymbol{x} - \boldsymbol{x}_t). \tag{2.39}$$

Newton's method is a second-order algorithm which updates the current solution $\boldsymbol{x}_t$ by approximating the function at $\boldsymbol{x}_t$ using the second-order Taylor polynomial $g(\boldsymbol{x})$ in (2.39) and minimizing the approximation instead. The resulting update rule, which is simply obtained by setting the derivative of $g(\boldsymbol{x})$ to zero, is given by

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \left[\nabla^2 f(\boldsymbol{x}_t)\right]^{-1} \nabla f(\boldsymbol{x}_t). \tag{2.40}$$

We refer to $-\left[\nabla^2 f(\boldsymbol{x}_t)\right]^{-1} \nabla f(\boldsymbol{x}_t)$ as the Newton direction. Newton's method, as stated in (2.40), does not involve a step size parameter as in (2.38). Nevertheless, in practice the algorithm is often extended to include a step size especially when combined with line search methods as described later in this section. There is one issue that needs to be taken care of. Unlike in the case of gradient descent, the Newton direction does not always provide a descent direction. However, one can show that a descent direction is obtained if the Hessian $\nabla^2 f(\boldsymbol{x_t})$ is positive definite. If this is not the case, it is still possible to add sufficiently large values to the diagonal

entries of the Hessian to make it positive definite in order to obtain a descent direction.

The drawback of first-order methods is that they typically converge much slower than second-order methods. Second-order methods come with the disadvantage that they need to compute the inverse of the Hessian which might be very time-consuming. Furthermore, the space required to store the Hessian depends quadratically on the number of parameters to optimize. This can be intractable if there are several tens of thousands variables to optimize. *Quasi-Newton methods* are first-order algorithms that use the available information from previous iterations to approximate the Hessian matrix at the current point. Instead of using the true Hessian $\nabla^2 f(\boldsymbol{x})$ when approximating the function as in (2.39), quasi-Newton methods use an approximated Hessian $\boldsymbol{B}_t$. The idea is to compute the matrix $\boldsymbol{B}_{t+1}$ such that the gradient of the corresponding quadratic approximation equals the gradient of f at $\boldsymbol{x}_{t+1}$ and $\boldsymbol{x}_t$. The matrix $\boldsymbol{B}_{t+1}$ is generally not unique which gives rise to different quasi-Newton algorithms.

The *BFGS* quasi-Newton algorithm [22], named after its inventors Broyden, Fletcher, Goldfarb and Shanno, arises by searching for the matrix $\boldsymbol{H}_{t+1} = \boldsymbol{B}_{t+1}^{-1}$ that is symmetric and closest to the previous approximation of the inverse $\boldsymbol{H}_t$ in terms of a weighted Frobenius norm. By approximating the inverse directly, the costly operation of inverting a matrix as required in Newton's method is avoided. The unique matrix $\boldsymbol{H}_{t+1}$ is given by the BFGS update formula

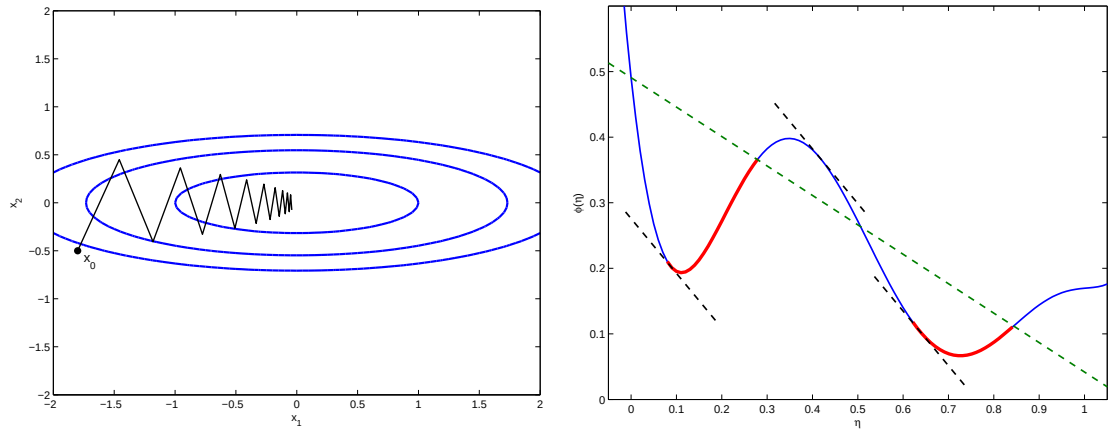
$$\boldsymbol{H}_{t+1} = \left(\boldsymbol{I} - \frac{\boldsymbol{s}_t \cdot \boldsymbol{y}_t^T}{\boldsymbol{y}_t^T \cdot \boldsymbol{s}_t}\right) \boldsymbol{H}_t \left(\boldsymbol{I} - \frac{\boldsymbol{y}_t \cdot \boldsymbol{s}_t^T}{\boldsymbol{y}_t^T \cdot \boldsymbol{s}_t}\right) + \frac{\boldsymbol{s}_t \cdot \boldsymbol{s}_t^T}{\boldsymbol{y}_t^T \cdot \boldsymbol{s}_t}, \tag{2.41}$$

where we refer to $\boldsymbol{s}_t = \boldsymbol{x}_{t+1} - \boldsymbol{x}_t$ as the difference in parameter space and to $\boldsymbol{y}_t = \nabla f(\boldsymbol{x}_{t+1}) - \nabla f(\boldsymbol{x}_t)$ as the gradient difference. The new approximation of the inverse Hessian is computed using the previous approximation $\boldsymbol{H}_t$ and the differences in gradient and parameter space from the previous to the current iteration. It remains to chose an initial matrix $\boldsymbol{H}_0$. One can show that the direction $-\boldsymbol{H}_{t+1} \cdot \nabla f(\boldsymbol{x}_{t+1})$ is a descent direction, given that the previous approximation $\boldsymbol{H}_t$ is positive definite. Hence, an arbitrary positive definite matrix can be chosen for $\boldsymbol{H}_0$. A common choice is the identity matrix for which the first iteration reduces to a simple gradient descent step wheras the following steps can start to make use of the approximated Hessian.

Quasi-Newton algorithms, as stated above, avoid the problem of evaluating the Hessian analytically and the effort of inverting it. Nevertheless, they do not solve the storage problem which can still make this approach intractable for optimization problems with huge amounts of variables. The solution to the memory problem is provied by limited memory variants of the corresponding algorithms. Notice that in the BFGS algorithm the approximation of the inverse Hessian $\boldsymbol{H}_t$ only depends on the initial approximation $\boldsymbol{H}_0$ and the differences in gradient and parameter space up to the current iteration. Nocedal [23] came up with an algorithm that is capable of expressing the product $\boldsymbol{H}_t \cdot \nabla f(\boldsymbol{x}_t)$, where $\boldsymbol{H}_t$ is computed according to the BFGS algorithm, using only inner products. As a consequence, the approximated Hessian and its inverse do not need to be stored explicitly. The *L-BFGS* algorithm, as a limited memory variant of the BFGS algorithm, only uses the last l iterations to compute this product. Hence, it suffices to store only the last l differences in gradient and parameter space together with the (sparse) initial matrix $\boldsymbol{H}_0$.

2.5.2 Line search

All algorithms explained so far update their current solution $\boldsymbol{x}_t$ by moving into a descent direction. Nevertheless, the algorithms do not tell us how far we should move into the given direction. Even in the case of Newton's method, in which the choice of the step size is usually chosen to be one, other choices might provide a higher decrease in function value. Especially if the second-order Taylor polynomial is a poor approximation of the true function, a step size $\eta \neq 1$ might prove useful. *Line search methods* are usually placed on top of direction finding algorithms in



(a) (b)

Figure 2.9: (a) shows the oscillating behavior of the gradient descent algorithm if the curvature differs significantly in each direction. The black line shows how gradient descent progresses towards the local minimum at the origin. (b) illustrates the Wolfe conditions. The dashed green line shows the sufficient decrease condition. Points with a function value below this line conform the sufficient decrease condition. The black dashed lines are tangents to the function where the curvature condition is satisfied with equality. The red parts of the function satisfy both Wolfe conditions.

order to compute an appropriate step size. Given a point $\boldsymbol{x}_t$ and a descent direction $\boldsymbol{v}$, the goal of line search methods is to find a positive step size η that minimizes the function f along the line $\boldsymbol{x}_t + \eta\boldsymbol{v}$. This can be written as the one-dimensional optimization problem

$$\begin{aligned} &\underset{\eta}{\text{minimize}} \quad \phi(\eta) = f(\boldsymbol{x}_t + \eta\boldsymbol{v}) \\ &\text{s.t.} \quad \eta > 0. \end{aligned} \tag{2.42}$$

Optimizing ϕ globally is generally hard, since optimizing f globally is hard. Instead of searching for a global minimizer of ϕ, one typically applies *inexact* line-search algorithms that aim to find a step size η that conforms some predefined conditions rather than being globally or locally optimal. One of the most common conditions used for line search methods are the *Wolfe conditions*, which are given by the two inequalities

$$f(\boldsymbol{x}_t + \eta\boldsymbol{v}) \leq f(\boldsymbol{x}_t) + c_1\eta\nabla f(\boldsymbol{x}_t)^T\boldsymbol{v} \tag{2.43}$$

and

$$\nabla f(\boldsymbol{x}_t + \eta\boldsymbol{v})^T\boldsymbol{v} \geq c_2\nabla f(\boldsymbol{x}_t)^T\boldsymbol{v}. \tag{2.44}$$

The expression $\nabla f(\boldsymbol{x}_t)^T\boldsymbol{v}$ is the directional derivative along the search direction $\boldsymbol{v}$ at $\boldsymbol{x}_t$. c_1 and c_2 are constants in $(0, 1)$. The right-hand side of (2.43) without the factor c_1 is a linear approximation to the one-dimensional problem at $\eta = 0$. The factor $c_1 \in (0, 1)$ decreases the magnitude of the slope of the linear approximation such that there exists a sufficiently small η that satisfies the condition. Since $\boldsymbol{v}$ is a descent direction, the expression $\nabla f(\boldsymbol{x}_t)^T\boldsymbol{v}$ is negative and hence the right-hand side of (2.43) is always smaller than $f(\boldsymbol{x}_t)$ for $\eta > 0$. Consequently, any positive η satisfying (2.43) is guaranteed to decrease the function value. (2.43) is therefore also known as the sufficient decrease condition.

The sufficient decrease condition alone does not prevent us from taking very small steps, since any sufficiently small η satisfies it. Therefore, a second condition is needed to ensure larger steps. Inequality (2.44) is called the curvature condition. The left-hand side of (2.44) is the derivative of ϕ at η whereas the right-hand side is the derivative of ϕ at zero scaled by the factor c_2. The factor

$c_2 \in (0, 1)$ ensures that the magnitude of the slope of the one-dimensional problem decreases proportionally to the slope at $\eta = 0$. The Wolfe conditions are depicted in Figure 2.9(b). It can be shown that there exist step sizes for any continuously differentiable function, that is bounded from below, which satisfy the Wolfe conditions. Furthermore, the Wolfe conditions can be used to show that algorithms like gradient descent converge towards a stationary point where the gradient vanishes when used in combination with line search algorithms [22].

The goal of inexact line search methods is now to find a step size that satisfies the Wolfe conditions with as few function and gradient evaluations as possible. Line search methods proceed by choosing an initial step size and checking the Wolfe conditions. If the conditions are not satisfied, the gathered information of the function value and gradient at the previously tried step sizes can be used to direct the choice of the next step size. A common way is to approximate the one-dimensional line search function ϕ by fitting a second or third-order polynomial to the function and derivative values obtained in the previous steps. The minimizer of this approximation often provides a good choice for the next step size to try out. In addition, by checking which conditions are violated one can conclude if the previous step size was chosen too large or too small. This information can further guide the search for an appropriate step size. In the best case the initial step size already conforms the Wolfe conditions. There exist techniques to incorporate knowledge from previous iterations to find a proper step size [22].

3

Hybrid Generative-Discriminative Gaussian Mixture Models

In this thesis, we aim to learn a generative classifier discriminatively in order to achieve a high classification performance, especially in the presence of missing data. We build a hybrid generative and discriminative classifier where each class conditional distribution $p(\boldsymbol{x}|\boldsymbol{\theta}_c)$ is modeled with a GMM and examples are classified according to the class with the highest posterior probability. In Section 3.1 we show how the soft margin formulation of SVMs can be adapted to learn a classifier simultaneously in a generative and discriminative way. In Section 3.2 we show how to tackle the resulting optimization problem with general purpose optimization algorithms. Section 3.3 presents how this model can be used in the presence of missing data.

3.1 Problem Formulation

The generative part of the learning procedure is based on ML estimation which is a fairly well understood problem in the case of GMMs. The discriminative part is based on maximizing the probabilistic margin. To this end, we adapt the techniques of the soft margin formulation for SVMs to our needs. The basic ideas of SVMs were already introduced in Section 2.2. Here we delve further into the technical details and briefly review how soft margin SVMs are trained. The soft margin formulation is then adapted to state the hybrid generative-discriminative objective. We illustrate difficulties of the hybrid objective that arise due to the large margin part and give interpretations for the hyperparameters.

3.1.1 Linear SVMs

We start with the case of a linear separable set where one can find a hyperplane that has the examples of both classes on different sides. The hyperplane, defined by the parameters $(\boldsymbol{w}, b)$, is given by the equation $\boldsymbol{x} \cdot \boldsymbol{w} + b = 0$. The data set is given by $\{(\boldsymbol{x}_1, c_1), \ldots, (\boldsymbol{x}_N, c_n)\}$ where $\boldsymbol{x}_n \in \mathbb{R}^D$ and $c_n \in \{-1, 1\}$. The decision rule of the classifier is then given by the sign of $\boldsymbol{x} \cdot \boldsymbol{w} + b$. The goal is to find the hyperplane with maximum margin to the data set, i.e. the closest points are as far away as possible from the decision boundary. Using basic geometric properties, the perpendicular distance of a point $\boldsymbol{x}$ to the hyperplane is given by $(\boldsymbol{x} \cdot \boldsymbol{w} + b)/\|\boldsymbol{w}\|$. The choice

of the class labels enables us to multiply this expression by c so that we can assume that it is positive for correctly classified examples. The problem of maximizing the margin with respect to the variables $\boldsymbol{w}$ and b can now be stated as

$$\underset{\boldsymbol{w},b}{\text{maximize}} \quad \min_n \frac{c_n \left(\boldsymbol{w} \cdot \boldsymbol{x}_n + b\right)}{\|\boldsymbol{w}\|}. \tag{3.1}$$

This expression is difficult to optimize. However, the parameters $\boldsymbol{w}$ and b are scale invariant so that multiplication by any factor α does not change the solution, i.e.

$$\frac{c_n \left((\alpha \boldsymbol{w}) \cdot \boldsymbol{x}_n + \alpha b\right)}{\|\alpha \boldsymbol{w}\|} = \frac{c_n \alpha \left(\boldsymbol{w} \cdot \boldsymbol{x}_n + b\right)}{\alpha \|\boldsymbol{w}\|}. \tag{3.2}$$

We can use this degree of freedom to normalize the parameters in such a way that the support vectors satisfy $c_n \left(\boldsymbol{w} \cdot \boldsymbol{x}_n + b\right) = 1$. Since the distance of the support vectors is minimal among the whole data set, all other points satisfy the constraint $c_n \left(\boldsymbol{w} \cdot \boldsymbol{x}_n + b\right) \geq 1$. The formulation above can now be recast as the following constrained optimization problem

$$\begin{aligned} &\underset{\boldsymbol{w},b}{\text{maximize}} \quad \frac{1}{\|\boldsymbol{w}\|} \\ &\text{s.t.} \quad c_n \left(\boldsymbol{w} \cdot \boldsymbol{x}_n + b\right) \geq 1 \quad \forall n \end{aligned} \tag{3.3}$$

or equivalently

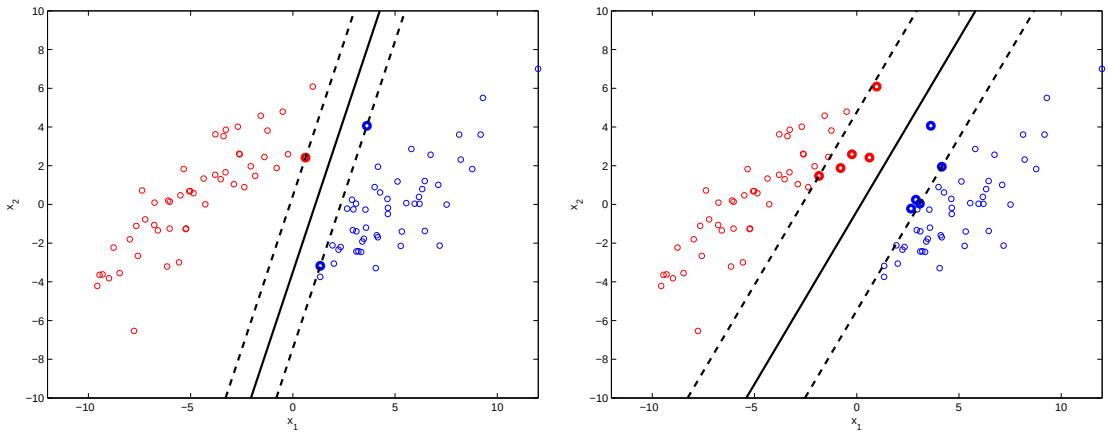
$$\begin{aligned} &\underset{\boldsymbol{w},b}{\text{minimize}} \quad \|\boldsymbol{w}\|^2 \\ &\text{s.t.} \quad c_n \left(\boldsymbol{w} \cdot \boldsymbol{x}_n + b\right) \geq 1 \quad \forall n. \end{aligned} \tag{3.4}$$

Note that this problem is convex and thus a global solution can be found. The points corresponding to constraints satisfied with equality will be the support vectors. By minimizing $\|\boldsymbol{w}\|^2$ it turns out that at least one point of each class will be a support vector. One can show that the support vectors completely determine the model. This elegant convex formulation is also a main reason why SVMs have achieved so much attention in the community.

Next we relax the problem to allow points to lie on the wrong side of the decision boundary. This brings us to the notion of soft margin SVMs. We introduce for each data example $\boldsymbol{x}_n$ a slack variable ξ_n and recast the problem as

$$\begin{aligned} &\underset{\boldsymbol{w},b,\boldsymbol{\xi}}{\text{minimize}} \quad \|\boldsymbol{w}\|^2 + \lambda \sum_{n=1}^{N} \xi_n \\ &\text{s.t.} \quad c_n \left(\boldsymbol{w} \cdot \boldsymbol{x}_n + b\right) \geq 1 - \xi_n \quad \forall n. \end{aligned} \tag{3.5}$$

There always exists a feasible solution to this problem by making the slacks sufficiently large. Nevertheless, points falling on the wrong side will cause a penalization according to how far they are away from the decision boundary. Actually, even correctly classified examples cause a penalization if they fall inside the region defined by the margin. It is easy to see that the slack of an incorrectly classified example is larger than one. Hence, the sum of slacks can be interpreted as an upper bound on the number of incorrectly classified examples. Therefore, the newly introduced hyperparameter λ determines how much we want to classify the training data exactly and how sensitive we deal with single data points lying close to the decision boundary. If λ is large, the optimization procedure will seek for a hyperplane with as few incorrectly classified examples as possible. Indeed, the hard-margin SVM formulation is recovered as λ goes to infinity. When λ becomes smaller it can happen that a few points close to the boundary may be ignored in the favor of producing a decision boundary with a larger margin for the



(a) (b)

Figure 3.1: (a) shows a hard margin SVM and (b) shows a soft margin SVM on the same data set. The solid line shows the decision boundary and the dashed line shows the margin which is defined by $\boldsymbol{w}^T\boldsymbol{x} + b = \pm 1$. The bold data examples are the support vectors. The hard margin SVM in this example seems prone to the outliers whereas the decision boundary of the soft margin appears more natural regarding the majority of the data examples.

remaining data points [24]. This might be a desired behavior since SVMs can overfit severely if support vectors happen to be outliers. The sum of slacks can therefore be seen as a measure of data misfit whereas the norm of the weight vector can be interpreted as a regularization term. Similarly to the hard margin SVM, a soft margin SVM is completely determined by the support vectors. However, every point falling inside the region defined by the margin is now a support vector. The hard margin and the soft margin SVMs are illustrated in Figure 3.1.

The soft margin formulation of the SVM can be rewritten as the equivalent unconstrained optimization problem

$$\underset{\boldsymbol{w},b}{\text{minimize}} \quad \|\boldsymbol{w}\|^2 + \lambda \sum_{n=1}^{N} \mathrm{h}\left(1 - c_n\left(\boldsymbol{w} \cdot \boldsymbol{x}_n + b\right)\right) \tag{3.6}$$

where $\mathrm{h}(t)$ is the *hinge function* which is given by

$$\mathrm{h}(t) = \begin{cases} 0 & t \leq 0 \\ t & t > 0 \end{cases}. \tag{3.7}$$

Note that this is a convex problem due to the convexity of the hinge function and optimal solutions can thus be guaranteed.

3.1.2 The hybrid generative-discriminative objective

We will now elaborate on this problem formulation in order to maximize generative models with respect to the probabilistic margin. As already stated in Section 2.2.5, the probabilistic margin of the n-th data example $(\boldsymbol{x}_n, c_n)$ is given by

$$\delta_n = \frac{p\left(c_n|\boldsymbol{x}_n\right)}{\max_{c \neq c_n} p\left(c|\boldsymbol{x}_n\right)} = \frac{p\left(\boldsymbol{x}_n, c_n\right)}{\max_{c \neq c_n} p\left(\boldsymbol{x}_n, c\right)}. \tag{3.8}$$

By defining the probabilistic margin for the n-th example $\boldsymbol{x}_n$ with respect to class c as

$$\delta_{n,c} = \frac{p\left(\boldsymbol{x}_n, c_n\right)}{p\left(\boldsymbol{x}_n, c\right)}, \tag{3.9}$$

we can rewrite the probabilistic margin as

$$\delta_n = \min_{c \neq c_n} \delta_{n,c}. \tag{3.10}$$

Note that a data example is correctly classified if the probabilistic margin is larger than one. These values are especially interesting in the logarithmic domain, since the generative part is also based on the log-likelihood function. In addition, subtraction rather than division more closely reflects the notion of a distance. Hence, we define

$$\beta_{n,c} = \log \delta_{n,c} = \log p\left(\boldsymbol{x}_n, c_n\right) - \log p\left(\boldsymbol{x}_n, c\right) \tag{3.11}$$

$$\beta_n = \log \delta_n = \min_{c \neq c_n} \left[\log p\left(\boldsymbol{x}_n, c_n\right) - \log p\left(\boldsymbol{x}_n, c\right)\right] = \min_{c \neq c_n} \beta_{n,c}. \tag{3.12}$$

A data example $(\boldsymbol{x}_n, c_n)$ is correctly classified if $\beta_n > 0$. We refer to the class $c \neq c_n$ that achieves the minimum margin as the *best competitor class*. We are now able to adapt the soft margin SVM formulation from above to state the hybrid generative-discriminative objective function, i.e.

$$l_{hybrid}(\boldsymbol{\theta}) = \underbrace{-\sum_{n=1}^{N} \log p\left(\boldsymbol{x}_n, c_n|\boldsymbol{\theta}\right)}_{\text{generative likelihood term}} + \underbrace{\lambda \sum_{n=1}^{N} \mathrm{h}\left(\gamma - \beta_n(\boldsymbol{\theta})\right)}_{\text{discriminative margin term}} . \tag{3.13}$$

We emphasize the dependence of the probabilistic margin on the parameters. Note the minus sign in front of the likelihood term in order to optimize both objectives simultaneously by a minimization problem. This objective is independent of the generative model being used and can in fact be used with any other probabilistic model. In this thesis the class conditional densities $p\left(\boldsymbol{x}|c\right)$ are all assumed to be GMMs. Thus, the parameters of the joint probability $p\left(\boldsymbol{x}, c|\boldsymbol{\theta}\right)$ are parametrized by a vector $\boldsymbol{\theta} = (\pi_1, \ldots, \pi_c, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_c)$ where $p\left(c|\boldsymbol{\theta}\right) = \pi_c$ and $\boldsymbol{\theta}_c = (\boldsymbol{\alpha}_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$. We denote the number of components of class c as K_c. $\boldsymbol{\alpha}_c = (\alpha_{c,1}, \ldots, \alpha_{c,K_c})$ contains the component priors of class c. $\boldsymbol{\mu}_c = (\boldsymbol{\mu}_{c,1}, \ldots, \boldsymbol{\mu}_{c,K_c})$ and $\boldsymbol{\Sigma}_c = (\boldsymbol{\Sigma}_{c,1}, \ldots, \boldsymbol{\Sigma}_{c,K_c})$ define the mixture components of class c.

3.1.3 Interpretation and problems of the hybrid objective

The similarity to the soft-margin formulation from above becomes immediately apparent. The log-likelihood function corresponds to the norm penalty on the weight vector $\boldsymbol{w}$ whereas the large margin term corresponds to the slack penalty. Indeed, we can interpret the first term as being a regularizer that ensures a probabilistic interpretation of the model while the second term tries to model an accurate classifier. The parameter λ governs the trade-off between the likelihood and the large margin term and hence controls the importance of the generative and the discriminative part respectively. If $\lambda = 0$, the log-likelihood objective is recovered which we can traditionally optimize with the EM algorithm described in Section 2.3.4. As λ goes to infinity, we essentially neglect the likelihood and merely optimize the margin.

One obvious difference to the SVM formulation is due to the γ parameter in place of the constant 1 from above. In the case of SVMs we saw that this constant arises due to the particular normalization of the weight vector. When minimizing the norm of the weight vector

$\boldsymbol{w}$ subject to the given constraints, it happens that the size of the margin is automatically one unit in terms of the decision rule $\boldsymbol{w} \cdot \boldsymbol{x} + b$. Samples where this value is greater than one will not be penalized by the hinge function. However, in our scenario there is no such normalization that allows us to use a fixed value. Instead, we have to introduce a hyperparameter γ which we refer to as the *desired margin*. This causes examples with a lower probabilistic margin than specified by γ to incur a loss. Similarly to the soft margin SVM, the large margin term can be seen as a measure of data misfit and we can come up with an interpretation related to that of SVMs. Data points which are assigned the wrong class incur at least a loss of γ. Hence, this sum divided by γ can be seen as an upper bound on the number of incorrectly classified examples, i.e. $\sum_{n=1}^{N} \mathrm{h}(\gamma - \beta_{n,c}(\boldsymbol{\theta}))/\gamma$.

Of course, since we are in the domain of large margin learning, a high margin is desired. However, we now demonstrate by an example that a very large margin can cause serious problems. Suppose there are two data examples $x_1 = 1$ and $x_2 = -1$ of distinct classes. For each class we have a single Gaussian defined by $\boldsymbol{\theta}_c = (\mu_c, \sigma_c^2)$ with $c \in \{1, 2\}$ and the class priors are equal, i.e. $\pi_1 = \pi_2$. The probabilistic margin in terms of the parameters is then given by

$$\beta = \log \frac{\frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{(x_1-\mu_1)^2}{2\sigma_1^2}\right)}{\frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{(x_1-\mu_2)^2}{2\sigma_2^2}\right)} + \log \frac{\frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{(x_2-\mu_2)^2}{2\sigma_2^2}\right)}{\frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{(x_2-\mu_1)^2}{2\sigma_1^2}\right)} = 2\left(\frac{\mu_1}{\sigma_1^2} - \frac{\mu_2}{\sigma_2^2}\right). \tag{3.14}$$

Given that $\mu_1 > 0$ and $\mu_2 < 0$, the margin can be made arbitrarily large by making the variances σ_1^2 and σ_2^2 small. This would result in degenerated Gaussians that collapse onto a single point. This problem can be solved by introducing a penalty on the trace of the inverse covariance matrix as in [15]. Nevertheless, one can still make the margin arbitrarily large by increasing μ_1 and decreasing μ_2, even if the variances are not small. In this scenario, the means would be far away from the data which is typically not a desired behavior. In either case, the decision boundary of the resulting classifier might not change. However, the probabilistic interpretation of the model could suffer severely. The problem of obtaining the highest possible margin is therefore ill-posed.

This justifies why we aim to achieve a desired margin rather than making the margin as large as possible. It turns out that tuning the parameter γ is crucial in order to achieve good results. If γ is chosen too large, examples that are already classified correctly with high certainty in terms of their class posterior probability might incur a loss. It is desired that only points close to the decision boundary are penalized. This is usually achieved for lower values of γ. However if γ is too small, the decision boundary might not get pushed away from the data examples. Hence, one usually seeks for some intermediate value that achieves the highest classification performance without severely degrading the likelihood. If the desired margin is carefully chosen and it is mainly those data points near the decision boundary that are affected by the loss, the optimization procedure will model the decision boundary well.

There is no general rule for how to chose the values of γ and λ. They rather need to be tuned individually for each application. Experiments have shown (cf. Chapter 4) that increasing either of the two parameters decreases the probabilistic interpretation of the model. It is often helpful to observe how the log-likelihood of solutions changes for different parameters. If the log-likelihood decreases drastically, one might think about decreasing any of the two values. Small changes in the log-likelihood could indicate that the parameters are chosen too conservative and can be increased instead.

Finally, we want to discuss the effects of the hinge function. The log-probabilistic margin of data examples that do not fulfill the desired margin is penalized linearly. This means that there is effectively no limit on the penalty of a single data example. It is especially outliers, which appear far on the wrong side of the decision boundary, that incur a high loss. As a consequence, such points are always considered in the large margin term and therefore influence the decision

boundary. In many cases better results would be achieved if these examples were simply removed from the data set. The problem arises because such outliers can lead the learning procedure into the wrong direction, since correcting a single outlier might reduce the loss more than correcting many examples that are only slightly on the wrong side of the decision boundary. On the other hand, outliers which lie far on the right side of the decision boundary are naturally handled by the hinge function, since they are not penalized at all.

Different solutions to the problem of outliers have been proposed. First, there is always the possibility to preprocess the data in such a way that outliers are detected and removed by a separate algorithm. A reduced set is then handed to the actual algorithm. For example, one could estimate a density for the given data and remove those points that have a relatively low probability compared to the rest of the data. Another possible way to handle outliers is to change the algorithm such that outliers are implicitly handled. Sha [15] estimates ML parameters and computes the penalty of each example using this model. The hinge function of each example is then weighted according to the inverse of its penalty. The idea is that the reduction in the loss of an outlier should be equal to that of a slightly misclassified example and that the learning procedure tends to favor data points near the decision boundary. In the case of soft margin SVMs, the hinge function can be replaced by other loss functions to reduce the effect of outliers. In particular the *ramp function*, given by

$$r_\nu(t) = \begin{cases} 0 & t \leq 0 \\ t & 0 < t \leq \nu \\ \nu & \nu < t \end{cases}, \tag{3.15}$$

can be used to reduce the impact of badly positioned examples [25].[9] The ramp function introduces a new hyperparameter ν which determines the maximum possible loss. Examples which are further away will not incur an arbitrary high loss and, therefore, have less effect on the learning procedure. Unfortunately, the ramp function is, unlike the hinge function, non-convex. Nevertheless, our problem is non-convex anyway and we believe the positive aspects to dominate.

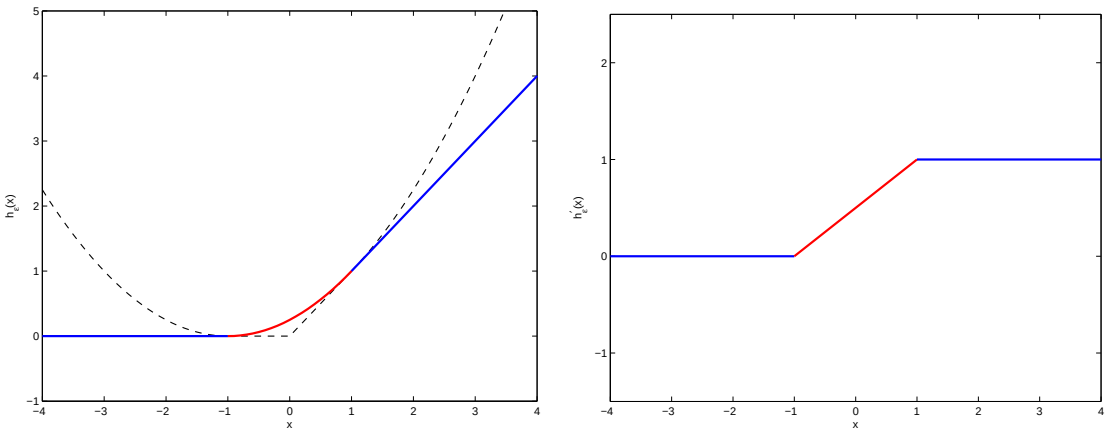
3.2 Optimizing the Hybrid GMM Problem

In the previous section we derived the hybrid generative-discriminative objective for GMMs. In this section we describe how this objective can be optimized. By substituting the definition of the log-probabilistic margin in (3.13), the hybrid objective is given by

$$l_{hybrid}(\boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p\left(\boldsymbol{x}_n, c_n|\boldsymbol{\theta}\right) + \lambda \sum_{n=1}^{N} \mathrm{h}\left(\max_{c \neq c_n} \left(\gamma - p\left(\boldsymbol{x}_n, c_n|\boldsymbol{\theta}\right) + p\left(\boldsymbol{x}_n, c|\boldsymbol{\theta}\right)\right)\right). \tag{3.16}$$

Many general gradient based algorithms, such as gradient descent, require the given function to be smooth, i.e. the gradient exists everywhere and is continuous. Especially convergence properties of many algorithms depend on the smoothness of the function. However, the objective given in (3.16) is clearly not differentiable due to the non-differentiability of the hinge and the maximum function. After demonstrating how the hybrid objective (3.16) can be translated into a smooth objective, we present ways to incorporate the constraints imposed on the parameters in the objective in order to arrive at an unconstrained optimization problem. We show the gradient of the smoothed hybrid objective and give detailed interpretations thereof. Finally, we propose methods to optimize the smooth hybrid objective.

[9] In fact, this helps reducing the number of support vectors. This effectively reduces the complexity of the model, since SVMs are defined by the support vectors.



(a) (b)

Figure 3.2: (a) shows the soft-hinge function for $\epsilon = 1$. The blue part is from the original hinge function and the red part is the quadratic part used to smooth the kink of the hinge function. (b) shows the derivative of the soft-hinge function.

3.2.1 Smoothing the hybrid objective

As a first step to make the hybrid objective amenable to optimization algorithms, we introduce smooth versions of the hinge and the maximum function. The hinge function is non-smooth at the kink at zero where the function starts to grow linearly. We can smooth the function by fitting a small quadratic piece around the non-differentiable part. We refer to the smoothed version as the *soft-hinge function*. This function is parametrized by ϵ which determines how large the area will be that is affected by the quadratic part. The soft-hinge function with parameter ϵ is defined by

$$\mathrm{h}_\epsilon(t) = \begin{cases} 0 & t < -\epsilon \\ t & t > \epsilon \\ \frac{(t+\epsilon)^2}{4\epsilon} & \text{otherwise} \end{cases} \tag{3.17}$$

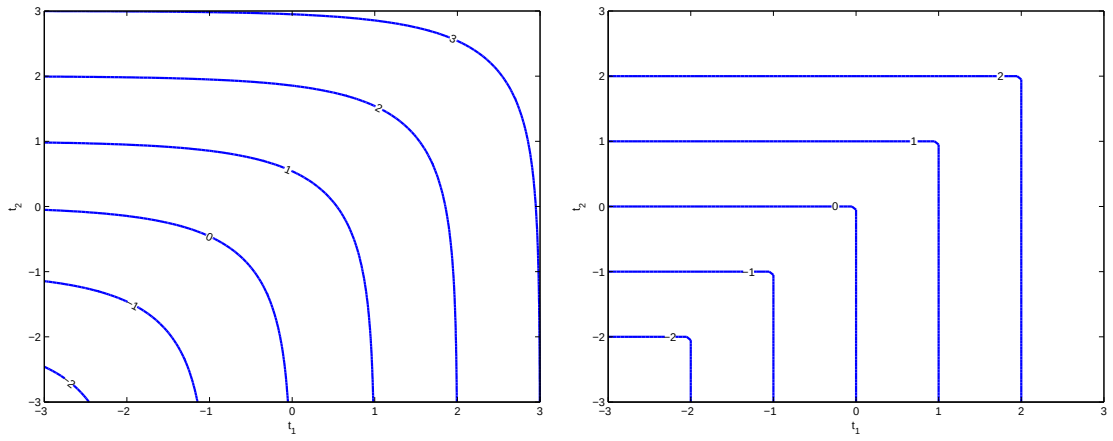
and its derivative is given by

$$\mathrm{h}'_\epsilon(t) = \begin{cases} 0 & t < -\epsilon \\ 1 & t > \epsilon \\ \frac{(t+\epsilon)}{2\epsilon} & \text{otherwise} \end{cases} . \tag{3.18}$$

The soft-hinge function and its derivative are illustrated in Figure 3.2. Outside the interval $[-\epsilon, \epsilon]$ the soft-hinge function equals the original hinge function. Inside this interval a piece of a parabola is used to smooth the kink. As the parameter ϵ goes to zero the original hinge function is recovered. The derivative of the hinge function is zero for $t < 0$ and one for $t > 1$ and it does not exist for $t = 0$. The soft-hinge function linearly interpolates between zero and one in the interval $[-\epsilon, \epsilon]$. Note that the ramp function (3.15) can be smoothed similarly.

The next part of the hybrid objective that needs to be smoothed is the maximum function. We approximate the maximum function with the *soft-max* variant used in [2], which is defined by

$$\underset{t_1,\ldots,t_L}{\operatorname{smax}} = \frac{1}{\eta} \log \sum_{i=1}^{L} \exp(\eta t_i). \tag{3.19}$$



(a) (b)

Figure 3.3: (b) shows the contours of the soft-max function for $\eta = 1$ in two dimension. (b) shows the contours of the maximum function. The soft-max function approximates the maximum function well in regions where t_1 differs significantly from t_2. There is a small discrepancy at $t_1 \approx t_2$ which is the part of the maximum function that needs to be smoothed.

The partial derivative of the soft-max function with respect to t_j is given by

$$\frac{\partial \operatorname{smax}_{t_1,\ldots,t_L}}{\partial t_j} = \frac{\exp(\eta t_j)}{\sum_{i=1}^{L} \exp(\eta t_i)}. \tag{3.20}$$

The parameter $\eta > 0$ governs the smoothness of the soft-max function. As η goes to infinity, the common maximum function is recovered. For negative values of η the function approximates the minimum function. The idea of the soft-max function is as follows: The arguments are scaled with a positive value and put into the exponential function. This causes the gap between individual values to be increased by orders of magnitudes. The following summation will then be affected mostly by the largest term. Taking the logarithm and normalizing with the inverse of η recovers an approximation of the largest value. There is also an intuitive meaning for the gradient of the soft-max function. In the case of the common maximum function, one can think of a zero-one vector where the largest entry is indicated by a one while all other entries are zero. The gradient of the soft-max function can then be seen as an approximation to this vector. Clearly all entries are non-negative and sum up to one. The largest argument will also produce the largest value in the corresponding entry of this vector. The same reasoning why the soft-max function approximates the maximum function applies to why the gradient approximates the zero-one vector representation. Actually, if the gap between the largest and the second largest value is high enough, the entry of the largest value will be close to one whereas all other entries will be close to zero. Figure 3.3 shows the contours of the soft-max and the maximum function respectively.

Using smoothed approximations leads to a smoothed hybrid objective function given by

$$\tilde{l}_{hybrid}(\boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(\boldsymbol{x}_n, c_n|\boldsymbol{\theta}) + \lambda \sum_{n=1}^{N} \mathrm{h}_\epsilon \left(\operatorname*{smax}_{c \neq c_n} (\gamma - p(\boldsymbol{x}_n, c_n|\boldsymbol{\theta}) + p(\boldsymbol{x}_n, c|\boldsymbol{\theta})) \right). \tag{3.21}$$

3.2.2 Incorporating the constraints into the objective

Recall that the parameters are subject to several constraints. On the one hand, the component priors of each class need to be non-negative and sum up to one. On the other hand, the covariance

matrices need to be positive semidefinite. We will now demonstrate how this constraints can be avoided by reformulating the objective function. To handle the non-negativity and sum-to-one constraints, we introduce a new unconstrained vector $\boldsymbol{z} = (z_1, \ldots, z_K)$ for each set of component priors $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)$. Each component prior α_k can then be expressed as a function of $\boldsymbol{z}$ by

$$\alpha_k(\boldsymbol{z}) = \frac{\exp(z_k)}{\sum_{k'=1}^{K} \exp(z_{k'})}. \tag{3.22}$$

The new values α_k satisfy the non-negativity and sum-to-one constraints due to the non-negativity of the exponential. A disadvantage of this method is that the newly introduced variables z_k are not unique. Adding an arbitrary constant to the values of z_k for $k = 1, \ldots, K$ does not change the resulting values of α_k. Nevertheless, this is no problem when searching for good model parameters.

To handle the constraints on the covariance matrices we first restrict ourselves to diagonal covariance matrices. The constraints for positive semidefiniteness are then simplified to non-negativity constraints on the entries on the diagonal. Since it is common to fix a small positive minimal value for the entries on the diagonal, we present a way how constraints of the more general form $\sigma \geq \zeta$ can be integrated into the objective function. Again, we introduce for each diagonal entry σ a new variable z. The variable σ can be expressed as a function of z by

$$\sigma(z) = \zeta + \exp(z). \tag{3.23}$$

Again, the non-negativity of the exponential shows that the constraint $\sigma \geq \zeta$ is satisfied by every z. Using these reformulations, it is possible to optimize the hybrid objective with algorithms for unconstrained optimization problems. Experiments have shown (cf. Chapter 4) that working with the unconstrained problem outperforms constrained optimization techniques in terms of needed time and achieved objective function value.

3.2.3 Interpretation of the gradient

We now investigate the gradient of the hybrid objective to gain further insights into the problem. Most optimization procedures are guided by the gradient of the given objective. The derivation of the gradient can be found in Appendix A and we will only state the results here. Notice that the log-likelihood and the large margin part of the hybrid objective are composed as a sum over all data examples. Since the gradient is a linear operator, we can compute it for each example $(\boldsymbol{x}_n, c_n)$ separately and the overall result will be the sum of the individual gradients.

Likelihood term

We begin with the gradient of the log-likelihood term. By assuming that the parameters of each class conditional likelihood $p(\boldsymbol{x}|\boldsymbol{\theta}_c)$ are pairwise disjoint, i.e. $\boldsymbol{\theta}_c \cap \boldsymbol{\theta}_{c'} = \emptyset$ for $c \neq c'$, it is straightforward to show that the gradient of the log-likelihood term with respect to $\alpha_{c,k}$, $\boldsymbol{\mu}_{c,k}$ and $\boldsymbol{\Sigma}_{c,k}$ is zero if $c \neq c_n$. Hence, we assume for now that $c = c_n$. The gradient of the log-likelihood regarding the data example $(\boldsymbol{x}_n, c_n)$ is then given by

$$\frac{\partial \log p(\boldsymbol{x}_n, c_n|\boldsymbol{\theta})}{\partial \alpha_{c,k}} = p(k|c, \boldsymbol{x}_n, \boldsymbol{\theta}) \frac{1}{\alpha_{c,k}} \tag{3.24}$$

$$\frac{\partial \log p(\boldsymbol{x}_n, c_n|\boldsymbol{\theta})}{\partial \boldsymbol{\mu}_{c,k}} = p(k|c, \boldsymbol{x}_n, \boldsymbol{\theta}) \boldsymbol{\Sigma}_{c,k}^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_{c,k}) \tag{3.25}$$

$$\frac{\partial \log p(\boldsymbol{x}_n, c_n|\boldsymbol{\theta})}{\partial \boldsymbol{\Sigma}_{c,k}} = p(k|c, \boldsymbol{x}_n, \boldsymbol{\theta}) \frac{1}{2} \left(\boldsymbol{\Sigma}_{c,k}^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_{c,k}) (\boldsymbol{x}_n - \boldsymbol{\mu}_{c,k})^T \boldsymbol{\Sigma}_{c,k}^{-1} - \boldsymbol{\Sigma}_{c,k}^{-1} \right) \tag{3.26}$$

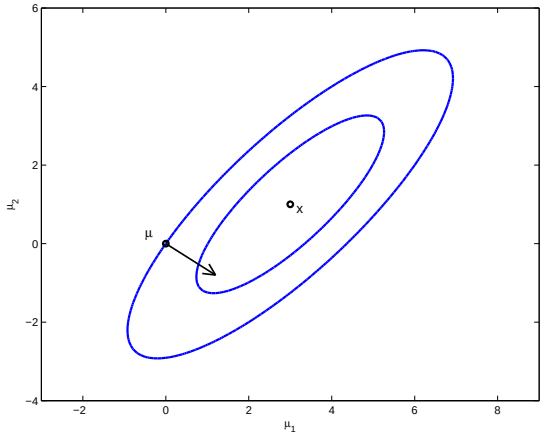


Figure 3.4: Illustration of the gradient of the Mahalanobis distance with respect to the mean. The negative gradient is perpendicular to the contour line of the Mahalanobis distance and points into the direction of steepest descent rather than towards the data example $\boldsymbol{x}$. Notice that the direction of the gradient is mostly governed by the direction of least variability.

with

$$p\left(k|c,\boldsymbol{x}_n,\boldsymbol{\theta}\right) = \frac{\alpha_{c,k}\mathcal{N}\left(\boldsymbol{x}_n|\boldsymbol{\mu}_{c,k},\boldsymbol{\Sigma}_{c,k}\right)}{\sum_{k'=1}^{K_c}\alpha_{c,k'}\mathcal{N}\left(\boldsymbol{x}_n|\boldsymbol{\mu}_{c,k'},\boldsymbol{\Sigma}_{c,k'}\right)}. \tag{3.27}$$

Note that the term $p\left(k|c,\boldsymbol{x}_n,\boldsymbol{\theta}\right)$ is common to all expressions. This term is the probability of component k having generated data example $\boldsymbol{x}_n$ with the current parameters by only considering the components of class c. It is intuitive to think that the gradient is computed for each component and the results are weighted according to the probability of the component being responsible for that example. Therefore, components with a low posterior probability are effectively neglected when computing the gradient of the corresponding example. The gradient can best be interpreted by thinking about what happens during a gradient descent step where we move into the direction of the negative gradient.[10] The component priors are increased by the inverse of their current value. If $\alpha_{c,k}$ is large, the inverse of $\alpha_{c,k}$ will be small and its value will only be slightly increased. If we ignore the weighting with the component posteriors, this would enforce all the component priors to become equal, since large values are increased more than small values. These values are then weighted with the component posteriors. In practice the component posteriors often have most of its mass on one or only a few components such that it is only the most probable components which are affected by a single example.

The mean is updated according to the vector $\boldsymbol{s} = \boldsymbol{\Sigma}_{c,k}^{-1}\left(\boldsymbol{x}_n - \boldsymbol{\mu}_{c,k}\right)$. $\boldsymbol{s}$ points into the direction of the negative gradient of the Mahalanobis distance with covariance matrix $\boldsymbol{\Sigma}_{c,k}$. The effect of the gradient with respect to the mean is best illustrated in case of a diagonal covariance matrix, since the inverse of a diagonal matrix is obtained by simply inverting each element on the diagonal. If the covariance matrix is the identity matrix, the negative gradient points simply into the direction of the data example. This would cause the mean to move directly towards the data example. In case of a general diagonal covariance matrix each dimension is weighted differently. This behavior stems from the Mahalanobis distance where a discrepancy in dimensions of high variability is penalized less than in dimensions where the variance is small. Hence, the correction of the mean value in dimensions of low variability has more impact on the Mahalanobis distance than in dimensions of high variability. In case of a full covariance matrix the effect is similar. Here the multiplication with the inverse covariance matrix also accounts for the correlations between the dimensions. Figure 3.4 illustrates the negative gradient with

[10] Note that we are optimizing the *negative* log-likelihood and thus the negative gradient is already given in Equations (3.24), (3.25) and (3.26).

respect to the mean.

The gradient with respect to the covariance matrix is governed by two terms. Notice that the first term can be written as $\boldsymbol{s}\boldsymbol{s}^T$ where $\boldsymbol{s}$ is the vector that governs the update of the mean. As we have seen, the vector $\boldsymbol{s}$ is related to the Mahalanobis distance with respect to the covariance matrix. Hence, the further a data example is away from the mean in terms of the Mahalanobis distance, the larger the covariance matrix will be after the update step. The second term $\boldsymbol{\Sigma}_{c,k}^{-1}$ arises due to the normalizing constant of the Gaussian. The normalizing constant of a Gaussian becomes larger as the variance becomes smaller. Hence, this term always causes the variance to decrease. However, the normalizing constant can not be made arbitrarily large since otherwise the Mahalanobis distance will grow simultaneously. Therefore, one seeks for a tradeoff between both terms. Only in case the term $\boldsymbol{s}\boldsymbol{s}^T$ vanishes, which happens if the mean lies exactly on the data point, the variance can be made arbitrarily small. This would result in a singularity as discussed in Section 2.3.3.

Large margin term

The gradient of the large margin term with respect to the model parameters $\boldsymbol{\theta}_c$ of class c is given by

$$\frac{\partial\, \mathrm{h}_\epsilon\left(\mathrm{smax}_{c' \neq c_n}(\gamma - \beta_{n,c'})\right)}{\partial \boldsymbol{\theta}_c} = \mathrm{h}_\epsilon'\left(\max_{c' \neq c_n}\left(\gamma - \beta_{n,c'}\right)\right) \cdot \begin{cases} \frac{\exp(-\eta\beta_{n,c})}{\sum_{c' \neq c_n}\exp(-\eta\beta_{n,c'})} \frac{\partial}{\partial \boldsymbol{\theta}_c} \log p\left(\boldsymbol{x}_n|\boldsymbol{\theta}_c\right) & c \neq c_n \\ -\frac{\partial}{\partial \boldsymbol{\theta}_c} \log p\left(\boldsymbol{x}_n|\boldsymbol{\theta}_{c_n}\right) & c = c_n \end{cases} \tag{3.28}$$

The gradient involves the multiplication with the derivative of the soft-hinge function (3.18). Recall that the derivative of the soft-hinge h_ϵ' is zero for values smaller than epsilon, one for values larger than epsilon and linearly interpolates between zero and one for intermediate values. This causes data examples whose margin is larger than $\gamma + \epsilon$ to not influence the gradient at all. This is a desired behavior, since this is reminiscent of SVMs where only the support vectors determine the decision boundary while all other data examples are effectively neglected. We have to distinguish now between two cases. On the one hand, if the gradient is computed with respect to the true class c_n, the result equals the gradient of the negative log-likelihood. On the other hand, if the gradient is computed with respect to some other class $c \neq c_n$, the result differs in sign and involves an additional factor due to the partial derivative of the soft-max function. This additional factor weights the gradient of each class according to the probabilistic margin. It is useful to consider the case where η goes to infinity so that only the weight of the best competitor class is one and all other weights are zero. The gradient with respect to the best competitor class is then computed by evaluating the gradient of the log-likelihood function for the best competitor class. As a consequence, the gradient of the large margin term causes the parameters of the true class to move towards the ML solution and simultaneously the parameters of the best competitor class to move away from the ML parameters of the true class. For instance, this would cause the component means to be attracted by examples of the true class while at the same time the means tend to move away from data examples of other classes. For moderate values of η not only the parameters of the best competitor class are affected. The parameters of other classes are affected as well and the results are weighted accordingly.

3.2.4 Optimization procedure

We propose to optimize the smoothed hybrid generative-discriminative objective function with general purpose optimization algorithms described in Section 2.5. ML solutions, computed with

the EM algorithm, turned out to provide good initial solutions for iterative procedures. Recall that the ML objective is recovered in the case of $\lambda = 0$. If the generative-discriminative trade-off parameter λ is only slightly increased we can also expect optimal solutions to change only slightly. This also justifies intuitively why ML parameter might be a good starting point.

We suggest to use the BFGS algorithm. The number of parameters to optimize is given by

$$C + \sum_{c=1}^{C} K_c + 2K_c \cdot D. \tag{3.29}$$

For applications with hundreds of features and many classes, the number of components can become large even for moderate numbers of components per class. Hence, we suggest to use the limited memory variant L-BFGS when storing the approximated inverse Hessian explicitly becomes intractable. If the techniques described in Section 3.2.2 are used to make the problem unconstrained, it is straightforward to compute the gradient with respect to the newly introduced variables $\boldsymbol{z}$ using the chain rule.

3.3 Hybrid GMMs and Missing Data

As a generative model, GMMs are naturally capable of dealing with missing data. In this section we present how to deal with missing features and missing labels using GMMs.

3.3.1 GMMs and missing features

As discussed in Section 2.4, the MAR property in the given data suffices that marginalization over the missing features is the valid operation to perform both ML estimation and classification according to the class with the highest posterior probability. Let $\boldsymbol{x} = (\boldsymbol{x}^o, \boldsymbol{x}^m)$ where $\boldsymbol{x}^o$ denotes the observed features and $\boldsymbol{x}^m$ denotes the missing features. Given a Gaussian distribution with parameters

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}^o \\ \boldsymbol{\mu}^m \end{pmatrix}, \qquad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}^{oo} & \boldsymbol{\Sigma}^{om} \\ \boldsymbol{\Sigma}^{mo} & \boldsymbol{\Sigma}^{mm} \end{pmatrix}, \tag{3.30}$$

where superscript o denotes the parameters corresponding to the observed features and superscript m denotes the parameters corresponding to the missing features and the covariance matrix $\boldsymbol{\Sigma}$ is decomposed into blocks. Then the marginal distribution over the observed features is also a Gaussian distribution defined by the parameters $\boldsymbol{\mu}^o$ and $\boldsymbol{\Sigma}^{oo}$ [3]. In other words, marginalization over the missing features is simply performed by taking those entries of the mean that coincide with the observed features and removing those columns and rows of the covariance matrix corresponding to the missing features. Using the linearity of the integral operator, it is now straightforward to integrate out the missing features of a GMM. The marginal distribution of a GMM is given by

$$\int \sum_{k=1}^{K} \alpha_k \mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right) d\boldsymbol{x}^m = \sum_{k=1}^{K} \alpha_k \int \mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right) d\boldsymbol{x}^m \tag{3.31}$$

$$= \sum_{k=1}^{K} \alpha_k \mathcal{N}\left(\boldsymbol{x}^o|\boldsymbol{\mu}_k^o, \boldsymbol{\Sigma}_k^{oo}\right). \tag{3.32}$$

Given that the MAR condition holds, data examples with missing features can thus be classified by simply ignoring the missing features. To estimate ML parameters for a GMM, the EM

algorithm can be extended to handle missing features [26] or general optimization algorithms can be used to optimize the marginal likelihood directly.

3.3.2 Semi-supervised learning with hybrid GMMs

The semi-supervised learning problem with GMMs is more involved. Of course, there is always the possibility to use general semi-supervised techniques, like self-learning, which are independent of the underlying supervised algorithm. In the case of a generative model one can also optimize both the joint likelihood $p(\boldsymbol{x}, c)$ of the labeled data and the marginal distribution $p(\boldsymbol{x})$ of the unlabeled data. Using the sum-rule of probability, the log-likelihood function of the marginal distribution of a data example $\boldsymbol{x}$ is given by

$$\log p(\boldsymbol{x}|\boldsymbol{\theta}) = \log \sum_{c=1}^{C} p(\boldsymbol{x}|\boldsymbol{\theta}_c)\, \pi_c. \tag{3.33}$$

The gradient of the log-likelihood with respect to the parameter $\boldsymbol{\theta}_c$ of class c are given by

$$\frac{\partial \log p(\boldsymbol{x}|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_c} = p(c|\boldsymbol{x}, \boldsymbol{\theta})\, \frac{\partial \log p(\boldsymbol{x}|\boldsymbol{\theta}_c)}{\partial \boldsymbol{\theta_c}}. \tag{3.34}$$

Details about the calculation of the gradient are provided in Appendix A. This is an intuitive result, since it equals the gradient of the log-likelihood function in case of known class labels with an additional weight according to the class posterior probability. These weights can also be seen as probabilistic labels of the unlabeled data. If the model with the current parameters is very certain about the class of an unlabeled data example and its class posterior probability is close to one, this example is effectively treated as being labeled with that class. We now extend the hybrid objective to include a term for the marginal distributions over the unlabeled data in the likelihood term. Given a set of labeled data $\{(\boldsymbol{x}_1^l, c_1), \ldots, (\boldsymbol{x}_{N_l}^l, c_{N_l})\}$ and a set of unlabeled data $\{\boldsymbol{x}_1^u, \ldots, \boldsymbol{x}_{N_u}^u\}$, we define the semi-supervised hybrid objective as

$$l_{ssl}(\boldsymbol{\theta}) = -\sum_{n=1}^{N_l} \log p\left(\boldsymbol{x}_n^l, c_n|\boldsymbol{\theta}\right) - \sum_{n=1}^{N_u} \log p(\boldsymbol{x}_n^u|\boldsymbol{\theta}) + \lambda \sum_{n=1}^{N_l} \mathrm{h}(\gamma - \beta_n(\boldsymbol{\theta})). \tag{3.35}$$

This objective uses the unlabeled data as additional information about the underlying marginal distribution $p(\boldsymbol{x})$, but does not incorporate it in the large margin term. The reason why the unlabeled data is excluded from the large margin term is that if predictions on unlabeled data are made with poor precision, the unlabeled data will have a negative impact on the performance, similar as in self-learning. Of course, this effect might also happen by optimizing the marginal likelihood alone. However, since the large margin term is mainly responsible for achieving a good classification performance, we expect this effect to be even worse. Hence, we stick to this more conservative strategy of using the unlabeled data only in the likelihood term. This semi-supervised learning strategy has therefore more of a generative than a discriminative character. The labeled data is best seen as points in the feature space where the decision boundary gets pushed away while at the same time the log-likelihood over both the labeled and unlabeled data is used to regularize the model parameters.

It might also be useful to introduce another trade-off parameter that governs the influence of the log-likelihood term of the unlabeled data to mitigate the possible negative effect of unlabeled data examples [27]. Especially in the case of pure ML estimation, if the number of unlabeled data examples N_u exceeds the number of labeled data examples N_l by orders of magnitudes, the influence of the labeled data becomes weak and the parameters are solely determined by the unlabeled data. However, the trade-off parameter for the generative and discriminative part λ

implicitly tunes the importance of the labeled data and experiments (cf. Chapter 4) support the fact that there is no need for another trade-off parameter. This is especially important to reduce the computational effort needed to perform grid search on the hyperparameter space, since the number of training runs to perform grows exponentially in the number of hyperparameters.

We suggest to use the same techniques for optimization as described in Section 3.2.4. When computing the initial solution for the quasi-Newton algorithm we propose to use the labeled data only. The experiments in Chapter 4 show that objective (3.35) is capable of improving the performance of a purely supervised model.

4 Experiments

This chapter shows experiments with the hybrid GMM classifier on both synthetic and real world data. We start in Section 4.1 with a brief overview about how we carried out the experiments. Section 4.2 shows experiments conducted on synthetic data. We focus on two-dimensional data in order to visualize the capabilities of the model. Furthermore, these examples are intended to demonstrate how the hyperparameters affect the resulting classifier. In Section 4.3 we present the results obtained on real world data and compare them with other state of the art classifiers. Section 4.4 presents results in the presence of missing data. We restrict ourselves to missing features at classification time and missing labels at training time.

4.1 Setup

This section describes the setup for the experiments in the subsequent sections. All optimization algorithms concerning the hybrid generative-discriminative GMM classifier are implemented in Matlab. Function and gradient evaluation was the most time-consuming part and has therefore additionally been implemented in C. We used the Matlab function *fmincon* to optimize the smoothed hybrid objective and existing implementations for the comparison models. SVMs were computed with libSVM [28] and the maximum margin (MM) and MCL optimized GMMs were computed with the available Matlab code from Pernkopf [10].

We optimize the smoothed hybrid objective with the BFGS quasi-Newton algorithm. For the larger data sets MNIST, TIMIT and USPS we use the limited memory variant L-BFGS due to the high amount of parameters to optimize. In these cases, the inverse Hessian is approximated with the 10 latest update steps. The algorithm stops if a stationary solution, where the largest entry of the gradient is sufficiently small, has been found. Depending on whether such a stationary solution can be computed in a decent amount of time, we stopped the optimization algorithm prematurely. For larger data sets we apply early stopping if the performance on a separate validation set does not increase for a fixed number of iterations. Furthermore, early stopping is performed if the generalization loss in (2.6) exceeds a prespecified value. In any case, the optimization procedure stops if a fixed number of maximum iterations has been reached. ML parameters, computed with the EM algorithm, serve as the initial parameters for the iterative quasi-Newton algorithm. For semi-supervised learning we computed the initial solution with the EM algorithm on the labeled data only. At least 10 random restarts are performed when

computing the initial solution with the EM algorithm. The class prior probabilities π_c are not modified during the optimization procedure. The class priors are either chosen to be uniform or to be the empirical prior given by (2.13). Furthermore, we restrict ourselves to diagonal covariance matrices such that the Gaussian density factorizes into a product of one-dimensional Gaussians.

The hybrid objective is determined by a number of hyperparameters. The numbers of components per class $K_1, \ldots, K_C$, the generative-discriminative trade-off parameter λ and the desired margin γ are determined with grid search. We restrict the number of components for each class to be equal, i.e. $K_c = K_{c'}$ for $c \neq c'$. Thus, we refer to the number of componets per class as K without the subscript indicating the corresponding class. Otherwise, the number of combinations to try would grow exponentially in the number of classes. The soft-hinge parameter ϵ is set to 0.1 and the soft-max parameter η is set to 10. We fixed a minimum positive value for the diagonal entries on the covariance matrix. This value was chosen by hand for each data set and we did not tune it for each application. Nevertheless, we observed that the model was on some sets sensitive to the choice of this value. When incorporating this value as constraints for each diagonal entry of the covariance matrices into the objective function, we observed difficulties with the Matlab function fmincon when these values were close to the minimal value. Hence, we transformed the constrained optimization problem into an unconstrained problem as described in 3.2.2. For the TIMIT data set we additionally tried some values for the ramp function parameter ν.

4.2 Experiments on Synthetic Data

In this section we demonstrate the behavior of the hybrid GMM classifier on synthetic data. We restrict ourselves to two-dimensional data in order to illustrate the classifier graphically.

4.2.1 Illustration of the hyperparameters

The first example, depicted in Figure 4.1, demonstrates how the hyperparameters λ and γ influence the classifier. The data comprises examples of two classes which are shown as red and blue points respectively. A GMM with two components has been learned for each class. The decision boundary is depicted as bold black line. Furthermore, the red and the blue line correspond to points which satisfy the desired margin with equality. Red points falling below the red line and blue points falling above the blue line violate the desired margin and are therefore penalized in the large margin term of the hybrid objective.

Figures 4.1(a), 4.1(b) and 4.1(c) show how the decision boundary and the desired margin change for different values of λ and fixed $\gamma = 1$. Figure 4.1(a) shows the pure generative solution for $\lambda = 0$ which is computed with the EM algorithm. Especially some of the blue points close to the decision boundary are misclassified with the ML solution. In Figure 4.1(b) some of the blue points are corrected at the cost of a single red misclassified example with a moderate value of λ. By increasing λ, more weight is put on achieving the desired margin for all data examples. Figure 4.1(c) shows the decision boundary for a high value of λ. Notice that all data examples are correctly classified and the desired margin is satisfied with equality by many data examples. The distance between the desired margin and the decision boundary has decreased drastically in order to achieve this. This is a sign that the probabilistic interpretation of the model has degraded since this happens if the entries in the covariance matrices decrease or the component means move away from the decision boundary. This shows that the model is sensitive to examples close to the decision boundary and that overfitting due to outliers can occur if λ is too large and γ is too small.

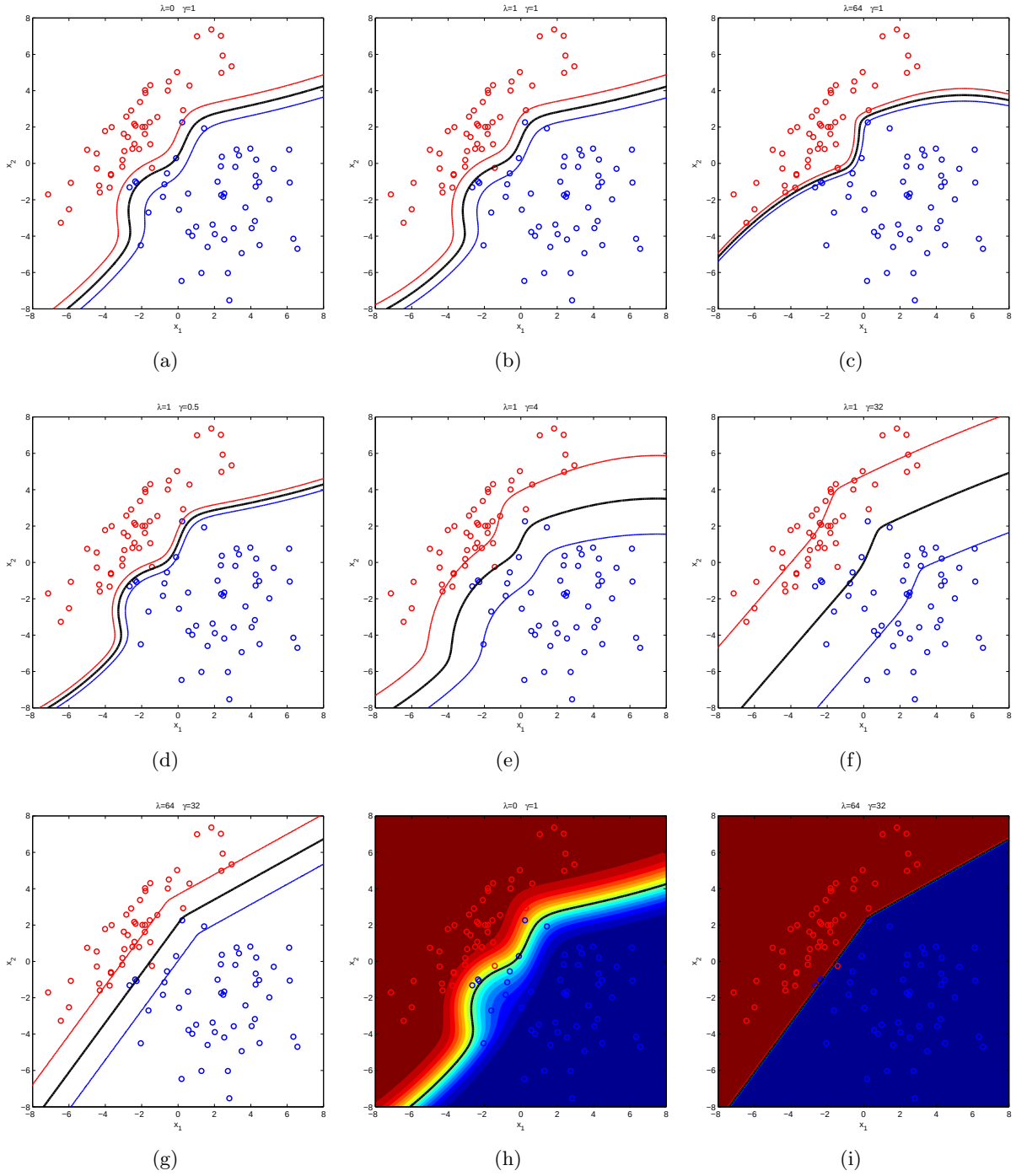
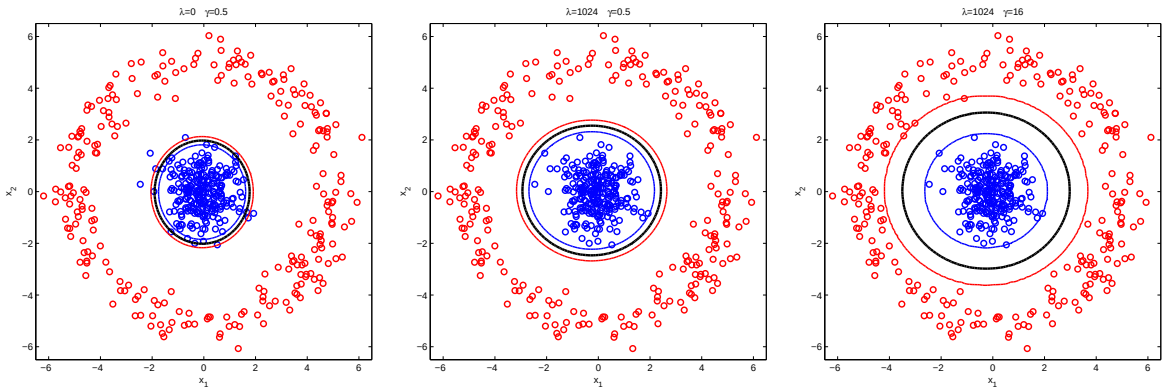


Figure 4.1: Illustration of how the hyperparameters λ and γ affect the model. Details are shown in the text.

Figures 4.1(d), 4.1(e) and 4.1(f) demonstrate how the decision boundary varies for different choices of the desired margin γ and fixed $\lambda = 1$. Figure 4.1(d) shows the decision boundary and the desired margin lines for a small γ. In this case, only points close to the decision boundary influence the large margin term. Figures 4.1(e) and 4.1(f) show the classifiers for an intermediate and a large value of γ respectively. The number of examples not satisfying the desired margin grows significantly and the decision boundary is determined by many data points rather than only a few examples close to it. This causes the decision boundary to be much smoother which is in many cases a good way to avoid overfitting and bad results due to outliers.

However, the relatively small choice of $\lambda = 1$ in Figure 4.1(f) results in many misclassified



(a) (b) (c)

Figure 4.2: Illustration of why choosing the desired margin λ too small might not push the decision boundary away from the data examples. Details are shown in the text.

examples. Figure 4.1(g) shows how the situation can be improved by a larger value of λ. The number of misclassified examples decreases and, unlike in Figure 4.1(c), single points have less impact on the decision boundary due to the high value of γ. On the other side, setting both λ and γ to a higher value typically harms the probabilistic interpretation of the model. This is illustrated in 4.1(h) and 4.1(i). The contours show the value of the class posterior probability $p\left(c|\boldsymbol{x}, \boldsymbol{\theta}\right)$ for $c = 1$. In the dark red and dark blue regions the classifier is confident about the true class of the inputs. In the brighter regions the classifier becomes less confident and in the yellow and green regions the class posterior probabilities are almost equal for both classes. In a binary classification task the decision boundary are those points for which the class posterior probability is 0.5. In the ML solution the class posterior probability usually changes smoothly for points close to the decision boundary as shown in Figure 4.1(h). If the values of λ and γ are too high, the probabilistic interpretation of the model could suffer severely. This results in the class posterior probability to change almost instantly to either zero or one when moving slightly away from the decision boundary.

4.2.2 Ring example

The next example, shown in Figure 4.2, illustrates that a too small value for the desired margin γ might not push the decision boundary away from the data examples. We generated a set of blue points which are surrounded by a set of red points. For each class we learned a single Gaussian. A single Gaussian is an appropriate model for the blue points but it would clearly need some more components to model the ring shaped set of red points. However, for this illustration we do not seek to model the data well. The ML solution, depicted in 4.2(a), corresponds to a ball-shaped Gaussian for each of the two classes with almost identical means. The covariance matrix for the blue points contains smaller entries than the covariance matrix for the red points. The ML solution classifies every red point correctly but misclassifies some of the blue points. In Figure 4.2(b) γ is chosen too small. All of the red points satisfy the desired margin and only a small fraction of the blue points are reached by the desired margin line. In this case, the resulting decision boundary for a high value of λ gets only slightly pushed away from the blue points and the red points have no impact at all. If the desired margin is chosen such that blue *and* red points fall inside the margin region, the decision boundary is placed in between the two sets. This is shown in 4.2(c). The resulting decision boundary in between the two classes is probably better suited for classification tasks.

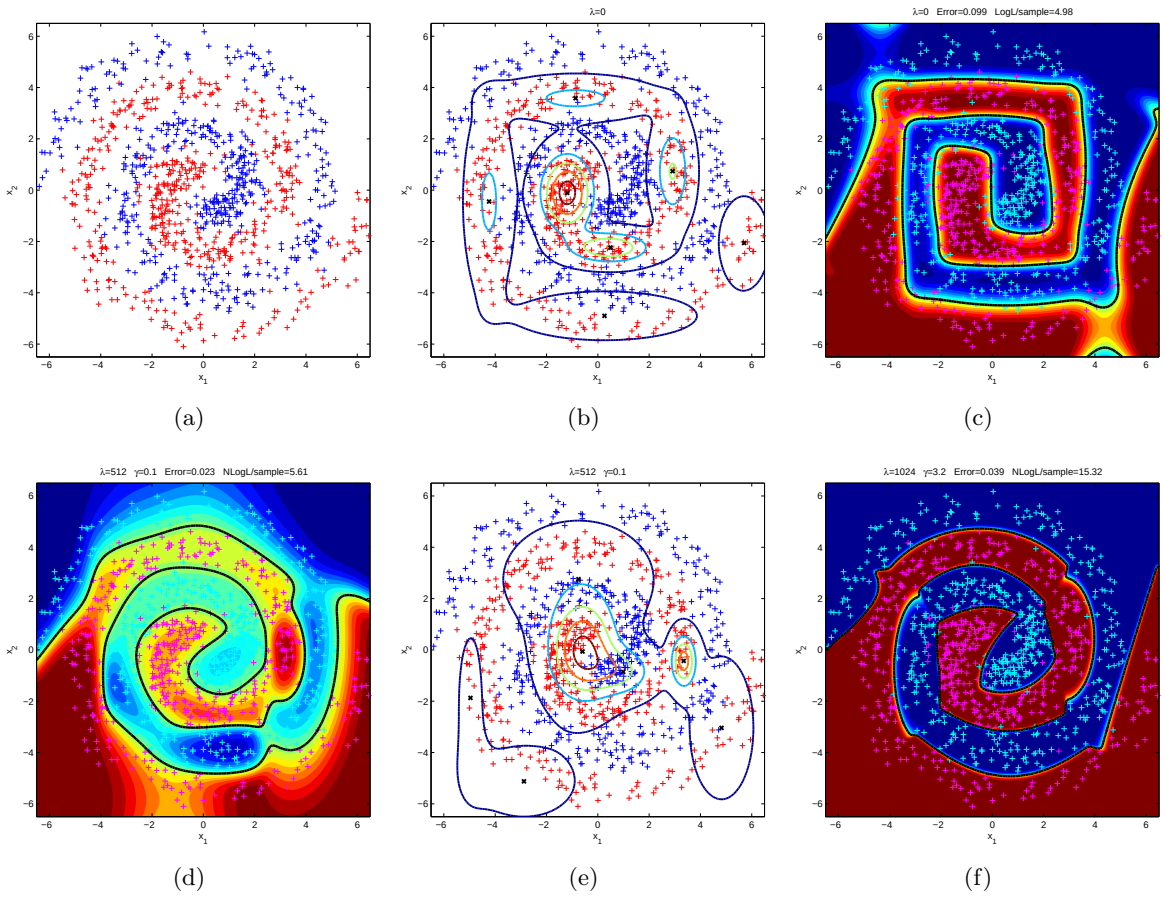
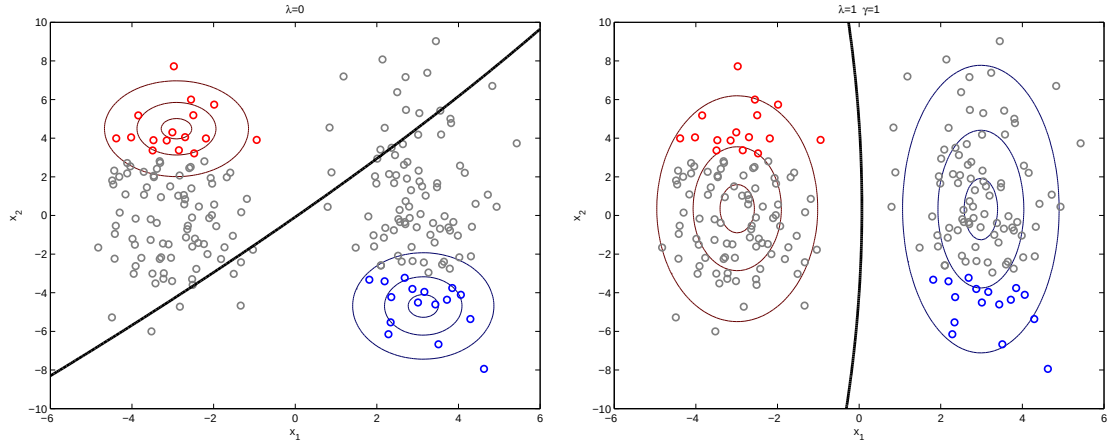


Figure 4.3: Illustration of the spiral example. Details are shown in the text.

4.2.3 Spiral example

The next example is inspired by an example from Pernkopf and Wohlmayr [10]. We generated 500 blue and 500 red points. The examples of each class are arranged in a spiral shape. The data set is depicted in 4.3(a). We choose the number of components for each class to be seven by inspection of the data set. Figure 4.3(b) shows the contours of the density for ML parameters. The means are shown as black crosses. The contribution of each component can easily be seen. The seven components roughly suffice to cover the whole spiral. Notice that all components are axis-aligned due to the restriction to diagonal covariance matrices. The class posterior probability and the decision boundary of the ML parameters are shown in Figure 4.3(c). We can see that the decision boundary is rectilinear due to the rectilinear contours of the class conditional densities and hence the approximation of the spiral is poor.

We tried several values for λ and γ and the best results in terms of training error are shown in Figure 4.3(d). These results were obtained for a low value of γ and a high value of λ. Such a parameter configuration focuses on modeling data points close to the decision boundary well. Since there are many data points close to the boundary, the risk of overfitting is relatively small. We can immediately see that the decision boundary fits now better to the data and the curved spiral shape has been recovered. On the other side, the red and the blue regions in the middle of the spiral, where we are quite sure about the true class label, have turned into green and yellow regions, were the class posterior probabilities do not suggest a high confidence when predicting the true class. Also the contours of the class conditional density for the red points, shown in Figure 4.3(e), have changed significantly. They do not cover the data points as accurate as the ML solution before. However, the probabilistic interpretation of this model is



(a) (b)

Figure 4.4: Illustration of how labeled data can help improving the decision boundary of a classifier. Details are shown in the text.

partly maintained. This can be seen by comparing the log-likelihood values of the ML solution and the hybrid solution. The negative log-likelihood has increased only slightly. Figure 4.3(f) shows the class posterior probability and the decision region for a high value of both λ and γ. In this case, the probabilistic interpretation has degraded severely which is indicated by the large increase in negative log-likelihood compared to the ML solution.

These examples show that no general rule about the optimal choice of the hyperparameters can be made and they rather need to be tuned separately for each application.

4.2.4 Semi-supervised learning example

The last example in this section illustrates the semi-supervised learning capabilities of the model. We generated two sets of points, red points on the left and blue points on the right with a clear gap in between. Then we removed the labels of the majority of the points such that only a small cluster of each color remains. We learned a single Gaussian for each class. The decision boundary and the contours of the class conditional densities of the ML solution learned on the labeled data alone are depicted in Figure 4.4(a). Clearly the decision boundary cuts through high density regions and the probability of the unlabeled data under this model is certainly poor. Figure 4.4(b) shows the results for $\lambda = 1$ and $\gamma = 1$ when also taking the unlabeled data into account. The decision boundary now lies in the low density region in between the two clusters. Furthermore, the contours of the class conditional densities also span the unlabeled data. The margin term, which is solely affected by the labeled data, prevents the class conditional densities from fitting unlabeled data from the other cluster.

4.3 Experiments on Real World Data

In this section we demonstrate the capabilities of the hybrid generative-discriminative GMM classifier on real world data. We start with a short description of the data sets being used. Furthermore, we explain which preprocessing steps have been applied to each of them. After introducing the detailed setup of the optimization procedure, we finally present the results. We compare the hybrid GMM classifier with ML optimized GMMs, MCL optimized GMMs, MM optimized GMMs and SVMs with linear and radial basis function (RBF) kernels. All GMM classifiers are restricted to diagonal covariance matrices.

4.3.1 Description of the data sets

We analyzed the performance of the classifiers on several different data sets.

MNIST

The *MNIST* data set is a commonly used data set for handwritten digit recognition [29]. Each data example is a 28×28 grayscale image. By treating each pixel as a feature, we can represent an image as a vector in $\mathbb{R}^{784}$. There are 10 classes, each corresponding to a particular digit. The data is split into 60000 training examples and 10000 test examples. We trained the models on the first 50000 examples of the training set and used the remaining 10000 examples as validation set for grid search. We conducted experiments with two different preprocessing steps:[11]

1. We removed those features that contained constant values over each of the 50000 training examples. This step affected mostly pixels located on the boundary and reduced the number of dimensions to 717. For SVMs, we additionally normalized the data by dividing each feature by 255. This normalization is a common preprocessing step on the MNIST data set and was used to decrease the training time of SVMs. We emphasize that this step was not necessary for the GMM classifiers.
2. We applied PCA to reduce the number of dimensions to 50. This step was performed in order to reduce the number of parameters substantially and to speed up the training process. The number of dimensions was chosen arbitrarily and not tuned by any means. The preserved variability of the PCA reduction according to (2.7) is 82.5%. For SVM training, the PCA features were additionally normalized to have zero mean and a standard deviation of one to speed up the training process. Again, this normalization was not necessary for the different GMM classifiers. In the remainder, we refer to the PCA reduced MNIST data set as *MNIST50*.

We evaluate the number of components $K \in \{2^0, \ldots, 2^5\}$ on MNIST and $K \in \{2^0, \ldots, 2^8\}$ on MNIST50. Due to the reduced number of parameters to estimate on MNIST50, we were able to investigate more components per class in the same time. The number of examples per class varies only slightly so that the empirical prior is close to uniform. Thus we selected a uniform distribution for the class prior probabilities π_c.

USPS

The *USPS* data set is another data set used for handwritten digit recognition [30]. The digits were extracted from the zip codes of mail envelopes. Each data example is a 16×16 grayscale image and can thus be represented as a vector in $\mathbb{R}^{256}$. Except for SVM training, where the data is normalized to have zero mean and a standard deviation of one, no preprocessing steps were performed. The data consists of 8000 training examples and 3000 test examples. We split the training data into 7000 training examples and 1000 validation examples for hyperparameter optimization. We tried the number of components $K \in \{2^0, \ldots, 2^5\}$ on the USPS data set. The distribution of the target values is even and thus the empirical prior is equal to a uniform distribution.

TIMIT

The *TIMIT* data set is used for speech classification [31]. Each example consists of 92 features and represents a phonetic segment which is classified to one of 39 phonemes. The data is split

[11] All preprocessing steps were computed solely with the training data. Validation and test data are then preprocessed according to the transformations obtained on the training set.

into 140173 training examples, 50735 validation examples (test) and 7211 test examples (core test). The training process is very demanding due to the large size of the training set, the large amount of classes and the high dimensionality. Hence, we evaluated the relatively small set of $K \in \{2^0, \ldots, 2^4\}$. Again, for SVM training we normalized the data to have zero mean and a standard deviation of one. The number of examples per class varies heavily and hence we chose the empirical prior for the class prior probabilities π_c. Using a uniform prior considerably degraded the classification error.

UCI

The *UCI Machine Learning Repository* [32] is a large collection of data sets for different machine learning tasks. We normalized each of these data sets to have zero mean and a standard deviation of one for all classifiers. Due to the relatively small UCI data sets, we performed 10-fold cross validation on each of them. We selected the following three data sets for our experiments:

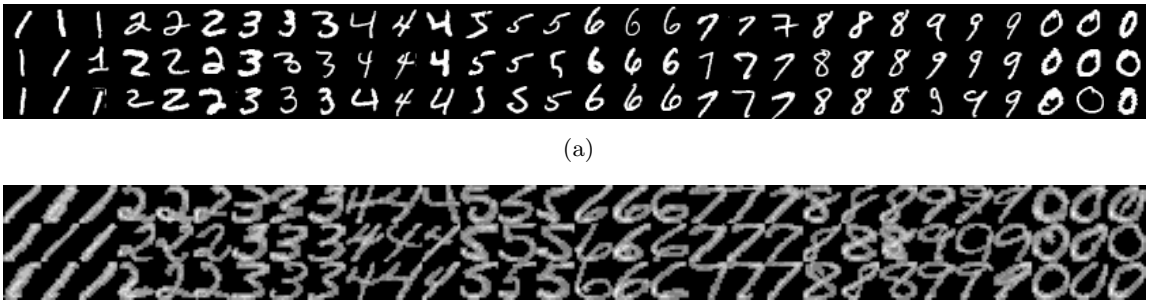
- The *Iris* data set contains three classes of different iris flowers. There are four features and 50 examples of each class. Hence, we chose a uniform class prior. We evaluated the set $K \in \{1, \ldots, 10\}$ for the number of components per class. Since this data set does not contain a separate test set, the recorded error values are computed as the mean of the validation errors over the 10 cross validation runs.
- The *Breast Cancer Wisconsin (Diagnostic)* data set is a binary classification task where the feature vectors are classified to either benign or malignant. The data consists of 30 features. There are 212 examples of the first class and 357 examples belonging to the second class. Thus, we computed the empirical prior for the class priors π_c. We evaluated the set $K \in \{1, \ldots, 8\}$ for the number of components per class. Again, the recorded classification errors are computed as the mean of the validations errors of the 10 cross validation runs.
- The *Image Segmentation* data set is used to classify a pixel of an image to one of 7 classes. The classes are brickface, sky, foliage, cement, window, path and grass. Each example is represented as a feature vector with 19 entries[12] which is computed over a 3×3 region of the image. There are 30 instances of each class as training examples and 300 instances of each class as test examples. Hence, we selected a uniform class prior. We evaluated the set $K \in \{1, \ldots, 6\}$ for the number of components per class. After obtaining the hyperparameters with 10-fold cross validation, the models were learned again with these parameters on the whole training set.

4.3.2 Optimization setup

The hyperparameters leading to the best results in terms of classification error were obtained by grid search. The hyperparameters were optimized on the following grids:

- GMM ML: K individual (see above)
- GMM Hybrid: K individual (see above), $\lambda \in \{2^0, \ldots, 2^{10}\}, \gamma \in \{0.1, 2^{-2}, \ldots, 2^6\}$
- GMM MCL: K individual (see above)
- GMM MM: K individual (see above), $\lambda \in \{10^{-3}, \ldots, 10^4\}$
- SVM Linear:[13] $\lambda \in \{2^{-10}, \ldots, 2^{10}\}$
- SVM RBF: $\lambda \in \{2^{-10}, \ldots, 2^{10}\}, \gamma \in \{2^{-10}, \ldots, 2^{10}\}$

[12] There are effectively 18 features since one feature is constant over all examples.
[13] The parameter λ corresponds to the parameter C in libSVM.



(a)

(b)

Figure 4.5: (a) depicts examples of the MNIST data set. (b) shows examples of the USPS data set.

The number of components K influences the number of model parameters and consequently the time needed for optimization. Furthermore, plausible values for K are related to the size of the available training data. If the number of training examples is high, one can typically also select larger values for K without severely overfitting the data. Hence, the set for values of K is chosen individually for each data set.

We used the following setup for the optimization procedures. On MNIST, MNIST50, USPS and TIMIT we perform a maximum number of 5000 iterations of the quasi-Newton L-BFGS algorithm. We applied the limited memory variant of the BFGS algorithm using the 10 latest update steps due to the high amount of variables to optimize.[14] After 1000 iterations, we check in each iteration if early stopping can be applied. Early stopping is performed if the best validation error found so far did not decrease in the latest 250 iterations or the generalization loss (2.6) exceeds 0.1. We select the parameters which resulted in the best validation performance over all iterations. At least 10 random restarts were performed when computing the ML GMMs. The same ML parameters were used as initial solution for each of the GMM classifiers.

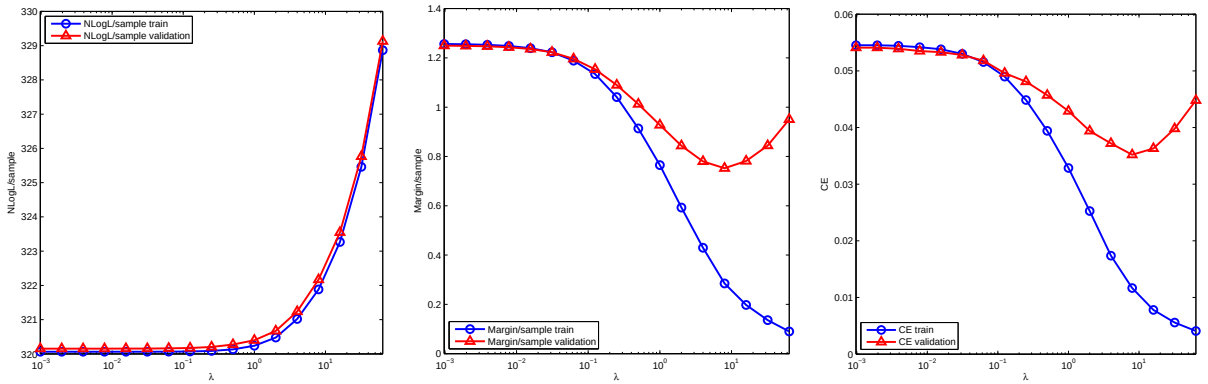
The numbers of training examples in the UCI data sets are much smaller than on the other data sets. Thus, we performed 10-fold cross validation to optimize the hyperparameters. We performed a maximum of 10000 iterations for each run and stopped prematurely if the largest entry of the gradient became sufficiently small. As mentioned in Section 2.1.3, early stopping is not applicable in conjunction with cross validation. For each run of the cross validation procedure, we computed ML parameters of the training partition with the EM algorithm using 10 random restarts. The ML solution then served as initial parameter configuration for the BFGS quasi-Newton algorithm. The number of parameters enabled us to stick to the full BFGS algorithm, rather than its memory limited variant L-BFGS. Except the Image Segmentation data set, where we are given a separate test set, the test error was computed by the mean of the validation errors over the 10 cross validation runs.

For the MCL optimized GMMs we computed 1000 iterations on every data set. For the MM optimized GMMs we computed 500 iterations. We did not use early stopping for these models and used the parameters after the full number of iterations. The initial parameters are the same ML solutions as in the hybrid GMMs and we also did not change the class priors π_c.

4.3.3 Results

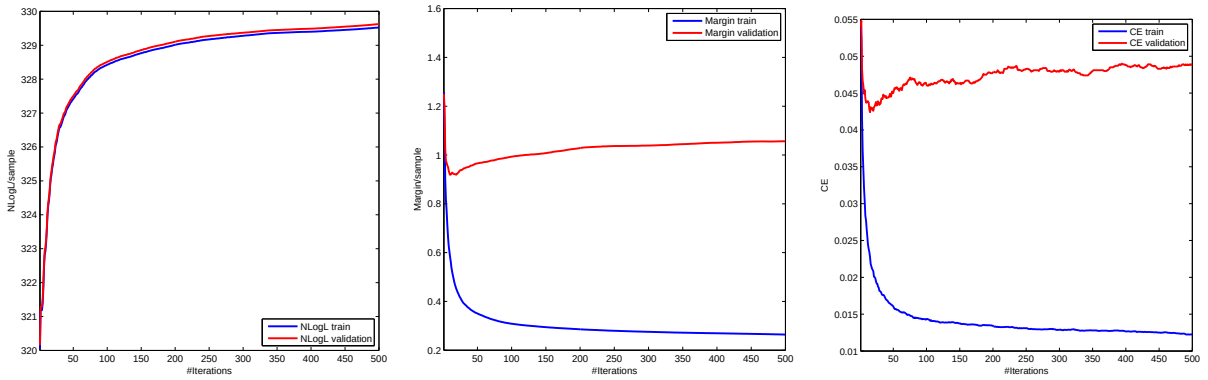
We first illustrate several characteristics of the hybrid GMM classifier on the MNIST50 data set. Figure 4.6 shows how the negative log-likelihood, the margin and the classification error change

[14] For the TIMIT data set, in the case of a single Gaussian per class, we would need to optimize more than 7000 parameters. Computing an approximation of the full inverse Hessian matrix becomes intractable even for small values of K.



(a) (b) (c)

Figure 4.6: Illustration of how the trade-off parameter λ influences (a) the negative log-likelihood (NLogL), (b) the margin and (c) the classification error (CE). The plots were computed on the MNIST50 data set with 16 components per class and desired margin $\gamma = 8$. Each model was optimized for 5000 iterations without early stopping



(a) (b) (c)

Figure 4.7: Illustration of how (a) the negative log-likelihood (NLogL), (b) the margin and (c) the classification error (CE) change over the number of iterations. The plots were computed on the MNIST50 data set with 16 components per class, generative-discriminative trade-off $\lambda = 64$ and desired margin $\gamma = 8$.

for different values of the trade-off parameter λ and a fixed desired margin γ. The probabilistic interpretation of a generative model can be quantified by its likelihood per sample. Figure 4.6(a) shows how the negative log-likelihood per example increases as λ increases. The negative log-likelihood of models for small values of λ hardly differs from that of the ML solution. At about $\lambda = 1$, we can see that the negative log-likelihood starts to increase drastically. Figure 4.6(b) shows how the margin decreases as λ becomes larger. The margin decreases consistently on the training set as λ increases. However, for the validation set the margin only decreases for smaller values of λ and increases again for larger values of λ, which is typically a sign of overfitting. It is therefore usually some intermediate value of λ, that results in the best generalization performance. Note the similarity of the margin per example in Figure 4.6(b) to the classification error in Figure 4.6(c).

Next we show how the same characteristics of the model change over the number of iterations of a single quasi-Newton optimization run. Recall that we start from the ML solution. Figure 4.7 shows the behavior on the MNIST50 data set for a fixed λ and a fixed γ. Figure 4.7(a) shows how the negative log-likelihood increases over the number of iterations. Figure 4.7(b) and Figure 4.7(c) illustrate that the margin and the classification error on the training set drop

consistently. However, the margin and the classification error on the validation set drop only at the beginning and start to increase again after a few iterations. This is a clear sign of overfitting which can be solved with the early stopping heuristic. It is remarkable that the values in these plots change severely at the beginning and then change only slightly in the remainder. While this can be a typical behavior in many applications, we cannot rule out that the classification error on the validation set will drop in later iterations. In our experiments we often observed that the validation error decreased after more than 1000 iterations.

Table 4.1 shows the results for different classifiers on the data sets described in Section 4.3.1. The hyperparameters which resulted in the best performance are shown in Table B.9. The performance of the hybrid GMM classifier increases the performance of ML GMMs substantially. Furthermore, we outperform MCL optimized GMMs and achieve a comparable performance to the MM optimized models from Pernkopf. We had difficulties with the implementation of MM optimized GMMs and were not able to compute results for the smaller data sets Iris, Breast Cancer and Image Segmentation. We believe that the results for MM optimized GMMs on MNIST are pessimistic and better results would be achieved if more iterations had been used.

SVMs with linear kernel and the hybrid GMM classifier perform evenly over all data sets, while SVMs with RBF kernel outperform the hybrid GMM classifier consistently. However, the number of parameters of SVMs are consistently larger than the number of parameters of GMMs as shown in Table 4.2. Furthermore, training and classification of SVMs on data sets with many classes, such as the TIMIT data set, takes a long time. We observed differences in classification time by a factor of 1000 for SVMs with RBF kernel and the hybrid GMM classifier on the TIMIT data set. libSVM builds the multiclass classifier according to the one-vs-one strategy and thus needs to solve a large amount of independent binary classification tasks.

dataset	GMM ML	GMM Hyb.	GMM CL	GMM MM	SVM Lin.	SVM RBF
MNIST	13.94	6.56	7.28	7.88	5.38	1.61
MNIST50	4.32	3.20	3.45	2.88	6.37	1.62
USPS	7.77	4.87	6.00	5.13	4.63	1.97
TIMIT	30.02	22.19	24.06	23.63	26.06	20.30
Iris	2.67	2.00	4.00	/	2.00	2.00
Breast	4.39	2.05	3.00	/	2.10	1.75
Segment.	13.00	10.14	9.43	/	7.90	8.14

Table 4.1: Classification errors (%) of different classifiers on the data sets described in Section 4.3.1.

dataset	GMM ML	GMM Hyb.	GMM CL	GMM MM	SVM Lin.	SVM RBF
MNIST	459210	114810	114810	114810	7292403	7978709
MNIST50	129290	129290	64650	129290	493561	892689
USPS	82090	10270	20530	20530	583741	797383
TIMIT	115479	115479	14469	115479	7036396	7756620

Table 4.2: Numer of parameters used for the best models. The parameters of the GMM classifiers are the class prior probabilities π_c, the component priors $\alpha_{c,k}$, the component means $\boldsymbol{\mu}_{c,k}$ and the diagonal component covariance matrices $\boldsymbol{\Sigma}_{c,k}$. The parameters of SVMs are the number of support vectors multiplied by the number of features, the non-zero weights α of the support vectors and the offsets ρ (terminology according to libSVM).

For the TIMIT data set we also replaced the soft-hinge function (3.17) by a smoothed version of the ramp function (3.15). Due to the increased computation effort caused by the additional hyperparameter ν, we evaluated only the reduced grid for the component priors $K \in \{1, 2\}$, the generative-discriminative trade-off $\lambda \in \{2^0, \ldots, 2^{10}\}$ and the desired margin $\gamma \in \{0.1, 2^{-2}, \ldots, 2^6\}$. For each of these combinations we used the maximum loss values

$\nu \in \{\gamma + 2^1, \ldots, \gamma + 2^4\}$. The classification performance of the ramp function compared to the hinge function increased in 82 out of 220 parameter configurations for at least one choice of the parameter ν. Using the ramp function, the classification error decreased by up to 0.4%. We observed an increase in likelihood in 60 out of the 82 cases where the classification performance improved. The performance increased only for hyperparameter configurations with $\lambda \geq 2^2$ and $\gamma \leq 2^3$. This shows that the ramp function can improve the classification performance and the probabilistic interpretation of the model. We were not able to improve the best models learned with the hinge function on the TIMIT data set since these models have a high desired margin γ.

4.4 Experiments with Missing Data

This section presents the results of the hybrid GMM classifier in the presence of missing data. We performed experiments with missing features at classification time and with missing labels at training time, i.e. semi-supervised learning.

4.4.1 Results with missing features at classification time

We conducted our experiments for classification with missing features on the MNIST, MNIST50, USPS and TIMIT data sets. The training set and the validation set contained no missing features. We selected the number of components for the GMM classifiers to be the number for which the ML solution achieved the smallest validation error. The remaining parameters of the GMM classifiers and SVMs were obtained by optimizing the performance on the separate held-out validation set.

We chose the number for all GMM classifiers to be equal in order to obtain fair results. For instance, the best hybrid GMM on the USPS data set was obtained with only two components per class whereas the pure generative solution performed best with 16 components. Without missing features hybrid GMMs clearly outperformed the ML solution, but on the other side, the hybrid model with only two components performed poorly in the presence of missing features.

For the GMM classifiers we marginalized out the missing features as described in Section 3.3.1. For SVMs we performed unconditional mean imputation and 3-nearest neighbor imputation. We conducted experiments for different numbers of missing features. Let D be the number of features and $p \in [0, 1]$ be the percentage of missing features for an experiment. For our experiments we removed randomly $round(D \cdot p)$ features of each example of the test set.

Figure 4.8 shows the classification errors of the different classifiers. The detailed classification errors of this experiments are shown in Appendix B. We observe a similar behavior as Peharz et al. [2]: The ML GMM has a relatively high error when no features are missing but its performance stays approximately constant up to 50% missing features. Except for the TIMIT data set, also the performance of hybrid GMMs stays almost constant up to 50% missing features. The hybrid model outperforms the ML solution up to 30% missing features on all data sets. For more than 50% missing features the hybrid model achieves similar error rates as the generative solution on MNIST50 and USPS and it outperforms the generative solution on MNIST. The test error of hybrid GMMs on TIMIT increases faster for smaller amounts of missing features and thus the performance for 50% missing features or more is poor.

Over all data sets hybrid GMMs performed even compared to MCL and MM optimized GMMs in the presence of missing data. We observed that the best models, obtained by minimizing the validation error, typically have a relatively high generative-discriminative trade-off parameter λ or a high desired margin γ. As we have shown earlier, large values for any of these two parameters

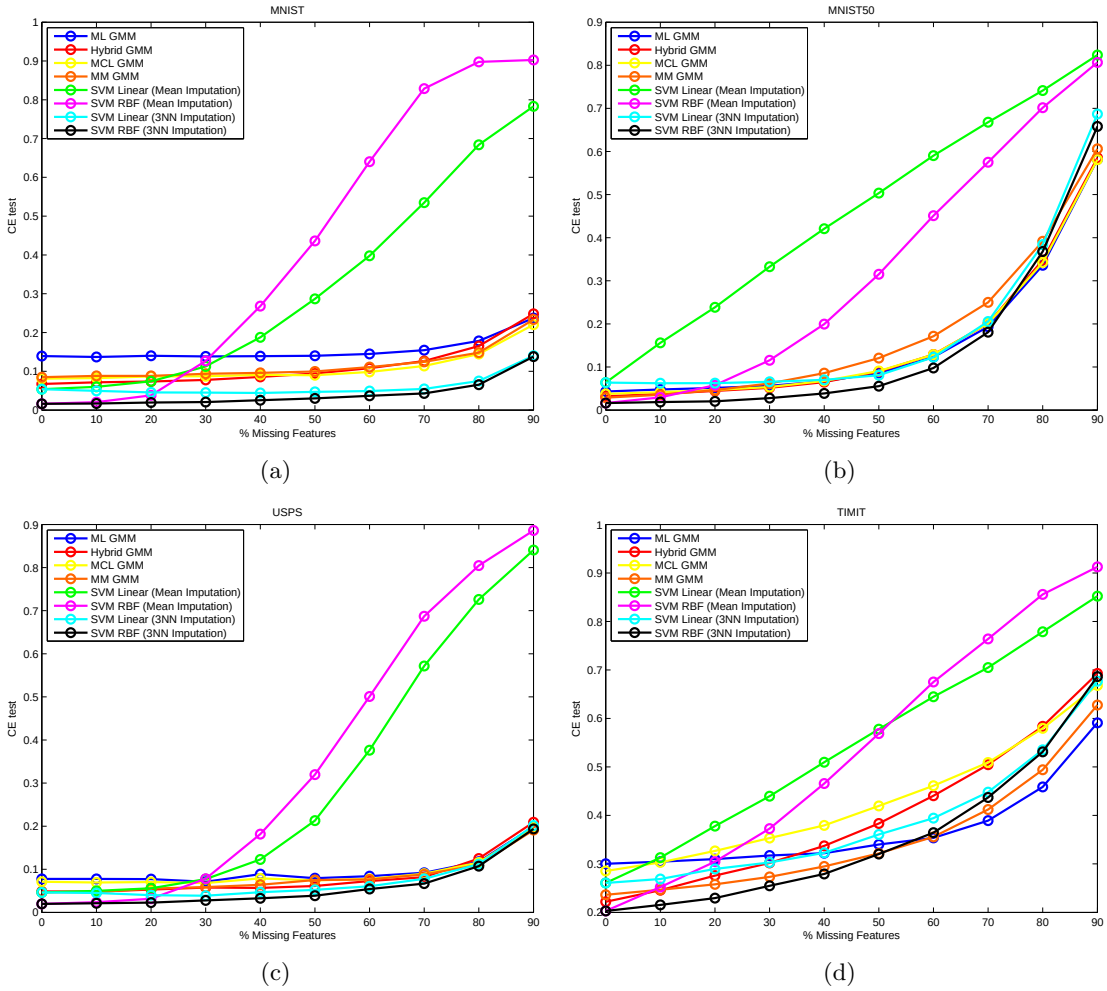
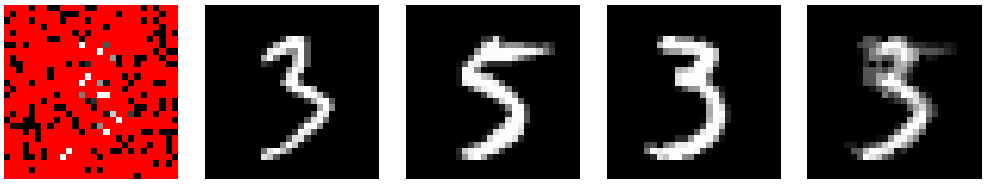


Figure 4.8: Illustration of the performance of several classifiers for different amounts of missing features on (a) MNIST, (b) MNIST50, (c) USPS and (d) TIMIT.

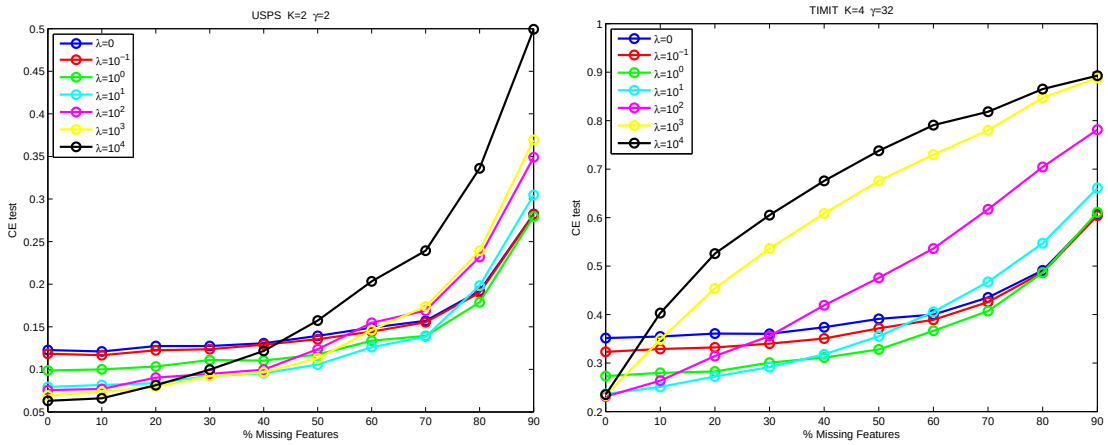
harms the probabilistic interpretation of the model and the classification performance in the presence of missing features. However, the large values for these parameters are not surprising, since the optimization criterion was to achieve a high validation performance and the validation set does not contain any missing features. We therefore propose to choose an optimization criterion that resembles the task one intends to solve, such as optimizing the performance on a validation set that itself contains missing features.

Furthermore, we see that the unconditional mean imputation strategy achieves only poor results, while 3-nearest neighbor imputation achieves a high performance for SVMs with both linear and RBF kernel over the whole range of missing features. We believe this behavior to stem from the way we remove the features from the test set, i.e. uniform at random which conforms the MCAR assumption. Imputation techniques are typically only valid if the MCAR assumption is given. Hence, the high performance can be attributed to the imputation technique rather than the SVM classifier. Figure 4.9 illustrates 3-nearest neighbor imputation on the MNIST data set. The image in Figure 4.9(a) shows a test example depicting the digit three with 80% missing features. The imputed image in Figure 4.9(e) recovers the original digit accurately. The imputed image is a blend of three training images out of which two are of the correct class. If there are many missing features this is almost the same as classifying a training image. Since the training set was used to train the models and the error on the training set is typically small, the performance is high.



(a) (b) (c) (d) (e)

Figure 4.9: Illustration of 3-nearest neighbor imputation on the MNIST data set. (a) shows a test example depicting the digit three with 80% missing features. The missing features are shown in red. (b), (c) and (d) show the nearest, second nearest and third nearest neighbors of (a) from the training set. The imputed image is shown in (e).



(a) (b)

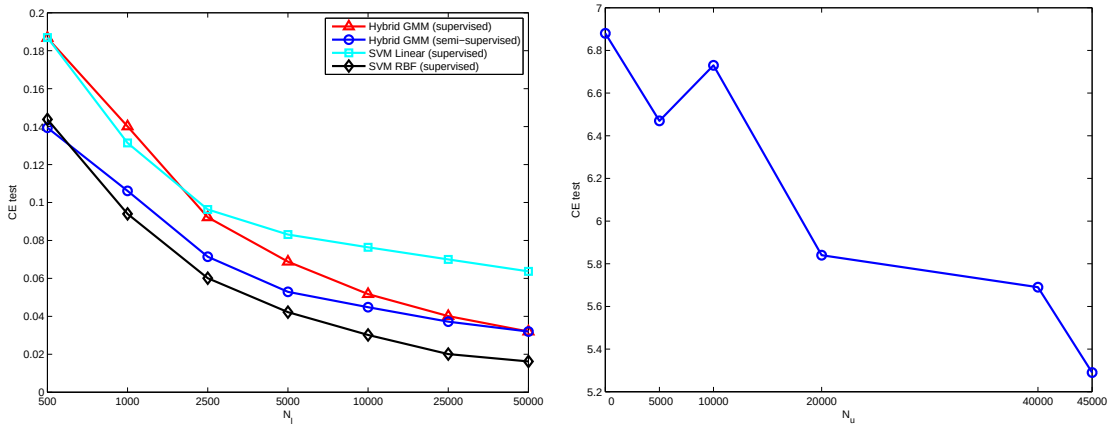
Figure 4.10: Illustration of the influence of lambda on the classification error (CE) in the presence of missing features. (a) shows results for $K = 2$ and $\gamma = 2$ for different values of λ on the USPS data set. (b) shows results for $K = 4$ and $\gamma = 32$ for different values of λ on the TIMIT data set.

Figure 4.10 illustrates the influence of lambda on the performance on data with missing features. The detailed classification errors are shown in Appendix B. The classification error of the generative solution ($\lambda = 0$) is high for fewer missing features but it becomes smaller compared to higher values of λ for more missing features. Nevertheless, this does not imply that the generative solution is always the best for many missing features. For instance, the relatively small choice of $\lambda = 1$ outperforms the ML solution over the whole range of missing features but its performance in case of less missing features is worse compared to larger values of λ.

4.4.2 Results for semi-supervised learning

We evaluated the hybrid GMM classifier on the MNIST50 data set with partly missing labels. We optimized the hyperparameters on a validation set with grid search for the values $K \in \{2^0, \ldots, 2^7\}$, $\lambda = \{2^0, \ldots, 2^{10}\}$ and $\gamma \in \{0.1, 2^{-2}, \ldots, 2^6\}$. We additionally evaluated the performance of SVMs with linear and RBF kernel on the labeled data only. Unfortunately, we were not able to find available software packages that could handle this semi-supervised problem for comparison.

The results are shown in Figure 4.11. Figure 4.11(a) shows that the model is capable of improving the classification performance if additional unlabeled data is given. Especially if only a few labeled examples are given, the performance improves significantly. In case of 500 labeled examples and 49500 unlabeled examples, the classification error drops by more than 4.5%



(a) (b)

Figure 4.11: Illustration of the classification error (CE) on the MNIST50 data set with semi-supervised learning. In (a) we vary the ratio of labeled and unlabeled data for a total amount of 50000 data examples, i.e. $N_l + N_u = 50000$. *The plot shows values for* $N_l \in \{500, 1000, 2500, 5000, 10000, 25000, 50000\}$. *(b) shows the classification error for a fixed number of labeled examples* $N_l = 5000$ *and a varying number of unlabeled examples* $N_u \in \{0, 5000, 10000, 20000, 40000, 45000\}$.

compared to hybrid the model, which was trained using the labeled data alone. In this case, we also outperform the SVM with RBF kernel that was trained with the labeled data only. The detailed classification errors of this experiment are shown in Table B.7 and the hyperparameters which result in the best performance are shown in Table B.10.

A similar experiment is depicted in Figure 4.11(b). Here the number of labeled data is fixed to $N_l = 5000$ while the amount of unlabeled data varies. We can see that the classifier tends to improve as the amount of unlabeled data grows. However, we also observe an increase in classification error for $N_u = 10000$ compared to $N_u = 5000$. This is an example that additional unlabeled data does not always improve the performance of a classifier. The detailed classification errors of this experiment are shown in Table B.8 in the appendix.

5 Conclusion

5.1 Summary

While classical ML and MAP optimized GMMs are usually not competitive in real world classification tasks, some discriminative strategies have been developed in the past in order to achieve a higher classification performance with GMMs [10, 15]. However, none of these approaches takes the likelihood into account and the probabilistic interpretation of the resulting classifier often only stems from the ML model serving as initial solution for subsequent optimization procedures. In this thesis we have shown how to learn GMMs in a hybrid generative-discriminative way in order to close this gap. We accomplished this task by formulating an objective that trades off between a likelihood term and a large margin term.

We demonstrated the capabilities of hybrid GMMs on synthetic data. Furthermore, we compared our model on real world data with the following classifiers: ML optimized GMMs, MCL optimized GMMs, MM optimized GMMs and SVMs with linear and RBF kernel. In the case of no missing data our model is competitive with MM optimized GMMs and linear SVMs and outperforms the other GMM classifiers. SVMs with RBF kernel achieved the best results. However, we have shown that SVMs need substantially more parameters than GMMs.

The goal of this thesis was to construct a classifier that is capable of dealing with missing data. We evaluated the performance of hybrid GMMs when classifying in the presence of missing features. In case of at most 30% missing features, hybrid GMMs outperformed generative GMMs. The classification error of the ML solution stays approximately constant for up to 50% and achieves a good performance for more than 50% missing features. Our model performed even compared to MCL and MM optimized GMMs. Hybrid GMMs performed poorly on the TIMIT data set in the presence of missing data. This happens because the hyperparameters are optimized on a validation set without missing features and thus the model tends to get more of a discriminative character. We have shown that tuning the hyperparameters carefully can result in a performance increase compared to the ML solution for both few and many missing features. We applied unconditional mean imputation and 3-nearest neighbor imputation for linear SVMs and SVMs with RBF kernel. While mean imputation provided the worst results, 3-nearest neighbor imputation performed best on all data sets. We believe the superior performance of the 3-nearest neighbor imputation heuristic to stem from the way we removed the features in our experiments. The features were removed according to the MCAR assumption.

We also conducted experiments with missing labels on the MNIST50 data set. We were able to increase the performance of the classifier by using additional unlabeled data. We compared the hybrid model to SVMs with linear and RBF kernel that only used the labeled data. In case of only a few labeled data examples, where a pure supervised classifier performs poorly, we were able to increase the performance substantially. In particular, on the MNIST50 data set with only 500 labeled data examples and 49500 unlabeled data examples we achieved a better performance than a SVM with RBF kernel which was learned on the labeled data alone. We have shown that more unlabeled data typically results in a higher performance gain. However, we also observed that additional unlabeled data can degrade the performance.

5.2 Future Work

Hybrid generative-discriminative GMMs are competitive with other state of the art GMM classifiers if there are no missing features. The likelihood term of the hybrid objective is used to regularize the model parameters. Nevertheless, when optimizing the classification performance on a separate validation set without missing features, the best results are typically obtained with hyperparameters that favor the discriminative character of the model. This results in a poor performance in case the number of missing features becomes large. If the amount of missing features is high, we believe that optimizing hyperparameters using a validation set that also contains missing features, results in a higher performance. Furthermore, future work should evaluate the performance of hybrid GMMs if the stronger MAR or NMAR assumptions hold. Although we provided the theoretical foundations of how to learn GMM in the presence of missing features, we leave experiments of this scenario for future studies.

The hybrid generative-discriminative objective is a non-convex and possibly highly multimodal function. As typical for such functions, the problem of getting stuck in bad local optima could prevent the model from achieving a high performance. More than 10 random restarts were performed when computing the initial ML solution with the EM algorithm. However, we did not investigate how different initial solutions when optimizing the hybrid objective might change the quality of the classifier. Future work should address this issue.

In our experiments we optimized three different hyperparameters, namely the number of components per class K, the generative-discriminative trade-off parameter λ and the desired margin γ. Therefore, we imposed the restriction that the number of components of all classes are equal. However, the number of components can typically be chosen larger the more data there is available. If the numbers of examples per class differ substantially, this restriction could become a problem. Future work should address heuristics for choosing different component numbers per class. Furthermore, since the number of hyperparameters without this restriction would be exponential in the number of classes, techniques to search the hyperparameter space more systematically than grid search could be promising.

Another restriction we have made is the restriction to diagonal covariance matrices. Although GMMs can model dependencies among the features using the superposition of the components, some applications might be more naturally solved by using full covariance matrices instead of diagonal covariances.

A Derivation of the Gradient

A.1 Gradient of the Hybrid Discriminative-Generative Objective

In this section we derive the gradient of the hybrid generative-discriminative objective for GMMs. Recall that the smoothed hybrid generative-discriminative objective is given by

$$\tilde{l}_{hybrid}(\boldsymbol{\theta}) = -\sum_{n=1}^{N} \left[\log p\left(\boldsymbol{x}_n|\boldsymbol{\theta}_{c_n}\right) + \log \pi_{c_n}\right] + \lambda \sum_{n=1}^{N} \mathrm{h}_\epsilon \left(\underset{c \neq c_n}{\mathrm{smax}}(\gamma - \beta_{n,c}(\boldsymbol{\theta}))\right). \tag{A.1}$$

To increase readability, we will restate the meaning of several expressions here. $\beta_{n,c}$ denotes the log-probabilistic margin of the n-th example with respect to class c which is defined as

$$\beta_{n,c}(\boldsymbol{\theta}) = \log p\left(\boldsymbol{x}_n, c_n|\boldsymbol{\theta}\right) - \log p\left(\boldsymbol{x}_n, c|\boldsymbol{\theta}\right). \tag{A.2}$$

The soft-hinge function h_ϵ is defined as

$$\mathrm{h}_\epsilon(t) = \begin{cases} 0 & t < -\epsilon \\ t & t > \epsilon \\ \frac{(t+\epsilon)^2}{4\epsilon} & \text{otherwise} \end{cases} \tag{A.3}$$

and its derivative is given by

$$\mathrm{h}'_\epsilon(t) = \begin{cases} 0 & t < -\epsilon \\ 1 & t > \epsilon \\ \frac{(t+\epsilon)}{2\epsilon} & \text{otherwise} \end{cases}. \tag{A.4}$$

The soft-max function smax is defined as

$$\underset{t_1,\ldots,t_L}{\mathrm{smax}} = \frac{1}{\eta} \log \sum_{i=1}^{L} \exp\left(\eta t_i\right) \tag{A.5}$$

and its partial derivative with respect to t_j is given by

$$\frac{\partial \operatorname{smax}_{t_1,\dots,t_L}}{\partial t_j} = \frac{\exp(\eta t_j)}{\sum_{i=1}^{L} \exp(\eta t_i)}. \tag{A.6}$$

The following rules of vector and matrix differentiation will aid in computing the gradient:

$$\frac{\partial \boldsymbol{a}^T \boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{a} \tag{A.7}$$

$$\frac{\partial \boldsymbol{x}^T \boldsymbol{S} \boldsymbol{x}}{\partial \boldsymbol{x}} = 2\boldsymbol{S}\boldsymbol{x} \tag{A.8}$$

$$\frac{\partial \det \boldsymbol{S}}{\partial \boldsymbol{S}} = \boldsymbol{S}^{-1} \det \boldsymbol{S} \tag{A.9}$$

$$\frac{\partial \boldsymbol{a}^T \boldsymbol{S}^{-1} \boldsymbol{a}}{\partial \boldsymbol{S}} = -\boldsymbol{S}^{-1} \boldsymbol{a}\boldsymbol{a}^T \boldsymbol{S}^{-1}, \tag{A.10}$$

where $\boldsymbol{x}$ and $\boldsymbol{a}$ are vectors in $\mathbb{R}^D$ and $\boldsymbol{S}$ is a symmetric square matrix in $\mathbb{R}^{D\times D}$. Equations (A.7) and (A.8) can be verified by element-wise differentiation. Equations (A.9) and (A.10) are provided in [33]. We start with the gradient of the density of the Gaussian distribution with respect to their mean and their symmetric positive semidefinite covariance matrix. Recall that the Gaussian is defined as

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D \det \boldsymbol{\Sigma}}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right). \tag{A.11}$$

The gradient with respect to $\boldsymbol{\mu}$ is given by

$$\frac{\partial \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})}{\partial \boldsymbol{\mu}} = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \frac{\partial}{\partial \boldsymbol{\mu}} \left(-\frac{1}{2}\boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} - \frac{1}{2}\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right) \tag{A.12}$$

$$= \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})\, \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}). \tag{A.13}$$

In order to compute the gradient with respect to the covariance matrix $\boldsymbol{\Sigma}$, we start with two auxiliary calculations. Using (A.9) the gradient of the normalizing factor is given by

$$\frac{\partial}{\partial \boldsymbol{\Sigma}} \frac{(\det \boldsymbol{\Sigma})^{-1/2}}{\sqrt{(2\pi)^D}} = \frac{-\frac{1}{2}(\det \boldsymbol{\Sigma})^{-3/2} \boldsymbol{\Sigma}^{-1} \det \boldsymbol{\Sigma}}{\sqrt{(2\pi)^D}} = -\frac{1}{2} \frac{1}{\sqrt{(2\pi)^D \det \boldsymbol{\Sigma}}} \boldsymbol{\Sigma}^{-1}. \tag{A.14}$$

Using (A.10), the gradient of the term inside the exponential is given by

$$\frac{\partial}{\partial \boldsymbol{\Sigma}} \left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) = \frac{1}{2}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}. \tag{A.15}$$

With these two results and the product rule, the gradient with respect to the covariance matrix is given by

$$\frac{\partial \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})}{\partial \boldsymbol{\Sigma}} = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \frac{1}{2} \left(\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} - \boldsymbol{\Sigma}^{-1}\right). \tag{A.16}$$

Next we will calculate the gradient of the log-likelihood function for GMMs. Recall that the density of a GMM with K components, defined by the parameters $\boldsymbol{\theta} = (\alpha_1, \dots, \alpha_K, \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K)$

where $\boldsymbol{\theta}_k = (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, is given by

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} \alpha_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{A.17}$$

Using the results (A.13) and (A.16), the gradient of the log-likelihood for GMMs with respect to the individual parameters are given by

$$\frac{\partial \log p(\boldsymbol{x}|\boldsymbol{\theta})}{\partial \alpha_k} = \frac{\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k'=1}^{K} \alpha_{k'} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})} = \frac{\alpha_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k'=1}^{K} \alpha_{k'} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})} \frac{1}{\alpha_k} \tag{A.18}$$

$$\frac{\partial \log p(\boldsymbol{x}|\boldsymbol{\theta})}{\partial \boldsymbol{\mu}_k} = \frac{\alpha_k}{\sum_{k'=1}^{K} \alpha_{k'} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})} \frac{\partial \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\partial \boldsymbol{\mu}_k} \tag{A.19}$$

$$= \frac{\alpha_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k'=1}^{K} \alpha_{k'} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})} \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_k) \tag{A.20}$$

$$\frac{\partial \log p(\boldsymbol{x}|\boldsymbol{\theta})}{\partial \boldsymbol{\Sigma}_k} = \frac{\alpha_k}{\sum_{k'=1}^{K} \alpha_{k'} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})} \frac{\partial \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\partial \boldsymbol{\Sigma}_k} \tag{A.21}$$

$$= \frac{\alpha_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k'=1}^{K} \alpha_{k'} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})} \frac{1}{2} \left(\boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_k) (\boldsymbol{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} - \boldsymbol{\Sigma}_k^{-1} \right). \tag{A.22}$$

Notice the common term at the beginning of each of these expression. We refer to this term as the component posterior probability $p(k|\boldsymbol{x}, \boldsymbol{\theta})$ which expresses the probability that component k is responsibly for generating data example $\boldsymbol{x}$. We can now turn to calculating the gradient of the hybrid objective with respect to parameters of the class conditional densities $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_C$. Note that both parts, the log-likelihood and the large margin term, consist of a sum over all examples. Since differentiation is linear, the gradient can be computed for each example separately and the overall result is obtained as the sum of the individual gradients. Thus, to keep the notation uncluttered, we only consider the gradient with respect to a single example $(\boldsymbol{x}_n, c_n)$ rather than the whole data set $\{(\boldsymbol{x}_1, c_1), \ldots, (\boldsymbol{x}_N, c_N)\}$. We start with the gradient of the log-likelihood term. Note that the class priors π_c are independent of the class model parameters $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_C$ and can therefore be neglected. The log-likelihood function decomposes into a sum of terms, each depending only on the parameters of a single class. Each example therefore only influences the gradient of the parameters of the corresponding class. Hence, to compute the gradient with respect to a particular class, it suffices to compute the gradient using (A.18), (A.20) and (A.22) for all examples of that class and sum up the results.

Next we compute the gradient of the large margin term. Here the soft-max function ties all the classes together and it is no longer possible to calculate the gradient for each class separately.

$$\frac{\partial}{\partial \boldsymbol{\theta}_c} \mathrm{h}_\epsilon \left(\operatorname*{smax}_{c' \neq c_n} (\gamma - \beta_{n,c'}) \right) = \mathrm{h}'_\epsilon \left(\operatorname*{smax}_{c' \neq c_n} (\gamma - \beta_{n,c'}) \right) \frac{\partial}{\partial \boldsymbol{\theta}_c} \operatorname*{smax}_{c' \neq c_n} (\gamma - \beta_{n,c'}) \tag{A.23}$$

$$= \mathrm{h}'_\epsilon \left(\operatorname*{smax}_{c' \neq c_n} (\gamma - \beta_{n,c'}) \right) \sum_{c' \neq c_n} \frac{\partial \operatorname{smax}_{c'' \neq c_n}}{\partial (\gamma - \beta_{n,c'})} \cdot \frac{\partial (\gamma - \beta_{n,c'})}{\partial \boldsymbol{\theta}_c} \tag{A.24}$$

$$= \mathrm{h}'_\epsilon \left(\operatorname*{smax}_{c' \neq c_n} (\gamma - \beta_{n,c'}) \right) \sum_{c' \neq c_n} \frac{\exp(-\eta \beta_{n,c'})}{\sum_{c'' \neq c_n} \exp(-\eta \beta_{n,c''})} \cdot \frac{\partial (\gamma - \beta_{n,c'})}{\partial \boldsymbol{\theta}_c} \tag{A.25}$$

The gradient of the last term is given by

$$\frac{\partial(\gamma - \beta_{n,c'})}{\partial \boldsymbol{\theta}_c} = \frac{\partial}{\partial \boldsymbol{\theta}_c} \left(\gamma - \log p\left(\boldsymbol{x}_n|\boldsymbol{\theta}_{c_n}\right) - \log p\left(c_n\right) + \log p\left(\boldsymbol{x}_n|\boldsymbol{\theta}_{c'}\right) + \log p\left(c'\right)\right) \tag{A.26}$$

$$= \begin{cases} -\frac{\partial}{\partial \boldsymbol{\theta}_c} \log p\left(\boldsymbol{x}_n|\boldsymbol{\theta}_{c_n}\right) & c = c_n \\ \frac{\partial}{\partial \boldsymbol{\theta}_c} \log p\left(\boldsymbol{x}_n|\boldsymbol{\theta}_{c'}\right) & c = c' \\ 0 & \text{otherwise} \end{cases}. \tag{A.27}$$

If we take the derivative with respect to the parameters of the true class, i.e. $c = c_n$, the partial derivative (A.27) is independent of the summation index c' and we can pull it out of the sum in (A.25). The sum itself is one, since summing over all the partial derivatives of the soft-max function always equals one. If we take the derivative with respect to some other class $c \neq c_n$, the terms for all classes except $c' = c$ vanish. The gradient of the large margin term for example $(\boldsymbol{x}_n, c_n)$ with respect to the parameters θ_c is therefore given by

$$\mathrm{h}'_\epsilon\left(\max_{c' \neq c_n} \left(\gamma - \beta_{n,c'}\right)\right) \cdot \begin{cases} \frac{\exp(-\eta\beta_{n,c})}{\sum_{c' \neq c_n} \exp\left(-\eta\beta_{n,c'}\right)} \frac{\partial}{\partial \boldsymbol{\theta}_c} \log p\left(\boldsymbol{x}_n|\boldsymbol{\theta}_c\right) & c \neq c_n \\ -\frac{\partial}{\partial \boldsymbol{\theta}_c} \log p\left(\boldsymbol{x}_n|\boldsymbol{\theta}_{c_n}\right) & c = c_n \end{cases}. \tag{A.28}$$

Detailed interpretations of these results are given in Section 3.2.3.

A.2 Gradient of the Marginal Distribution

In this section we show how to calculate the gradient of the marginal distribution $p\left(\boldsymbol{x}|\boldsymbol{\theta}\right)$ which is needed to optimize the semi-supervised hybrid objective. Using the sum-rule of probability the marginal distribution of the data example $\boldsymbol{x}$ is given by

$$p\left(\boldsymbol{x}|\boldsymbol{\theta}\right) = \sum_{c=1}^{C} p\left(\boldsymbol{x}|\boldsymbol{\theta}_c\right) \pi_c. \tag{A.29}$$

The gradient of the marginal log-likelihood with respect to the parameter $\boldsymbol{\theta}_c$ of class c are then given by

$$\frac{\partial \log p\left(\boldsymbol{x}|\boldsymbol{\theta}\right)}{\partial \boldsymbol{\theta}_c} = \frac{\pi_c}{p\left(\boldsymbol{x}|\boldsymbol{\theta}\right)} \frac{\partial p\left(\boldsymbol{x}|\boldsymbol{\theta}_c\right)}{\partial \boldsymbol{\theta}_c}. \tag{A.30}$$

By multiplying the numerator and the denominator by $p\left(\boldsymbol{x}|\boldsymbol{\theta}_c\right)$ we obtain

$$\frac{\partial \log p\left(\boldsymbol{x}|\boldsymbol{\theta}\right)}{\partial \boldsymbol{\theta}_c} = p\left(c|\boldsymbol{x}, \boldsymbol{\theta}\right) \frac{1}{p\left(\boldsymbol{x}|\boldsymbol{\theta}_c\right)} \frac{\partial p\left(\boldsymbol{x}|\boldsymbol{\theta}_c\right)}{\partial \boldsymbol{\theta}_c}. \tag{A.31}$$

We recognize this expression as the gradient of the log-likelihood function of the data example as if it is labeled with class c weighted with the class posterior probability of class c. Hence, we can rewrite this expression as

$$\frac{\partial \log p\left(\boldsymbol{x}|\boldsymbol{\theta}\right)}{\partial \boldsymbol{\theta}_c} = p\left(c|\boldsymbol{x}, \boldsymbol{\theta}\right) \frac{\partial \log p\left(\boldsymbol{x}|\boldsymbol{\theta}_c\right)}{\partial \boldsymbol{\theta}_c}. \tag{A.32}$$

The gradient of the marginal log-likelihood is further discussed in Section 3.3.2.

B Supplementary Tables

The following tables contain the classification errors of several figures in Section 4.4 and the best hyperparameters of different experiments. Tables B.1, B.2, B.3 and B.4 contain the data shown in Figure 4.8. Mean imputation and 3-nearest neighbor imputation are indicated by (M) and (3NN) respectively. Tables B.5 and B.6 contain the data shown in Figure 4.10. Tables B.7 and B.8 contain the data shown in Figure 4.11. Table B.9 and Table B.10 contain the best hyperparameters corresponding to Table 4.1 and Figure 4.11(a) respectively.

classifier	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
GMM ML	13.94	13.69	14.00	13.84	13.90	13.99	14.48	15.44	17.83	23.73
GMM Hybrid	6.73	7.17	7.38	7.77	8.56	9.44	10.80	12.65	16.50	24.82
GMM MCL	8.10	8.25	8.68	8.72	9.08	8.98	9.84	11.38	14.46	22.02
GMM MM	8.49	8.81	8.81	9.35	9.60	9.95	11.08	12.43	14.80	23.36
SVM Lin. (M)	5.38	6.01	7.41	11.26	18.76	28.68	39.77	53.47	68.39	78.31
SVM Lin. (3NN)	5.38	4.95	4.56	4.51	4.42	4.69	4.90	5.46	7.48	14.03
SVM RBF (M)	1.61	2.05	3.79	12.67	26.80	43.60	64.04	82.87	89.76	90.26
SVM RBF (3NN)	1.61	1.69	1.95	2.07	2.55	3.02	3.68	4.29	6.54	13.78

Table B.1: Classification errors (%) of several classifiers for different numbers of missing features on MNIST (see Figure 4.8(a)).

classifier	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
GMM ML	4.32	4.79	5.15	5.63	6.68	8.62	12.31	19.33	33.60	58.17
GMM Hybrid	3.20	3.75	4.44	5.16	6.49	8.94	12.89	20.06	34.80	58.57
GMM MCL	3.75	4.16	4.66	5.36	6.77	9.06	12.87	20.07	34.24	58.09
GMM MM	2.88	3.55	4.64	6.19	8.58	12.06	17.13	24.99	39.16	60.61
SVM Lin. (M)	6.37	15.60	23.84	33.26	42.08	50.35	59.03	66.79	74.14	82.39
SVM Lin. (3NN)	6.37	6.24	6.26	6.58	7.02	8.09	12.25	20.54	38.40	68.68
SVM RBF (M)	1.62	2.94	5.78	11.56	19.94	31.52	45.10	57.51	70.15	80.65
SVM RBF (3NN)	1.62	1.86	2.03	2.78	3.87	5.54	9.76	18.07	36.80	65.81

Table B.2: Classification errors (%) of several classifiers for different numbers of missing features on MNIST50 (see Figure 4.8(b)).

classifier	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
GMM ML	7.77	7.77	7.73	7.10	8.87	7.93	8.40	9.20	11.77	19.87
GMM Hybrid	4.87	4.83	5.30	5.77	5.67	6.13	7.27	8.17	12.50	20.90
GMM MCL	7.10	6.93	7.03	6.80	7.90	7.43	7.77	8.97	11.57	19.70
GMM MM	4.70	4.70	5.53	5.87	6.40	7.50	7.63	8.80	11.03	19.10
SVM Lin. (M)	4.63	5.00	5.60	7.73	12.27	21.30	37.63	57.17	72.60	84.07
SVM Lin. (3NN)	4.63	4.50	4.00	3.87	4.67	5.23	6.07	7.78	11.20	20.17
SVM RBF (M)	1.97	2.37	3.17	7.80	18.17	31.97	50.10	68.70	80.47	88.60
SVM RBF (3NN)	1.97	2.13	2.27	2.77	3.27	3.87	5.43	6.67	10.73	19.37

Table B.3: Classification errors (%) of several classifiers for different numbers of missing features on USPS (see Figure 4.8(c)).

classifier	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
GMM ML	30.02	30.47	30.99	31.73	32.20	34.00	35.36	38.93	45.87	59.10
GMM Hybrid	22.19	24.57	27.58	30.19	33.71	38.33	44.06	50.46	58.38	69.28
GMM MCL	28.53	30.31	32.67	35.32	37.94	41.95	46.12	50.91	57.95	66.80
GMM MM	23.63	24.67	25.78	27.32	29.48	32.17	35.58	41.21	49.42	62.77
SVM Lin. (M)	26.06	31.29	37.78	43.96	50.96	57.79	64.47	70.49	77.87	85.22
SVM Lin. (3NN)	26.06	26.89	28.96	30.30	32.27	36.06	39.44	44.79	53.52	67.66
SVM RBF (M)	20.30	25.21	30.47	37.28	46.58	56.87	67.51	76.40	85.59	91.26
SVM RBF (3NN)	20.30	21.54	22.95	25.48	27.92	32.03	36.44	43.70	53.10	68.63

Table B.4: Classification errors (%) of several classifiers for different numbers of missing features on TIMIT (see Figure 4.8(d)).

λ	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
0	12.27	12.10	12.73	12.73	13.07	13.93	14.90	15.70	19.20	28.23
10^{-1}	11.83	11.67	12.23	12.37	12.90	13.53	14.43	15.53	19.00	28.17
10^{0}	9.87	10.00	10.33	11.10	11.03	11.73	13.37	13.93	17.87	27.93
10^{1}	7.93	8.17	8.37	9.23	9.53	10.57	12.60	13.83	19.83	30.47
10^{2}	7.57	7.70	9.03	9.47	9.97	12.37	15.47	16.90	23.20	34.90
10^{3}	6.93	7.33	7.97	9.20	9.63	11.30	14.60	17.40	23.93	36.93
10^{4}	6.30	6.60	8.13	9.97	12.17	15.73	20.33	23.93	33.60	49.93

Table B.5: Classification errors (%) for different values of λ and different numbers of missing features on USPS with fixed $K = 2$ and $\gamma = 2$ of hybrid GMMs (see Figure 4.10(a)).

λ	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
0	35.15	35.47	36.08	36.03	37.39	39.12	39.98	43.53	49.12	60.62
10^{-1}	32.31	32.91	33.23	34.00	35.07	37.17	38.93	42.57	48.69	60.39
10^{0}	27.32	27.99	28.26	30.08	31.15	32.81	36.65	40.72	48.63	61.07
10^{1}	23.62	25.09	27.24	29.18	31.80	35.54	40.58	46.75	54.72	66.09
10^{2}	23.09	26.38	31.44	35.57	41.92	47.58	53.63	61.71	70.43	78.14
10^{3}	23.31	34.86	45.42	53.64	60.85	67.58	72.99	78.03	84.70	88.79
10^{4}	23.49	40.30	52.59	60.52	67.58	73.79	79.07	81.83	86.51	89.27

Table B.6: Classification errors (%) for different values of λ and different numbers of missing features on TIMIT with fixed $K = 4$ and $\gamma = 32$ of hybrid GMMs (see Figure 4.10(b)).

classifier	500	1000	2500	5000	10000	25000	50000
GMM Hybrid (SV)	18.67	14.01	9.22	6.88	5.17	4.01	3.20
GMM Hybird (SSL)	13.94	10.61	7.14	5.29	4.48	3.72	3.20
SVM Linear (SV)	18.69	13.14	9.63	8.31	7.64	7.00	6.37
SVM RBF (SV)	14.38	9.40	6.01	4.22	3.02	2.01	1.62

Table B.7: Classification errors (%) of several classifiers for different numbers of labeled data examples N_l. The supervised (SV) classifiers used only the labeled data examples. The semi-supervised classifier (SSL) additionally used the remaining $50000 - N_l$ data examples as unlabeled data (see Figure 4.11(a)).

N_u	0	5000	10000	20000	40000	45000
CE	6.88	6.47	6.73	5.84	5.69	5.29

Table B.8: Classification errors (%) of semi-supervised hybrid GMMs for different numbers of unlabeled data N_u with a fixed amount of 5000 labeled data examples (see Figure 4.11(b)).

dataset	GMM ML	GMM Hybrid	GMM MCL	GMM MM
MNIST	$K = 32$	$K = 8, \lambda = 2^9, \gamma = 2^6$	$K = 8$	$K = 8, \lambda = 10^{-2}$
MNIST50	$K = 128$	$K = 128, \lambda = 2^0, \gamma = 2^5$	$K = 64$	$K = 128, \lambda = 10^{-2}$
USPS	$K = 16$	$K = 2, \lambda = 2^{10}, \gamma = 2^6$	$K = 4$	$K = 4, \lambda = 10^{-2}$
TIMIT	$K = 16$	$K = 16, \lambda = 2^5, \gamma = 2^5$	$K = 2$	$K = 16, \lambda = 10^{-2}$
Iris	$K = 4$	$K = 4, \lambda = 2^0, \gamma = 0.1$	$K = 3$	/
Breast	$K = 4$	$K = 1, \lambda = 2^5, \gamma = 2^6$	$K = 6$	/
Segment.	$K = 3$	$K = 1, \lambda = 2^5, \gamma = 2^6$	$K = 4$	/

dataset	SVM Linear	SVM RBF
MNIST	$\lambda = 2^{-4}$	$\lambda = 2^4, \gamma = 2^{-6}$
MNIST50	$\lambda = 2^{-1}$	$\lambda = 2^1, \gamma = 2^{-5}$
USPS	$\lambda = 2^{-8}$	$\lambda = 2^2, \gamma = 2^{-8}$
TIMIT	$\lambda = 2^{-4}$	$\lambda = 2^2, \gamma = 2^{-7}$
Iris	$\lambda = 2^{-1}$	$\lambda = 2^0, \gamma = 2^{-3}$
Breast	$\lambda = 2^{-3}$	$\lambda = 2^3, \gamma = 2^{-6}$
Segment.	$\lambda = 2^4$	$\lambda = 2^9, \gamma = 2^{-7}$

Table B.9: The hyperparameters which achieved the best classification performance without missing features (see Table 4.1).

N_l	Hybrid GMM (SV)	Hybrid GMM (SSL)	SVM Linear	SVM RBF
500	$K = 2, \lambda = 2^0, \gamma = 2^6$	$K = 2, \lambda = 2^7, \gamma = 2^4$	$\lambda = 2^{-6}$	$\lambda = 2^1, \gamma = 2^{-6}$
1000	$K = 2, \lambda = 2^0, \gamma = 2^5$	$K = 2, \lambda = 2^6, \gamma = 2^4$	$\lambda = 2^{-5}$	$\lambda = 2^1, \gamma = 2^{-6}$
2500	$K = 4, \lambda = 2^0, \gamma = 2^5$	$K = 16, \lambda = 2^4, \gamma = 2^5$	$\lambda = 2^{-4}$	$\lambda = 2^2, \gamma = 2^{-6}$
5000	$K = 16, \lambda = 2^0, \gamma = 2^5$	$K = 64, \lambda = 2^2, \gamma = 2^5$	$\lambda = 2^{-3}$	$\lambda = 2^2, \gamma = 2^{-6}$
10000	$K = 32, \lambda = 2^0, \gamma = 2^5$	$K = 64, \lambda = 2^1, \gamma = 2^5$	$\lambda = 2^{-4}$	$\lambda = 2^2, \gamma = 2^{-5}$
25000	$K = 32, \lambda = 2^0, \gamma = 2^4$	$K = 32, \lambda = 2^2, \gamma = 2^4$	$\lambda = 2^{-1}$	$\lambda = 2^3, \gamma = 2^{-5}$
50000	$K = 128, \lambda = 2^0, \gamma = 2^5$	/	$\lambda = 2^{-1}$	$\lambda = 2^1, \gamma = 2^{-5}$

Table B.10: The hyperparameters which achieved the best classification performance in the experiment for semi-supervised learning (see Figure 4.11(a)).

C
List of Acronyms

EM	expectation maximization
GMM	Gaussian mixture model
KL	Kullback-Leibler
MAP	maximum a-posteriori
MAR	missing at random
MCAR	missing completely at random
MCL	maximum conditional likelihood
ML	maximum likelihood
MM	maximum margin
NMAR	not missing at random
PCA	principal component analysis
RBF	radial basis function
SVM	support vector machine

Bibliography

[1] J. Lasserre and C. M. Bishop, "Generative or Discriminative? Getting the Best of Both Worlds," *Bayesian Statistics*, vol. 8, pp. 3–24, 2007.

[2] R. Peharz, S. Tschiatschek, and F. Pernkopf, "The Most Generative Maximum Margin Bayesian Networks," in *International Conference on Machine Learning, JMLR W&CP*, vol. 28, 2013, pp. 235–243.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed. Springer, 2006.

[4] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281–305, 2012.

[5] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of The 30th International Conference on Machine Learning*, 2013, pp. 115–123.

[6] L. Prechelt, "Early Stopping — But When?" in *Neural Networks: Tricks of the Trade*, ser. Lecture Notes in Computer Science, G. Montavon, G. Orr, and K.-R. Müller, Eds. Springer Berlin Heidelberg, 2012, vol. 7700, pp. 53–67.

[7] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge university press, 2009.

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

[9] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[10] F. Pernkopf and M. Wohlmayr, "Large Margin Learning of Bayesian Classifiers based on Gaussian Mixture Models," in *European Conference on Machine Learning (ECML)*, ser. Lecture notes in computer science, vol. 6323, Springer. Springer, Sep. 2010, pp. 50–55.

[11] C. Cortes and V. N. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[12] K. Crammer and Y. Singer, "On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines," *J. Mach. Learn. Res.*, vol. 2, pp. 265–292, 2002.

[13] Y. Guo, D. Wilkinson, and D. Schuurmans, "Maximum Margin Bayesian Networks," in *Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI2005)*, 2005, pp. 233–242.

[14] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," *Advances in Neural Information Processing Systems*, vol. 14, p. 841, 2002.

[15] F. Sha, "Large margin training of acoustic models for speech recognition," Ph.D. dissertation, University of Pennsylvania, 2007.

[16] B. M. Marlin, "Missing Data Problems in Machine Learning," Ph.D. dissertation, University of Toronto, 2008.

[17] R. J. A. Little and D. B. Rubin, *Statistical analysis with missing data*. Wiley, 2002.

[18] P. Jonsson and C. Wohlin, "An Evaluation of k-Nearest Neighbour Imputation Using Likert Data," in *Proceedings of the Software Metrics, 10th International Symposium*, ser. METRICS '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 108–118.

[19] O. Chapelle, B. Schölkopf, A. Zien, and Others, *Semi-supervised learning*. MIT press Cambridge, 2006.

[20] X. Zhu, A. B. Goldberg, R. Brachman, and T. Dietterich, *Introduction to Semi-Supervised Learning*. Morgan and Claypool Publishers, 2009.

[21] F. G. Cozman, I. Cohen, M. C. Cirelo, and Others, "Semi-supervised learning of mixture models," in *International Conference on Machine Learning*, 2003, pp. 99–106.

[22] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 2006, vol. 2.

[23] J. Nocedal, "Updating quasi-Newton matrices with limited storage," *Mathematics of computation*, vol. 35, no. 151, pp. 773–782, 1980.

[24] A. Ben-Hur and J. Weston, "A User's Guide to Support Vector Machines," in *Data Mining Techniques for the Life Sciences*. Springer, 2010, pp. 223–239.

[25] S. Ertekin, L. Bottou, and C. L. Giles, "Nonconvex online support vector machines," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 2, pp. 368–381, 2011.

[26] Z. Ghahramani and M. I. Jordan, "Supervised learning from incomplete data via an EM approach," in *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, 1994, pp. 120–127.

[27] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell, "Text classification from labeled and unlabeled documents using EM," *Machine learning*, vol. 39, no. 2-3, pp. 103–134, 2000.

[28] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[29] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[30] F. Pernkopf and M. Wohlmayr, "On Discriminative Parameter Learning of Bayesian Network Classifiers," in *European Conference on Machine Learning (ECML 2009)*, Bled, Sep. 2009, pp. 221–237.

[31] M. Ratajczak, S. Tschiatschek, and F. Pernkopf, "Neural and Higher-Order Factors in Conditional Random Fields for Phoneme Classification," in *Interspeech*, 2015.

[32] M. Lichman, "{UCI} Machine Learning Repository," 2013.

[33] K. B. Petersen and M. S. Pedersen, *The Matrix Cookbook*. Citeseer, 2012.