FEATURE SELECTION AND TRANSFER LEARNING ALGORITHMS WITH APPLICATIONS ON CREDIT RISK ANALYSIS

by

Gül Efşan Bozkurt Gönen

B.S., Computer Engineering, Middle East Technical University, 2005

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University

2012

FEATURE SELECTION AND TRANSFER LEARNING ALGORITHMS WITH APPLICATIONS ON CREDIT RISK ANALYSIS

APPROVED BY:

Prof. Fikret Gürgen (Thesis Supervisor)	.....
Prof. Lale Akarun	.....
Assist. Prof. Olcay Kurşun	.....

DATE OF APPROVAL: 15.06.2012

To my lovely father Hüseyin Bozkurt ...

ABSTRACT

FEATURE SELECTION AND TRANSFER LEARNING ALGORITHMS WITH APPLICATIONS ON CREDIT RISK ANALYSIS

Many financial organizations such as banks and retailers use computational credit risk analysis (CRA) tools heavily due to recent financial crises and more strict regulations. This strategy enables them to manage their financial and operational risks within the pool of financial institutes. Machine learning algorithms especially binary classifiers are very popular for that purpose. In real-life applications such as CRA, feature selection algorithms are used to decrease data acquisition cost and to increase interpretability of the decision process. Using feature selection methods directly on CRA data sets may not help due to categorical variables such as marital status. Such variables are usually are converted into binary features using 1-of-k encoding and eliminating a subset of features from a group does not help in terms of data collection cost or interpretability. In this thesis, we propose to use the probit classifier with a proper prior structure and multiple kernel learning with a proper kernel construction procedure to perform group-wise feature selection. Experiments on two standard CRA data sets show the validity and effectiveness of the proposed binary classification algorithm variants. Robustness against dynamic conditions such as currency changes is another important property for CRA systems. They should perform reasonably good with limited data after such changes and the best strategy is to exploit existing data using transfer learning. We also extend the probit classifier towards transfer learning by mapping different data sets into a unified subspace and learning a common classifier. Experiments on two standard CRA data sets show the usefulness of transfer learning for such cases.

ÖZET

ÖZNİTELİK SEÇME VE TRANSFER ÖĞRENME ALGORİTMALARI VE KREDİ RİSK ANALİZİ ÜZERİNE UYGULAMALARI

Bankalar ve perakendeciler gibi birçok finansal kurum son zamanlardaki finansal krizler ve daha katı düzenlemeler nedeniyle ağırlıkla hesaplamalı kredi risk analiz (KRA) araçlarını kullanmaktadır. Bu strateji onların finansal kurumlar havuzundaki finansal ve operasyonel risklerini yönetebilmelerini sağlar. Yapay öğrenme algoritmaları özellikle ikili sınıflandırıcılar bu amaç için çok uygundur. Öznitelik seçme algoritmaları KRA benzeri uygulamalarda veri toplama maliyetini düşürmek ve karar mekanizmasının yorumlanabilirliğini artırmak için kullanılır. Öznitelik seçme yöntemlerinin KRA veri kümelerine doğrudan uygulanması medeni durum gibi kategorik değişkenler nedeniyle başarılı olmayabilir. Buna benzer değişkenler genellikle ikili özniteliklere çevrilir ve özniteliklerin belli bir kısmını elemek veri toplama maliyeti ya da yorumlanabilirlik açısından yardımcı olmaz. Bu tezde çoklu öznitelik seçimi için özel bir öncül dağılım kullanan probit sınıflandırıcı ve özel bir çekirdek hesaplama yöntemi kullanan çoklu çekirdek öğrenimi yöntemleri geliştirdik. İki standart KRA veri kümesi üzerindeki deneyler önerilen ikili sınıflandırma algoritmalarının geçerliliğini ve etkinliğini gösterdi. KRA sistemleri için başka önemli bir özellik ise para birimi değişiklikleri gibi dimanik koşullara dayanıklıktır. Buna benzer değişikliklerden sonra sınırlı miktarda veri ile makul bir şekilde çalışmaları beklenmektedir ve transfer öğrenimi ile mevcut veriden faydalanılması bunun için en iyi stratejidir. Değişik veri kümelerini ortak bir altuzaya taşıyarak ve burada ortak bir sınıflandırıcı öğrenerek probit sınıflandırıcıyı transfer öğrenimine uyarladık. İki standart KRA veri kümesi üzerindeki deneyler transfer öğreniminin benzer durumlardaki faydasını gösterdi.

TABLE OF CONTENTS

ABSTRACT iv
ÖZET v
LIST OF FIGURES viii
LIST OF TABLES x
LIST OF SYMBOLS xi
LIST OF ACRONYMS/ABBREVIATIONS xiv
1. INTRODUCTION 1
1.1. Outline of the Thesis 3
2. RELATED WORK 4
3. GROUP-WISE FEATURE SELECTION USING PROBIT CLASSIFIER . . 7
3.1. Inference Details 9
3.2. Prediction 13
3.3. Group-Wise Feature Selection 14
4. GROUP-WISE FEATURE SELECTION USING MULTIPLE KERNEL LEARNING 16
4.1. Training Details 19
4.2. Prediction 20
4.3. Group-Wise Feature Selection 20
5. TRANSFER LEARNING USING PROBIT CLASSIFIER 22
5.1. Inference Details 25
5.2. Prediction 31
6. EXPERIMENTS 33
6.1. Data Sets 33
6.1.1. Australian Credit Data Set 33
6.1.2. German Credit Data Set 33
6.2. Performance Measures 34
6.2.1. Classification Accuracy 34
6.2.2. Area Under Curve 36
6.3. Feature Selection Experiments 37

6.3.1. Experimental Setup 37
6.3.2. Results 38
6.3.3. Convergence Analysis 45
6.4. Transfer Learning Experiments 48
6.4.1. Experimental Setup 49
6.4.2. Results 49
7. CONCLUSIONS 52
APPENDIX A: MATLAB SOURCE CODES 54
REFERENCES 63

LIST OF FIGURES

Figure 3.1. Graphical model and distributional assumptions of the probit classifier. 7

Figure 3.2. Inference algorithm of the probit classifier. 11

Figure 4.1. Training algorithm of the ℓ_p-norm MKL. 20

Figure 5.1. Graphical model and distributional assumptions of the transfer learning probit classifier. 23

Figure 5.2. Inference algorithm of the transfer learning probit classifier. 27

Figure 6.1. Feature weights obtained by `PROBIT` on `Australian Credit` data set in feature selection experiments. 40

Figure 6.2. Feature weights obtained by ℓ_p-`MKL` on `Australian Credit` data set in feature selection experiments. 41

Figure 6.3. Feature weights obtained by `PROBIT` on `German Credit` data set in feature selection experiments. 43

Figure 6.4. Feature weights obtained by ℓ_p-`MKL` on `German Credit` data set in feature selection experiments. 44

Figure 6.5. `PROBIT` lower bounds on `Australian Credit` data set in feature selection experiments. 45

Figure 6.6. ℓ_p-MKL iteration counts on Australian Credit data set in feature selection experiments. 46

Figure 6.7. PROBIT lower bounds on German Credit data set in feature selection experiments. 47

Figure 6.8. ℓ_p-MKL iteration counts on German Credit data set in feature selection experiments. 48

Figure 6.9. Classification results on Australian Credit data set in transfer learning experiments. 50

Figure 6.10. Classification results on German Credit data set in transfer learning experiments. 51

LIST OF TABLES

Table 6.1. Variable information of `Australian Credit` data set. 34

Table 6.2. Variable information of `German Credit` data set. 35

Table 6.3. Confusion matrix for binary classification problems. 36

Table 6.4. Parameter values for different scenarios of feature selection experiments. 37

Table 6.5. Classification results on `Australian Credit` data set in feature selection experiments. 38

Table 6.6. Variables selected by `PROBIT` on `Australian Credit` data set in feature selection experiments. 39

Table 6.7. Variables selected by ℓ_p-`MKL` on `Australian Credit` data set in feature selection experiments. 39

Table 6.8. Classification results on `German Credit` data set in feature selection experiments. 41

Table 6.9. Variables selected by `PROBIT` on `German Credit` data set in feature selection experiments. 42

Table 6.10. Variables selected by ℓ_p-`MKL` on `German Credit` data set in feature selection experiments. 42

Table 6.11. Parameter values of transfer learning experiments. 49

LIST OF SYMBOLS

$\lvert \cdot \rvert$	Determinant
$[\cdot]$	Index
$\langle \cdot, \cdot \rangle$	Inner product
$\lVert \cdot \rVert_p$	ℓ_p-norm
$\widetilde{\cdot}$	Posterior expectation
$\top$	Transpose
b	Bias term
C	Trade-off parameter
D	Dimensionality of the original feature space
$\mathrm{diag}(\cdot)$	Diagonal matrix function
$\mathrm{E}[\cdot]$	Expectation
$\exp(\cdot)$	Exponential function
$f(\cdot)$	Discriminant function
$g(\cdot)$	Group function
$\mathcal{G}(\cdot; \cdot, \cdot)$	Gamma distribution
$k(\cdot, \cdot)$	Kernel function
$\mathcal{L}$	Lower bound
$\log(\cdot)$	Logarithm function
N	Number of training instances
$\mathbb{N}$	Natural numbers
$\mathcal{N}(\cdot; \cdot, \cdot)$	Normal distribution
$p(\cdot)$	Probability distribution
P	Number of kernels to be combined
$q(\cdot)$	Approximate posterior distribution
R	Subspace dimensionality
$\mathbb{R}$	Real numbers
$\mathbb{R}_+$	Nonnegative real numbers
$\mathbb{R}_{++}$	Positive real numbers

S	Dimensionality of mapped feature space
T	Number of data sets
$\mathcal{TN}(\cdot;\cdot,\cdot,\cdot)$	Truncated normal distribution
$\mathrm{tr}(\cdot)$	Trace function
$\boldsymbol{x}$	Data point
$\mathbf{X}$	Data points
$\boldsymbol{w}$	Weight coefficients
y	Class label
$\boldsymbol{y}$	Class labels
$\mathcal{Z}$	Normalization coefficient
α	Shape parameter or support vector coefficient
$\alpha(\cdot)$	Shape function
$\boldsymbol{\alpha}$	Support vector coefficients
β	Scale parameter
$\beta(\cdot)$	Scale function
$\Gamma(\cdot)$	Gamma function
$\delta(\cdot)$	Kronecker delta; 1 if the parameter is true, 0 otherwise
$\boldsymbol{\zeta}$	Hyper-parameters
η	Kernel weight
$\boldsymbol{\eta}$	Kernel weights
$\boldsymbol{\Theta}$	Latent variables
$\mu(\cdot)$	Mean function
$\boldsymbol{\mu}$	Mean vector
ξ	Slack variable
$\boldsymbol{\xi}$	Slack variables
$\boldsymbol{\Xi}$	Prior variables
$\rho(\cdot)$	Truncation rule
$\Sigma(\cdot)$	Covariance function
$\boldsymbol{\Sigma}$	Covariance matrix
$\phi(\cdot)$	Standardized normal probability density function

$\Phi(\cdot)$	Standardized normal cumulative distribution function or mapping function
$\psi(\cdot)$	Digamma function

LIST OF ACRONYMS/ABBREVIATIONS

AUC	Area Under Curve
CRA	Credit Risk Analysis
DT	Decision Tree
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GA	Genetic Algorithm
MKL	Multiple Kernel Learning
NN	Neural Network
SVM	Support Vector Machine
ROC	Receiver Operating Characteristic
TN	True Negative
TP	True Positive
TPR	True Positive Rate

1. INTRODUCTION

Credit risk is the loss of capital in case of the credit borrower's failure for refunding the total amount of debt to recover the liability. *Credit risk analysis* (CRA) is an important topic in the financial management field used by many financial organizations such as banks and retailers. Due to recent financial crises and more strict regulations, CRA becomes the major focus point of financial and banking industry since accurate estimation of credit risks enables a more efficient funding for world economy (Basel I, 1988).

Banks are required to manage financial and operational risks for providing a safer environment for them within the pool of all international banks (Basel II, 2004). A safer financial environment facilitates the transmission of money for convenient use in the economy. If a bank or financial organization wants to accomplish long term success, it should follow an exhaustive and powerful strategy for CRA (Basel III, 2011). Measuring the credit risk accurately also allows banks to organize upcoming lending transactions to achieve targeted return/risk characteristics. Nowadays, financial organizations are building their own software solutions to analyze their gathered data. Another benefit of CRA is for accounting companies. If an accounting company monitors a potentially troubled company and forgets to notify the credit borrower with a warning signal, then the company can face a costly lawsuit. Therefore, as the credit industry expands, CRA methods become used comprehensively to evaluate the credit applications (Thomas, 2000).

Up to now, many computational methods are proposed for CRA (Atiya, 2001). CRA methods generally aim to classify credit applicants into two groups, namely, *approved* and *disapproved*, according to properties of the applicants such as income, profession, possession, marital status, the number of people liable to look after, previous credit history, and age. Credit suppliers want to increase the volume of credit supply without increasing the failure ratio extremely (Huang *et al.*, 2007) and developing reliable computational models is the key to successful credit operations. CRA methods

also ensure better examination of existing accounts, faster results in decision processing, and better precedence assignments for credit collections (Brill, 1998).

The main motivation of computational CRA methods is to have a robust binary classification algorithm for classifying credit applicants or a robust clustering algorithm for assigning them into predefined applicant categories. Most of the existing solutions in the literature formulate the problem as a binary classification problem and apply the standard classification algorithms such as *decision trees* (DTs), *neural networks* (NNs), and *support vector machines* (SVMs). The complexity of such learning algorithms mostly depends on the number of input features. Feature selection algorithms are proposed to reduce the number of features used for prediction. It is not necessary to use all of the input features since redundant features do not provide any useful information for classification. When we are able to explain the data with fewer features, we acquire better knowledge about the process that generates this data (Alpaydın, 2010). In this thesis, we also follow these lines of research using probabilistic and discriminative classification algorithms, namely, the probit classifier and *multiple kernel learning* (MKL), coupled with feature selection capability for lower data acquisition cost, better interpretability of the decision process, and lower test time complexity.

CRA problems usually contain categorical variables (e.g., marital status) and these variables are converted into binary features using 1-of-k encoding. When we have grouped features such as these, performing feature selection at the feature level does not produce sparse results because not eliminating a single feature from a group requires to collect data about the corresponding variable for test data points (i.e., new credit applicants in our case). We propose to use the probit classifier with a proper prior structure and MKL with a proper kernel construction procedure to perform feature selection in a group-wise manner. By doing these, we can decide whether we need to include a categorical variable into the final decision function or not.

Transfer learning algorithms are proposed to exploit different data sets from related tasks and to learn better predictors for them. These methods are especially useful when some of the data sets do not have enough training data to learn reasonable mod-

els. In real-life scenarios, CRA models have to be robust against dynamic conditions such as currency changes, new economic regulations, and updated data collection procedures. Existing data sets are considered as obsolete and usually discarded after such changes. Instead of discarding, we can make use of existing data sets with a transfer learning algorithm. This strategy allows us to obtain better predictors because there is not enough training data just after a major change.

We also extend the probit classifier towards transfer learning by mapping different data sets into a unified subspace and learning a shared classifier in that common subspace. Our proposed method can transfer information between different data sets to learn a better predictor for each of them.

1.1. Outline of the Thesis

In Chapter 2, we give an overview of the related work by considering existing machine learning solutions for CRA. Chapter 3 introduces the probit classifier and its inference mechanism with a deterministic variational approximation, and explains the details of our proposed extension towards group-wise feature selection. Chapter 4 introduces MKL and shows how we can perform group-wise feature selection with suitable kernel calculations and sparsity on the kernel-level. Chapter 5 extends the probit classifier towards transfer learning and gives the details of its inference mechanism with a deterministic variational approximation. In Chapter 6, we evaluate the performances of our proposed group-wise feature selection and transfer learning methods on two well-known CRA data sets. In Chapter 7, we summarize our contributions and conclude the thesis.

2. RELATED WORK

There are two main categories for CRA methods: (i) structural approaches and (ii) statistical approaches. Structural approaches directly depend on some financial measures of the credit applicant such as total assets, yearly profit/loss rate, and growth rate. The credit interest rate is decided by looking at these measures (Kotsiantis *et al.*, 2005). The most obvious problem of such approaches is the lack of proper mathematical or statistical tools. Statistical approaches depend on empirical tools that use the credit history to build a predictor used for new credit applicants. Different machine learning algorithms such as NNs and SVMs are applied to CRA problems. In this thesis, we are focusing on the second approach by considering two different classification schemes for CRA. We first give a structured review of the recent machine learning studies by considering commonly used methods.

NNs are frequently used for credit risk estimation due to the fact that most of the existing statistical softwares include them as standard computational tools (Atiya, 2001). For example, Angelini *et al.* (2008) give a successful application scenario for Italian small businesses using two different NN strategies. Abdou *et al.* (2008) compare NNs with other standard methods for CRA of Egyptian banks and obtain the best results using NNs. Yao *et al.* (2009) propose a CRA method using fuzzy NNs for Chinese commercial banks. Khashman (2010) compare different learning algorithms for supervised NNs on German credit data set. Derelioğlu and Gürgen (2011) propose a rule extraction system using a NN-based approach for CRA of small medium enterprises in Turkey.

SVMs are also excessively used for CRA applications due to their good empirical performance. Huang *et al.* (2004) show that SVMs slightly outperform NNs for CRA on two data sets from Taiwanese financial institutes and USA commercial banks. Chen and Shih (2006) also report a very similar result on a Taiwan banking data set. Yongqiao *et al.* (2005) propose a new fuzzy SVM algorithm for classifying credit applicants. Van Gestel *et al.* (2006) develop a Bayesian least squares SVM classifier and test the

classifier on a commercial credit data set based on Belgian and Dutch firms. Li *et al.* (2006) formulates a least squares SVM classifier for joint classification and feature selection to provide interpretability using MKL framework. Huang *et al.* (2007) show that SVMs outperform NNs and DTs on two standard credit risk data sets and propose a hybrid method that performs feature selection for SVMs using *genetic algorithms* (GAs). Yoon and Kwon (2010) propose a CRA method built on credit card sales information using SVMs to solve the missing financial data problem. Kim and Sohn (2010) build a credit risk estimation method for Korean small-and-medium enterprises using SVMs.

Inspired from SVMs, multiple criteria programming framework is proposed for classification and applied to CRA problems (Shi, 2010). Peng *et al.* (2008) introduce a multiple criteria convex quadratic programming model that tries to maximize the intra-class distance between classes and to minimize the within-class distance, and test the proposed algorithm on four different CRA data sets. Li *et al.* (2011) extend the same idea with a combination of GAs and MKL towards coupled classification and feature selection.

No single classification algorithm can produce the best results for all classification problems. There are two standard approaches to solve this issue: (i) classifier selection and (ii) classifier combination. In classifier selection, different classifiers are trained and evaluated using a cross-validation approach. At the end, the best performing classifier is used for testing the system. Instead of relying on a single classifier, we can construct a meta-classifier that combines the predictions of multiple classifiers, known as classifier combination or classifier ensemble. This combination strategy is also applied to CRA extensively. Lai *et al.* (2006a) propose to use an NN ensemble by training a diverse set of networks and combining uncorrelated ones to obtain a reliable prediction scheme. Lai *et al.* (2006b) also formulate an NN ensemble method by training different NNs on different subsets of the training data (known as bagging) and combining the predictions of these networks with another NN to get the final prediction. Huang *et al.* (2006) examine various classification algorithms and construct classifier ensembles using random committee and voted perceptron techniques to evaluate the

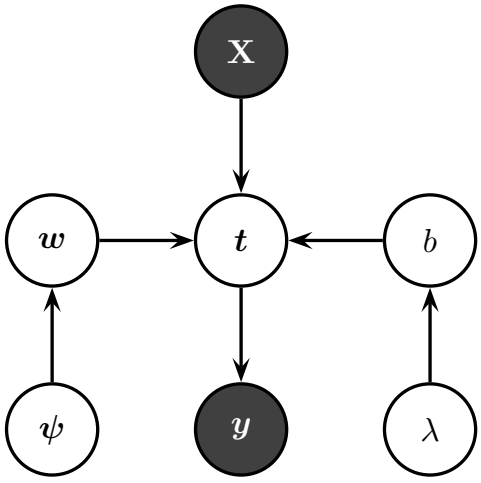
customers of a Chinese bank. Hsieh and Hung (2010) present a CRA method that uses NNs, SVMs, and Bayesian networks as the base classifiers of the ensemble. Zhou *et al.* (2010) offer an ensemble method that uses least squares SVMs with different kernels for the combination. Twala (2010) compares different ensemble strategies and shows that using a classifier ensemble outperforms single classifiers on four different CRA data sets. Peng *et al.* (2011) propose three different methods to compare and to combine base classifiers using their predictions as inputs.

In addition to such machine learning methods, there are also evolutionary algorithms such as GAs proposed for CRA. Finlay (2009) applies a GA approach to optimize business measures instead of a statistical model objective as in machine learning algorithms. Min and Jeong (2009) propose a binary classifier using a GA-based formulation and obtain comparable results to statistical approaches.

3. GROUP-WISE FEATURE SELECTION USING PROBIT CLASSIFIER

We use the probit classifier, which is a generalized linear model, as our probabilistic classification method (Albert and Chib, 1993). We use a fully Bayesian formulation to give the classifier feature selection capability using suitable hyper-parameters for the prior distributions. We first describe the details of our probabilistic model and then explain how this model can be used for coupled feature selection and classification.

Figure 3.1 illustrates the probit classifier with a graphical model and its distributional assumptions. First, the data matrix $\mathbf{X}$ is used to calculate the classification scores for training data points using the classification parameters $\{b, \boldsymbol{w}\}$. Finally, the given label vector $\boldsymbol{y}$ is generated from the classification score vector $\boldsymbol{t}$.



$$
\begin{aligned}
\lambda &\sim \mathcal{G}(\lambda; \alpha_\lambda, \beta_\lambda) && \\
b|\lambda &\sim \mathcal{N}(b; 0, \lambda^{-1}) && \\
\psi_f &\sim \mathcal{G}(\psi_f; \alpha_\psi, \beta_\psi) && \forall f \\
w_f|\psi_f &\sim \mathcal{N}(w_f; 0, \psi_f^{-1}) && \forall f \\
t_i|b, \boldsymbol{w}, \boldsymbol{x}_i &\sim \mathcal{N}(t_i; \boldsymbol{w}^\top \boldsymbol{x}_i + b, 1) && \forall i \\
y_i|t_i &\sim \delta(t_i y_i > 0) && \forall i
\end{aligned}
$$

Figure 3.1. Graphical model and distributional assumptions of the probit classifier.

The notation we use for the probit classifier is as follows: N is the number of training instances. D shows the dimensionality of the input space. The $D \times N$ data matrix is denoted by $\mathbf{X}$, where the $D \times 1$-dimensional columns of $\mathbf{X}$ by $\boldsymbol{x}_i$. The $D \times 1$ vector of weight parameters w_f is denoted by $\boldsymbol{w}$. The $D \times 1$ vector of priors ψ_f is denoted by $\boldsymbol{\psi}$. The bias parameter is denoted by b and its prior is denoted by λ. The $N \times 1$ vector of auxiliary variables t_i is represented as $\boldsymbol{t}$. The $N \times 1$ vector of associated target

values is represented as $\boldsymbol{y}$, where each element $y_i \in \{-1, +1\}$. As short-hand notations, all priors in the model are denoted by $\boldsymbol{\Xi} = \{\lambda, \boldsymbol{\psi}\}$, where the remaining variables by $\boldsymbol{\Theta} = \{b, \boldsymbol{t}, \boldsymbol{w}\}$ and the hyper-parameters by $\boldsymbol{\zeta} = \{\alpha_\lambda, \beta_\lambda, \alpha_\psi, \beta_\psi\}$. Dependence on $\boldsymbol{\zeta}$ is omitted for clarity throughout the rest. $\mathcal{N}(\cdot; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ denotes the normal distribution with the mean vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$. $\mathcal{G}(\cdot; \alpha, \beta)$ denotes the gamma distribution with the shape parameter α and the scale parameter β. $\delta(\cdot)$ denotes the Kronecker delta function that returns 1 if its argument is true and 0 otherwise.

The auxiliary variables between the class labels and the training instances are introduced to make the inference procedures efficient (Albert and Chib, 1993). Exact inference for this probabilistic model is intractable and using a Gibbs sampling approach is computationally expensive (Gelfand and Smith, 1990). We instead formulate a deterministic variational approximation procedure for efficient inference.

The variational methods use a lower bound on the marginal likelihood using an ensemble of factored posteriors to find the joint parameter distribution (Beal, 2003). Assuming independence between the approximate posteriors in the factorable ensemble can be justified because there is not a strong coupling between the model parameters. We can write the factorable ensemble approximation of the required posterior as

$$p(\boldsymbol{\Theta}, \boldsymbol{\Xi} | \mathbf{X}, \boldsymbol{y}) \approx q(\boldsymbol{\Theta}, \boldsymbol{\Xi}) = q(\lambda) q(\boldsymbol{\psi}) q(b, \boldsymbol{w}) q(\boldsymbol{t})$$

and define each factor in the ensemble just like its full conditional distribution:

$$\begin{aligned}
q(\lambda) &= \mathcal{G}(\lambda; \alpha(\lambda), \beta(\lambda)) \\
q(\boldsymbol{\psi}) &= \prod_{f=1}^{D} \mathcal{G}(\psi_f; \alpha(\psi_f), \beta(\psi_f)) \\
q(b, \boldsymbol{w}) &= \mathcal{N}\left(\begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix} ; \mu(b, \boldsymbol{w}), \Sigma(b, \boldsymbol{w}) \right) \\
q(\boldsymbol{t}) &= \prod_{i=1}^{N} \mathcal{TN}(t_i; \mu(t_i), \Sigma(t_i), \rho(t_i))
\end{aligned}$$

where $\alpha(\cdot)$, $\beta(\cdot)$, $\mu(\cdot)$, and $\Sigma(\cdot)$ denote shape parameter, scale parameter, mean vector, and covariance matrix for their arguments, respectively. $\mathcal{TN}(\cdot; \boldsymbol{\mu}, \boldsymbol{\Sigma}, \rho(\cdot))$ denotes the truncated normal distribution with the mean vector $\boldsymbol{\mu}$, the covariance matrix $\boldsymbol{\Sigma}$, and the truncation rule $\rho(\cdot)$ such that $\mathcal{TN}(\cdot; \boldsymbol{\mu}, \boldsymbol{\Sigma}, \rho(\cdot)) \propto \mathcal{N}(\cdot; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ if $\rho(\cdot)$ is true and $\mathcal{TN}(\cdot; \boldsymbol{\mu}, \boldsymbol{\Sigma}, \rho(\cdot)) = 0$ otherwise.

We can bound the marginal likelihood using Jensen's inequality:

$$\log p(\boldsymbol{y}|\mathbf{X}) \geq \mathrm{E}_{q(\boldsymbol{\Theta},\boldsymbol{\Xi})}[\log p(\boldsymbol{y}, \boldsymbol{\Theta}, \boldsymbol{\Xi}|\mathbf{X})] - \mathrm{E}_{q(\boldsymbol{\Theta},\boldsymbol{\Xi})}[\log q(\boldsymbol{\Theta}, \boldsymbol{\Xi})] \tag{3.1}$$

and optimize this bound by maximizing with respect to each factor separately until convergence. The approximate posterior distribution of a specific factor $\boldsymbol{\tau}$ is

$$q(\boldsymbol{\tau}) \propto \exp\left(\mathrm{E}_{q(\{\boldsymbol{\Theta},\boldsymbol{\Xi}\}\setminus\boldsymbol{\tau})}[\log p(\boldsymbol{y}, \boldsymbol{\Theta}, \boldsymbol{\Xi}|\mathbf{X})]\right).$$

For the probit classifier, thanks to the conjugacy, the resulting approximate posterior distribution of each factor follows the same distribution as the corresponding factor.

3.1. Inference Details

The approximate posterior distributions of the priors on the bias and the weight vector can be found in terms of gamma distributions:

$$q(\lambda) = \mathcal{G}\left(\lambda; \alpha_\lambda + \frac{1}{2}, \left(\frac{1}{\beta_\lambda} + \frac{\widetilde{b^2}}{2}\right)^{-1}\right) \tag{3.2}$$

$$q(\boldsymbol{\psi}) = \prod_{f=1}^{D} \mathcal{G}\left(\psi_f; \alpha_\psi + \frac{1}{2}, \left(\frac{1}{\beta_\psi} + \frac{\widetilde{w_f^2}}{2}\right)^{-1}\right) \tag{3.3}$$

where the tilde notation denotes the posterior expectations as usual, i.e., $\widetilde{h(\boldsymbol{\tau})} = \mathrm{E}_{q(\boldsymbol{\tau})}[h(\boldsymbol{\tau})]$. The approximate posterior distribution of the classification parameters

is a product of multivariate normal distributions:

$$q(b, \boldsymbol{w}) = \mathcal{N}\left(\begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix} ; \Sigma(b, \boldsymbol{w}) \begin{bmatrix} \mathbf{1}^\top \widetilde{\boldsymbol{t}} \\ \mathbf{X}\widetilde{\boldsymbol{t}} \end{bmatrix}, \begin{bmatrix} \widetilde{\lambda} + N & \mathbf{1}^\top \mathbf{X}^\top \\ \mathbf{X}\mathbf{1} & \mathrm{diag}(\widetilde{\boldsymbol{\psi}}) + \mathbf{X}\mathbf{X}^\top \end{bmatrix}^{-1} \right). \tag{3.4}$$

The approximate posterior distribution of the auxiliary variables is a product of truncated normal distributions:

$$q(\boldsymbol{t}) = \prod_{i=1}^{N} \mathcal{TN}(t_i; \widetilde{\boldsymbol{w}^\top} \boldsymbol{x}_i + \widetilde{b}, 1, t_i y_i > 0) \tag{3.5}$$

where we need to find the posterior expectations in order to update the approximate posterior distributions of the projected instances and the classification parameters. Fortunately, the truncated normal distribution has a closed-form formula for its expectation.

The complete inference algorithm is listed in Figure 3.2. The inference mechanism sequentially updates the approximate posterior distributions of the model parameters and the latent variables until convergence, which can be checked by monitoring the lower bound in Equation 3.1. The first term of the lower bound corresponds to the sum of exponential form expectations of the distributions in the joint likelihood. The second term is the sum of negative entropies of the approximate posteriors in the ensemble. The only nonstandard distribution in the second term is the truncated normal distributions of the auxiliary variables; nevertheless, the truncated normal distribution has a closed-form formula also for its entropy.

The variational lower bound of probit classification model can be written as

$$\mathcal{L} = \mathrm{E}_{q(\boldsymbol{\Theta}, \boldsymbol{\Xi})}[\log p(\boldsymbol{y}, \boldsymbol{\Theta}, \boldsymbol{\Xi} | \mathbf{X})] - \mathrm{E}_{q(\boldsymbol{\Theta}, \boldsymbol{\Xi})}[\log q(\boldsymbol{\Theta}, \boldsymbol{\Xi})]$$

where the joint likelihood is defined as

$$p(\boldsymbol{y}, \boldsymbol{\Theta}, \boldsymbol{\Xi} | \mathbf{X}) = p(\lambda) p(b|\lambda) p(\boldsymbol{\psi}) p(\boldsymbol{w}|\boldsymbol{\psi}) p(\boldsymbol{t}|b, \boldsymbol{w}, \mathbf{X}) p(\boldsymbol{y}|\boldsymbol{t}).$$

Require: $\mathbf{X}$, $\boldsymbol{y}$, α_λ, β_λ, α_ψ, and β_ψ

Initialize $q(b, \boldsymbol{w})$ and $q(\boldsymbol{t})$ randomly
repeat
  Update $q(\lambda)$ using Equation 3.2
  Update $q(\boldsymbol{\psi})$ using Equation 3.3
  Update $q(b, \boldsymbol{w})$ using Equation 3.4
  Update $q(\boldsymbol{t})$ using Equation 3.5
until convergence
return $q(b, \boldsymbol{w})$

Figure 3.2. Inference algorithm of the probit classifier.

Using these definitions, the variational lower bound becomes

$$\begin{aligned}\mathcal{L} = &\underbrace{\mathrm{E}_{q(\lambda)}[\log p(\lambda)]}_{\mathcal{L}_1} + \underbrace{\mathrm{E}_{q(\lambda)q(b,\boldsymbol{w})}[\log p(b|\lambda)]}_{\mathcal{L}_2} + \underbrace{\mathrm{E}_{q(\boldsymbol{\psi})}[\log p(\boldsymbol{\psi})]}_{\mathcal{L}_3} + \underbrace{\mathrm{E}_{q(\boldsymbol{\psi})q(b,\boldsymbol{w})}[\log p(\boldsymbol{w}|\boldsymbol{\psi})]}_{\mathcal{L}_4} \\ &+ \underbrace{\mathrm{E}_{q(b,\boldsymbol{w})q(\boldsymbol{t})}[\log p(\boldsymbol{t}|b, \boldsymbol{w}, \mathbf{X})]}_{\mathcal{L}_5} + \underbrace{\mathrm{E}_{q(\boldsymbol{t})}[\log p(\boldsymbol{y}|\boldsymbol{t})]}_{\mathcal{L}_6} - \underbrace{\mathrm{E}_{q(\lambda)}[\log q(\lambda)]}_{\mathcal{L}_7} - \underbrace{\mathrm{E}_{q(\boldsymbol{\psi})}[\log q(\boldsymbol{\psi})]}_{\mathcal{L}_8} \\ &- \underbrace{\mathrm{E}_{q(b,\boldsymbol{w})}[\log q(b, \boldsymbol{w})]}_{\mathcal{L}_9} - \underbrace{\mathrm{E}_{q(\boldsymbol{t})}[\log q(\boldsymbol{t})]}_{\mathcal{L}_{10}}\end{aligned}$$

where the exponential form expectations of the distributions in the joint likelihood can be calculated as

$$\begin{aligned}\mathcal{L}_1 &= (\alpha_\lambda - 1)\widetilde{\log \lambda} - \frac{\widetilde{\lambda}}{\beta_\lambda} - \log \Gamma(\alpha_\lambda) - \alpha_\lambda \log \beta_\lambda \\ \mathcal{L}_2 &= -\frac{1}{2}\widetilde{\lambda}\widetilde{b^2} - \frac{1}{2}\log 2\pi + \frac{1}{2}\log \widetilde{\lambda} \\ \mathcal{L}_3 &= \sum_{f=1}^{D}\left((\alpha_\psi - 1)\widetilde{\log \psi_f} - \frac{\widetilde{\psi_f}}{\beta_\psi} - \log \Gamma(\alpha_\psi) - \alpha_\psi \log \beta_\psi\right) \\ \mathcal{L}_4 &= -\frac{1}{2}\operatorname{tr}(\operatorname{diag}(\widetilde{\boldsymbol{\psi}})\widetilde{\boldsymbol{w}\boldsymbol{w}^\top}) - \frac{1}{2}D\log 2\pi + \frac{1}{2}\log|\operatorname{diag}(\widetilde{\boldsymbol{\psi}})| \\ \mathcal{L}_5 &= \sum_{i=1}^{N}\left(-\frac{1}{2}\widetilde{t_i^2} + \widetilde{t_i}(\widetilde{\boldsymbol{w}^\top}\boldsymbol{x}_i + \widetilde{b}) - \frac{1}{2}\left(\operatorname{tr}(\widetilde{\boldsymbol{w}\boldsymbol{w}^\top}\boldsymbol{x}_i\boldsymbol{x}_i^\top) + 2\boldsymbol{x}_i^\top\widetilde{\boldsymbol{w}}\widetilde{b} + \widetilde{b^2}\right) - \frac{1}{2}\log 2\pi\right) \\ \mathcal{L}_6 &= 0\end{aligned}$$

and the negative entropies of the approximate posteriors in the ensemble are given as

$$\begin{aligned}
\mathcal{L}_7 &= -\alpha(\lambda) - \log\beta(\lambda) - \log\Gamma(\alpha(\lambda)) - (1-\alpha(\lambda))\psi(\alpha(\lambda)) \\
\mathcal{L}_8 &= \sum_{f=1}^{D}(-\alpha(\psi_f) - \log\beta(\psi_f) - \log\Gamma(\alpha(\psi_f)) - (1-\alpha(\psi_f))\psi(\alpha(\psi_f))) \\
\mathcal{L}_9 &= -\frac{1}{2}(D+1)(\log 2\pi + 1) - \frac{1}{2}\log|\Sigma(b, \boldsymbol{w})| \\
\mathcal{L}_{10} &= \sum_{i=1}^{N}\left(-\frac{1}{2}(\log 2\pi + \Sigma(t_i)) - \log \mathcal{Z}_i\right)
\end{aligned}$$

where $\Gamma(\cdot)$ denotes the gamma function and $\psi(\cdot)$ denotes the digamma function.

The posterior expectations needed in order to update approximate posterior distributions and to calculate the lower bound can be given as

$$\begin{aligned}
\widetilde{\lambda} &= \alpha(\lambda)\beta(\lambda) \\
\widetilde{\log\lambda} &= \psi(\alpha(\lambda)) + \log\beta(\lambda) \\
\widetilde{\psi_f} &= \alpha(\psi_f)\beta(\psi_f) && \forall f \\
\widetilde{\log\psi_f} &= \psi(\alpha(\psi_f)) + \log\beta(\psi_f) && \forall f \\
\widetilde{\boldsymbol{\psi}} &= \begin{bmatrix}\alpha(\psi_1)\beta(\psi_1) & \alpha(\psi_2)\beta(\psi_2) & \cdots & \alpha(\psi_D)\beta(\psi_D)\end{bmatrix}^\top \\
\widetilde{b} &= \mu(b) \\
\widetilde{b^2} &= \mu(b)^2 + \Sigma(b) \\
\widetilde{w_f^2} &= \mu(w_f)^2 + \Sigma(w_f) \\
\widetilde{\boldsymbol{w}} &= \begin{bmatrix}\mu(w_1) & \mu(w_2) & \cdots & \mu(w_D)\end{bmatrix}^\top \\
\widetilde{\boldsymbol{w}\boldsymbol{w}^\top} &= \mu(\boldsymbol{w})\mu(\boldsymbol{w})^\top + \Sigma(\boldsymbol{w}).
\end{aligned}$$

The only nonstandard distribution we need to operate on is the truncated normal distribution used for the auxiliary variables. From the model definition, the truncation

points for each auxiliary variable are defined as

$$(l_i, u_i) = \begin{cases} (-\infty, 0) & \text{if } y_i = -1 \\ (0, +\infty) & \text{otherwise} \end{cases} \qquad \forall i$$

where l_i and u_i denote the lower and upper truncation points, respectively. The normalization coefficient, the expectation, and the variance of the auxiliary variables can be calculated as

$$\begin{aligned} \mathcal{Z}_i &= \Phi(\beta_i) - \Phi(\alpha_i) && \forall i \\ \widetilde{t_i} &= \widetilde{\boldsymbol{w}^\top} \boldsymbol{x}_i + \widetilde{b} + \frac{\phi(\alpha_i) - \phi(\beta_i)}{\mathcal{Z}_i} && \forall i \\ \widetilde{t_i^2} - (\widetilde{t_i})^2 &= 1 + \frac{\alpha_i \phi(\alpha_i) - \beta_i \phi(\beta_i)}{\mathcal{Z}_i} - \frac{(\phi(\alpha_i) - \phi(\beta_i))^2}{\mathcal{Z}_i^2} && \forall i \end{aligned}$$

where $\phi(\cdot)$ is the standardized normal probability density function and $\{\alpha_i, \beta_i\}$ are defined as

$$\begin{aligned} \alpha_i &= l_i - \widetilde{\boldsymbol{w}^\top} \boldsymbol{x}_i - \widetilde{b} && \forall i \\ \beta_i &= u_i - \widetilde{\boldsymbol{w}^\top} \boldsymbol{x}_i - \widetilde{b} && \forall i. \end{aligned}$$

3.2. Prediction

The predictive distribution of the auxiliary variable $t_\star$ can also be found by replacing $p(b, \boldsymbol{w}|\mathbf{X}, \boldsymbol{y})$ with its approximate posterior distribution $q(b, \boldsymbol{w})$:

$$p(t_\star|\boldsymbol{x}_\star, \mathbf{X}, \boldsymbol{y}) = \mathcal{N}\left(t_\star; \mu(b, \boldsymbol{w})^\top \begin{bmatrix} 1 \\ \boldsymbol{x}_\star \end{bmatrix}, 1 + \begin{bmatrix} 1 & \boldsymbol{x}_\star \end{bmatrix} \Sigma(b, \boldsymbol{w}) \begin{bmatrix} 1 \\ \boldsymbol{x}_\star \end{bmatrix} \right)$$

and the predictive distribution of the class label $y_\star$ can be formulated using the auxiliary variable distribution:

$$p(y_\star = +1|\boldsymbol{x}_\star, \mathbf{X}, \boldsymbol{y}) = \Phi\left(\frac{\mu(t_\star)}{\Sigma(t_\star)}\right)$$

where $\Phi(\cdot)$ is the standardized normal cumulative distribution function.

3.3. Group-Wise Feature Selection

We can give the probit classifier feature selection capability using suitable hyper-parameters for the prior distributions on the weight vector precisions. Hyper-parameter values are usually selected as $(\alpha_\psi, \beta_\psi) = (1, 1)$. In this case, there will be no explicit feature selection mechanism and most of the features will be used in the final decision function with nonzero weights. In order to perform feature selection, we can use sparsity-inducing hyper-parameters such as $(\alpha_\psi, \beta_\psi) = (10^{-10}, 10^{+10})$. In this case, some of the weights are forced to become zero and the corresponding features will be eliminated leading to feature selection.

Suppose that the features are categorized into P distinct groups and $g(f)$ gives the group of feature f. The precision priors of the probabilistic model become

$$\psi_m \sim \mathcal{G}(\psi_m; \alpha_\psi, \beta_\psi) \qquad \forall m$$

and the corresponding factor in the factorable ensemble approximation is given as

$$q(\boldsymbol{\psi}) = \prod_{m=1}^{P} \mathcal{G}(\psi_m; \alpha(\psi_m), \beta(\psi_m)).$$

The approximate posterior distributions of the priors on the weight vector can again be found in terms of gamma distributions:

$$q(\boldsymbol{\psi}) = \prod_{m=1}^{P} \mathcal{G}\left(\psi_m; \alpha_\psi + \frac{1}{2}\sum_{f=1}^{D}\delta(g(f)=m), \left(\frac{1}{\beta_\psi} + \frac{1}{2}\sum_{f=1}^{D}\delta(g(f)=m)\widetilde{w_f^2}\right)^{-1}\right)$$

and the approximate posterior distribution of the classification parameters is a product of multivariate normal distributions:

$$q(b, \boldsymbol{w}) = \mathcal{N}\left(\begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix}; \Sigma(b, \boldsymbol{w}) \begin{bmatrix} \mathbf{1}^\top \widetilde{\boldsymbol{t}} \\ \mathbf{X}\widetilde{\boldsymbol{t}} \end{bmatrix}, \begin{bmatrix} \widetilde{\lambda} + N & \mathbf{1}^\top \mathbf{X}^\top \\ \mathbf{X}\mathbf{1} & \operatorname{diag}\left(\begin{bmatrix} \widetilde{\psi_{g(1)}} & \dots & \widetilde{\psi_{g(D)}} \end{bmatrix}^\top\right) + \mathbf{X}\mathbf{X}^\top \end{bmatrix}^{-1}\right).$$

Our Matlab implementation for the probit classifier with group-wise feature selection capability is available in Appendix A. Note that if each feature is considered as a separate group, our implementation reduces to the standard probit classifier.

4. GROUP-WISE FEATURE SELECTION USING MULTIPLE KERNEL LEARNING

SVM is a discriminative classifier proposed for binary classification problems and is based on the theory of structural risk minimization (Vapnik, 1998). Given a sample of N independent and identically distributed training instances $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ where $\boldsymbol{x}_i$ is the D-dimensional input vector and $y_i \in \{-1, +1\}$ is its class label, SVM basically finds the linear discriminant with the maximum margin in the feature space induced by the mapping function $\Phi\colon \mathbb{R}^D \to \mathbb{R}^S$. The resulting discriminant function is

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \Phi(\boldsymbol{x}) \rangle + b.$$

The classifier can be trained by solving the following quadratic optimization problem:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^N \xi_i \\ \text{with respect to} \quad & \boldsymbol{w} \in \mathbb{R}^S, \ \ \boldsymbol{\xi} \in \mathbb{R}_+^N, \ \ b \in \mathbb{R} \\ \text{subject to} \quad & y_i(\langle \boldsymbol{w}, \Phi(\boldsymbol{x}_i) \rangle + b) \geq 1 - \xi_i \qquad \forall i \end{aligned}$$

where $\boldsymbol{w}$ is the vector of weight coefficients, C is a predefined positive trade-off parameter between model simplicity and classification error, $\boldsymbol{\xi}$ is the vector of slack variables, and b is the bias term. Instead of solving this optimization problem directly, the Lagrangian dual function enables us to obtain the following dual formulation:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \underbrace{\langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j) \rangle}_{k(\boldsymbol{x}_i, \boldsymbol{x}_j)} \\ \text{with respect to} \quad & \boldsymbol{\alpha} \in \mathbb{R}_+^N \\ \text{subject to} \quad & \sum_{i=1}^N \alpha_i y_i = 0 \\ & C \geq \alpha_i \geq 0 \qquad \forall i \end{aligned}$$

where $k\colon \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$ is named the *kernel function* and $\boldsymbol{\alpha}$ is the vector of dual variables corresponding to each separation constraint. Solving this, we get $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \Phi(\boldsymbol{x}_i)$ and the discriminant function can be rewritten as

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b.$$

There are several kernel functions successfully used in the literature, such as the linear kernel (k_{LIN}), the polynomial kernel (k_{POL}), and the Gaussian kernel (k_{GAU}):

$$\begin{aligned}
k_{LIN}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \\
k_{POL}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= (\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + 1)^q, \qquad q \in \mathbb{N} \\
k_{GAU}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \exp\left(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2 / s^2\right), \qquad s \in \mathbb{R}_{++}.
\end{aligned}$$

There are also kernel functions proposed for particular applications, such as natural language processing (Lodhi *et al.*, 2002) and bioinformatics (Schölkopf *et al.*, 2004).

Selecting the kernel function $k(\cdot, \cdot)$ and its parameters (e.g., q or s) is an important issue in training. Generally, a cross-validation procedure is used to choose the best performing kernel function among a set of kernel functions on a separate validation set different from the training set. In recent years, MKL methods have been proposed, where we use multiple kernels instead of selecting one specific kernel function and its corresponding parameters:

$$k_\eta(\boldsymbol{x}_i, \boldsymbol{x}_j) = f_\eta(\{k_m(\boldsymbol{x}_i^m, \boldsymbol{x}_j^m)\}_{m=1}^P)$$

where the combination function, $f_\eta\colon \mathbb{R}^P \rightarrow \mathbb{R}$, can be a linear or a nonlinear function. Kernel functions, $\{k_m\colon \mathbb{R}^{D_m} \times \mathbb{R}^{D_m} \rightarrow \mathbb{R}\}_{m=1}^P$, take P feature representations (not necessarily different) of data instances: $\boldsymbol{x}_i = \{\boldsymbol{x}_i^m\}_{m=1}^P$ where $\boldsymbol{x}_i^m \in \mathbb{R}^{D_m}$, and D_m is the dimensionality of the corresponding feature representation. $\boldsymbol{\eta}$ parameterizes the

combination function and the more common implementation is:

$$k_\eta(\boldsymbol{x}_i, \boldsymbol{x}_j) = f_\eta(\{k_m(\boldsymbol{x}_i^m, \boldsymbol{x}_j^m)\}_{m=1}^P | \boldsymbol{\eta})$$

where the parameters are used to combine a set of predefined kernels (i.e., we know the kernel functions and corresponding kernel parameters before training). It is also possible to view this as

$$k_\eta(\boldsymbol{x}_i, \boldsymbol{x}_j) = f_\eta(\{k_m(\boldsymbol{x}_i^m, \boldsymbol{x}_j^m | \boldsymbol{\eta})\}_{m=1}^P)$$

where the parameters integrated into the kernel functions are optimized during training. Most of the existing MKL algorithms fall into the first category and try to combine predefined kernels in an optimal way.

The reasoning is similar to combining different classifiers: Instead of choosing a single kernel function, it is better to have a set and let an algorithm do the picking or combination. There can be two uses of MKL: (i) Different kernels correspond to different notions of similarity and instead of trying to find which works best, a learning method does the picking for us, or may use a combination of them. Using a specific kernel may be a source of bias, and in allowing a learner to choose among a set of kernels, a better solution can be found. (ii) Different kernels may be using inputs coming from different representations possibly from different sources or modalities. Since these are different representations, they have different measures of similarity corresponding to different kernels. In such a case, combining kernels is one possible way to combine multiple information sources. Noble (2004) calls this method of combining kernels *intermediate combination* and contrasts this with *early combination* (where features from different sources are concatenated and fed to a single learner) and *late combination* (where different features are fed to different classifiers whose decisions are then combined by a fixed or trained combiner).

There are many MKL algorithms proposed in the literature (see Gönen and Alpaydın (2011) for a recent survey). Linear combination methods are the most popular and have two basic categories: unweighted sum (i.e., using sum or mean of the kernels as the combined kernel) and weighted sum. In the weighted sum case, we can linearly parameterize the combination function:

$$k_\eta(\boldsymbol{x}_i, \boldsymbol{x}_j) = f_\eta(\{k_m(\boldsymbol{x}_i^m, \boldsymbol{x}_j^m)\}_{m=1}^P | \boldsymbol{\eta}) = \sum_{m=1}^P \eta_m k_m(\boldsymbol{x}_i^m, \boldsymbol{x}_j^m) \tag{4.1}$$

where $\boldsymbol{\eta}$ denotes the kernel weights. Different versions of this approach differ in the way they put restrictions on $\boldsymbol{\eta}$: the linear sum (i.e., $\boldsymbol{\eta} \in \mathbb{R}^P$), the conic sum (i.e., $\boldsymbol{\eta} \in \mathbb{R}_+^P$), or the convex sum (i.e., $\boldsymbol{\eta} \in \mathbb{R}_+^P$ and $\sum_{m=1}^P \eta_m = 1$).

4.1. Training Details

We pick a specific MKL formulation, which enables us to tune kernel-level sparsity and to perform feature selection by choosing a model parameter suitably. Xu *et al.* (2010) and Kloft *et al.* (2011) propose an efficient optimization method for arbitrary ℓ_p-norms with $p \geq 1$. Although they approach the problem from different perspectives, they find the same closed-form solution for updating the kernel weights at each iteration. In order to derive the update equation, Xu *et al.* (2010) use the equivalence between group Lasso and MKL, as shown by Bach (2008), whereas Kloft *et al.* (2011) use a block coordinate-descent method. Both studies formulate an alternating optimization method that solves an SVM at each iteration and update the kernel weights as follows:

$$\eta_m = \frac{\|\boldsymbol{w}_m\|_2^{\frac{2}{p+1}}}{\left(\sum_{h=1}^P \|\boldsymbol{w}_h\|_2^{\frac{2p}{p+1}}\right)^{1/p}} \qquad \forall m \tag{4.2}$$

where $\|\boldsymbol{w}_m\|_2^2 = \eta_m^2 \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j k_m(\boldsymbol{x}_i^m, \boldsymbol{x}_j^m)$ from the duality conditions. After convergence, we obtain the optimal support vector coefficients and kernel weights. The complete training algorithm is listed in Figure 4.1.

Require: $\{k_m(\cdot,\cdot)\}_{m=1}^{P}$, $\{\mathbf{X}^m\}_{m=1}^{P}$, $\boldsymbol{y}$, and p

1: Initialize $\boldsymbol{\eta}$ randomly

2: **repeat**

3: Solve a single-kernel SVM using $k_\eta(\cdot,\cdot)$ in Equation 4.1

4: Update $\boldsymbol{\eta}$ using Equation 4.2

5: **until** convergence

6: **return** $\boldsymbol{\alpha}, b, \boldsymbol{\eta}$

Figure 4.1. Training algorithm of the ℓ_p-norm MKL.

4.2. Prediction

Using the optimal support vector coefficients and kernel weights, the discriminant function for a test data point can be found as

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i y_i \left(\sum_{m=1}^{P} \eta_m k_m(\boldsymbol{x}_i^m, \boldsymbol{x}^m) \right) + b.$$

In addition to sample-level sparsity (i.e., having zero support vector coefficients for some of the training data points) like in SVMs, MKL can also have kernel-level sparsity by having zero weights for some of the input kernels. We do not need to evaluate a kernel function and to collect data for that particular kernel if its weight is zero.

4.3. Group-Wise Feature Selection

We can give MKL feature selection capability by calculating separate kernels for each feature and using a sparsity-inducing norm on the kernel weights. When $p = 2$, there will be no explicit kernel selection mechanism and most of the kernels will be used in the combined kernel with nonzero weights. In order to perform kernel selection, we can use a sparsity-inducing norm such as $p = 1$. In this case, some of the kernel weights are forced to become zero and the corresponding kernel will be eliminated leading to feature selection.

We can also perform group-wise feature selection by defining separate kernels for each feature group. Like in the previous section, suppose that the features are categorized into P distinct groups and $\mathcal{I}_m$ gives the feature indices of group m. The input representation used for kernel m is defined as $\boldsymbol{x}^m = \boldsymbol{x}[\mathcal{I}_m]$ where $[\cdot]$ indexes the elements of a vector.

We use Matlab implementation of the ℓ_p-norm MKL provided by Gönen and Alpaydın (2011), which is available at `www.cmpe.boun.edu.tr/~gonen/mkl`.

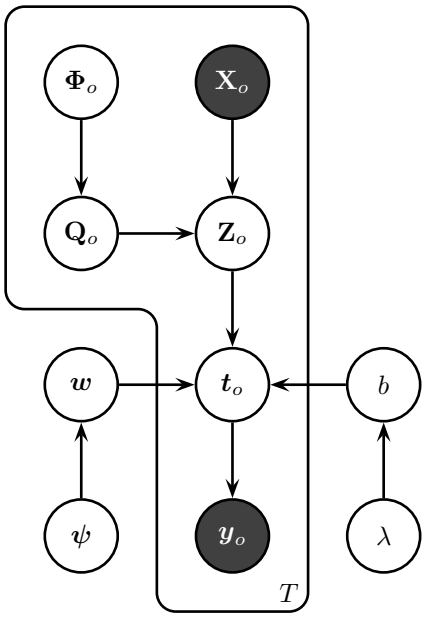
5. TRANSFER LEARNING USING PROBIT CLASSIFIER

Transfer learning considers applications with related tasks of different data sets and aims to exploit these different data sets to obtain better learners than the learners that can be found from each data set separately. It is especially useful when we do not have enough training instances in each data set to build a reliable learner. There are two common approaches for transfer learning: (i) training separate learners for each data set in a coupled manner by forcing them to have similar model parameters, and (ii) projecting the data points from each data set into a unified subspace and training a common learner in this subspace. We focus on the second approach in this thesis.

We propose to combine linear dimensionality reduction and linear classification in a joint probabilistic model in order to obtain predictive subspaces for transfer learning problems. The main idea is to map the training instances of different data sets to a unified subspace using linear projection matrices and to estimate the class labels in this projected subspace with the probit classifier. We should consider the predictive performance of the unified subspace while learning the projection matrices to transfer information between data sets.

Figure 5.1 illustrates the transfer learning probit classifier with a graphical model and its distributional assumptions. First, we project the data matrices $\{\mathbf{X}_o\}_{o=1}^T$ into a unified low-dimensional space using the projection matrices $\{\mathbf{Q}_o\}_{o=1}^T$. The low-dimensional representations of training data points $\{\mathbf{Z}_o\}_{o=1}^T$ are used to calculate the classification scores using the shared set of classification parameters $\{b, \boldsymbol{w}\}$. Finally, the given label vectors $\{\boldsymbol{y}_o\}_{o=1}^T$ are generated from the classification score vectors $\{\boldsymbol{t}_o\}_{o=1}^T$.

The notation we use for the transfer learning probit classifier is as follows: T denotes the number of data sets (i.e., tasks). N_o and D_o are the number of training instances and the dimensionality of the input space for the corresponding data set. R gives the dimensionality of the unified projected subspace. The $D_o \times N_o$ data matrix of each data set is denoted by $\mathbf{X}_o$, where the $D_o \times 1$-dimensional columns of $\mathbf{X}_o$ by $\boldsymbol{x}_{o,i}$.



$$
\begin{aligned}
\phi^f_{o,s} &\sim \mathcal{G}(\phi^f_{o,s}; \alpha_\phi, \beta_\phi) && \forall(o, f, s) \\
q^f_{o,s}|\phi^f_{o,s} &\sim \mathcal{N}(q^f_{o,s}; 0, \phi_s^{-1}) && \forall(o, f, s) \\
z^s_{o,i}|\boldsymbol{q}_{o,s}, \boldsymbol{x}_{o,i} &\sim \mathcal{N}(z^s_{o,i}; \boldsymbol{q}^\top_{o,s}\boldsymbol{x}_{o,i}, 1) && \forall(o, s, i) \\
\lambda &\sim \mathcal{G}(\lambda; \alpha_\lambda, \beta_\lambda) \\
b|\lambda &\sim \mathcal{N}(b; 0, \lambda^{-1}) \\
\psi_s &\sim \mathcal{G}(\psi_s; \alpha_\psi, \beta_\psi) && \forall s \\
w_s|\psi_s &\sim \mathcal{N}(w_s; 0, \psi_s^{-1}) && \forall s \\
t_{o,i}|b, \boldsymbol{w}, \boldsymbol{z}_{o,i} &\sim \mathcal{N}(t_{o,i}; \boldsymbol{w}^\top \boldsymbol{z}_{o,i} + b, 1) && \forall(o, i) \\
y_{o,i}|t_{o,i} &\sim \delta(t_{o,i} y_{o,i} > 0) && \forall(o, i)
\end{aligned}
$$

Figure 5.1. Graphical model and distributional assumptions of the transfer learning probit classifier.

The $D_o \times R$ matrix of projection variables $q^f_{o,s}$ is denoted by $\mathbf{Q}_o$, where the $D_o \times 1$-dimensional columns of $\mathbf{Q}_o$ by $\boldsymbol{q}_{o,s}$. The $D_o \times R$ matrix of priors $\phi^f_{o,s}$ is denoted by $\boldsymbol{\Phi}_o$, where the $D_o \times 1$-dimensional columns of $\boldsymbol{\Phi}_o$ by $\boldsymbol{\phi}_{o,s}$. The $R \times N_o$ matrix of projected variables $z^s_{o,i}$ is represented as $\mathbf{Z}_o$, where the $R \times 1$-dimensional columns of $\mathbf{Z}_o$ as $\boldsymbol{z}_{o,i}$ and the corresponding $N_o \times 1$-dimensional rows as $\boldsymbol{z}^s_o$. The $R \times 1$ vector of weight parameters w_s is denoted by $\boldsymbol{w}$. The $R \times 1$ vector of priors ψ_s is denoted by $\boldsymbol{\psi}$. The bias parameter is denoted by b and its prior is denoted by λ. The $N_o \times 1$ vector of auxiliary variables $t_{o,i}$ is represented as $\boldsymbol{t}_o$. The $N_o \times 1$ vector of associated target values is represented as $\boldsymbol{y}_o$, where each element $y_{o,i} \in \{-1, +1\}$. As short-hand notations, all priors in the model are denoted by $\boldsymbol{\Xi} = \{\lambda, \{\boldsymbol{\Phi}_o\}_{o=1}^T, \boldsymbol{\psi}\}$, where the remaining variables by $\boldsymbol{\Theta} = \{b, \{\mathbf{Q}_o\}_{o=1}^T, \{\boldsymbol{t}_o\}_{o=1}^T, \boldsymbol{w}, \{\mathbf{Z}_o\}_{o=1}^T\}$ and the hyper-parameters by $\boldsymbol{\zeta} = \{\alpha_\lambda, \beta_\lambda, \alpha_\phi, \beta_\phi, \alpha_\psi, \beta_\psi\}$. Dependence on $\boldsymbol{\zeta}$ is again omitted for clarity throughout the rest.

Exact inference for this probabilistic model is also intractable. We again formulate a deterministic variational approximation procedure for efficient inference instead of using a Gibbs sampling approach as in the probit classifier case.

We can write the factorable ensemble approximation of the required posterior as

$$p(\boldsymbol{\Theta}, \boldsymbol{\Xi} | \{\mathbf{X}_o\}_{o=1}^T, \{\boldsymbol{y}_o\}_{o=1}^T) \approx q(\boldsymbol{\Theta}, \boldsymbol{\Xi}) = q(\{\boldsymbol{\Phi}_o\}_{o=1}^T) q(\{\mathbf{Q}_o\}_{o=1}^T) q(\{\mathbf{Z}_o\}_{o=1}^T) q(\lambda) q(\boldsymbol{\psi}) q(b, \boldsymbol{w}) q(\{\boldsymbol{t}_o\}_{o=1}^T)$$

and define each factor in the ensemble just like its full conditional distribution:

$$\begin{aligned}
q(\{\boldsymbol{\Phi}_o\}_{o=1}^T) &= \prod_{o=1}^{T} \prod_{f=1}^{D_o} \prod_{s=1}^{R} \mathcal{G}(\phi_{o,s}^f; \alpha(\phi_{o,s}^f), \beta(\phi_{o,s}^f)) \\
q(\{\mathbf{Q}_o\}_{o=1}^T) &= \prod_{o=1}^{T} \prod_{s=1}^{R} \mathcal{N}(\boldsymbol{q}_{o,s}; \mu(\boldsymbol{q}_{o,s}), \Sigma(\boldsymbol{q}_{o,s})) \\
q(\{\mathbf{Z}_o\}_{o=1}^T) &= \prod_{o=1}^{T} \prod_{i=1}^{N_o} \mathcal{N}(\boldsymbol{z}_{o,i}; \mu(\boldsymbol{z}_{o,i}), \Sigma(\boldsymbol{z}_{o,i})) \\
q(\lambda) &= \mathcal{G}(\lambda; \alpha(\lambda), \beta(\lambda)) \\
q(\boldsymbol{\psi}) &= \prod_{s=1}^{R} \mathcal{G}(\psi_s; \alpha(\psi_s), \beta(\psi_s)) \\
q(b, \boldsymbol{w}) &= \mathcal{N}\left(\begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix}; \mu(b, \boldsymbol{w}), \Sigma(b, \boldsymbol{w}) \right) \\
q(\{\boldsymbol{t}_o\}_{o=1}^T) &= \prod_{o=1}^{T} \prod_{i=1}^{N} \mathcal{TN}(t_{o,i}; \mu(t_{o,i}), \Sigma(t_{o,i}), \rho(t_{o,i})).
\end{aligned}$$

We can bound the marginal likelihood using Jensen's inequality:

$$\log p(\{\boldsymbol{y}_o\}_{o=1}^T | \{\mathbf{X}_o\}_{o=1}^T) \geq \mathrm{E}_{q(\boldsymbol{\Theta}, \boldsymbol{\Xi})}[\log p(\{\boldsymbol{y}_o\}_{o=1}^T, \boldsymbol{\Theta}, \boldsymbol{\Xi} | \{\mathbf{X}_o\}_{o=1}^T)] - \mathrm{E}_{q(\boldsymbol{\Theta}, \boldsymbol{\Xi})}[\log q(\boldsymbol{\Theta}, \boldsymbol{\Xi})] \quad (5.1)$$

and optimize this bound by maximizing with respect to each factor separately until convergence. The approximate posterior distribution of a specific factor $\boldsymbol{\tau}$ is found as

$$q(\boldsymbol{\tau}) \propto \exp\left(\mathrm{E}_{q(\{\boldsymbol{\Theta},\boldsymbol{\Xi}\}\setminus\boldsymbol{\tau})}[\log p(\{\boldsymbol{y}_o\}_{o=1}^T, \boldsymbol{\Theta}, \boldsymbol{\Xi}|\{\mathbf{X}_o\}_{o=1}^T)]\right).$$

Similar to the probit classifier, the resulting approximate posterior distribution of each factor follows the same distribution as the corresponding factor also for the transfer learning probit classifier.

5.1. Inference Details

The dimensionality reduction part has two sets of parameters: the projection matrices that have normally distributed entries and the prior matrices that determine the precisions for these projection matrices. The approximate posterior distribution of the priors can be formulated as a product of gamma distributions:

$$q(\{\boldsymbol{\Phi}_o\}_{o=1}^T) = \prod_{o=1}^{T}\prod_{f=1}^{D_o}\prod_{s=1}^{R}\mathcal{G}\left(\phi_{o,s}^f; \alpha_\phi + \frac{1}{2}, \left(\frac{1}{\beta_\phi} + \frac{\widetilde{(q_{o,s}^f)^2}}{2}\right)^{-1}\right). \tag{5.2}$$

The approximate posterior distribution of the projection matrices is a product of multivariate normal distributions:

$$q(\{\mathbf{Q}_o\}_{o=1}^T) = \prod_{o=1}^{T}\prod_{s=1}^{R}\mathcal{N}(\boldsymbol{q}_{o,s}; \Sigma(\boldsymbol{q}_{o,s})\mathbf{X}_o\widetilde{\boldsymbol{z}_o^s}, (\mathrm{diag}(\widetilde{\boldsymbol{\phi}_{\boldsymbol{o},\boldsymbol{s}}}) + \mathbf{X}_o\mathbf{X}_o^\top)^{-1}). \tag{5.3}$$

The approximate posterior distribution of the projected instances can be found as a product of multivariate normal distributions:

$$q(\{\mathbf{Z}_o\}_{o=1}^T) = \prod_{o=1}^{T}\prod_{i=1}^{N_o}\mathcal{N}(\boldsymbol{z}_{o,i}; \Sigma(\boldsymbol{z}_{o,i})(\widetilde{\mathbf{Q}_o^\top}\boldsymbol{x}_{o,i} + \widetilde{t_{o,i}}\widetilde{\boldsymbol{w}} - \widetilde{b}\widetilde{\boldsymbol{w}}), (\mathbf{I} + \widetilde{\boldsymbol{w}\boldsymbol{w}^\top})^{-1}). \tag{5.4}$$

The binary classification part has two sets of parameters: the bias vector and the weight matrix that have normally distributed entries, and the corresponding priors are from gamma distribution. The approximate posterior distributions of the priors on the bias and the weight vector can be found in terms of gamma distributions:

$$q(\lambda) = \mathcal{G}\left(\lambda; \alpha_\lambda + \frac{1}{2}, \left(\frac{1}{\beta_\lambda} + \frac{\widetilde{b^2}}{2}\right)^{-1}\right) \tag{5.5}$$

$$q(\boldsymbol{\psi}) = \prod_{s=1}^{R} \mathcal{G}\left(\psi_s; \alpha_\psi + \frac{1}{2}, \left(\frac{1}{\beta_\psi} + \frac{\widetilde{w_s^2}}{2}\right)^{-1}\right). \tag{5.6}$$

The approximate posterior distribution of the classification parameters is a product of multivariate normal distributions:

$$q(b, \boldsymbol{w}) = \mathcal{N}\left(\begin{bmatrix} b \\ \boldsymbol{w} \end{bmatrix}; \Sigma(b, \boldsymbol{w}) \begin{bmatrix} \sum_{o=1}^{T} \mathbf{1}^\top \widetilde{\boldsymbol{t}_o} \\ \sum_{o=1}^{T} \widetilde{\mathbf{Z}_o} \widetilde{\boldsymbol{t}_o} \end{bmatrix}, \begin{bmatrix} \widetilde{\lambda} + \sum_{o=1}^{T} N_o & \sum_{o=1}^{T} \mathbf{1}^\top \widetilde{\mathbf{Z}_o^\top} \\ \sum_{o=1}^{T} \widetilde{\mathbf{Z}_o} \mathbf{1} & \operatorname{diag}(\widetilde{\boldsymbol{\psi}}) + \sum_{o=1}^{T} \widetilde{\mathbf{Z}_o \mathbf{Z}_o^\top} \end{bmatrix}^{-1}\right) \tag{5.7}$$

where we couple different data sets using the same bias parameter and weight vector for classification. The projection matrix for each data set tries to embed corresponding data points accordingly. The approximate posterior distribution of the auxiliary variables is a product of truncated normal distributions:

$$q(\{\boldsymbol{t}_o\}_{o=1}^{T}) = \prod_{o=1}^{T} \prod_{i=1}^{N} \mathcal{TN}(t_{o,i}; \widetilde{\boldsymbol{w}^\top} \widetilde{\boldsymbol{z}_{o,i}} + \widetilde{b}, 1, t_{o,i} y_{o,i} > 0) \tag{5.8}$$

where we need to find the posterior expectations in order to update the approximate posterior distributions of the projected instances and the classification parameters.

The complete inference algorithm is listed in Figure 5.2. Similar to the probit classifier, the inference mechanism sequentially updates the approximate posterior distributions of the model parameters and the latent variables until convergence, which

can again be checked by monitoring the lower bound in Equation 5.1. Our Matlab implementation for the transfer learning probit classifier is also available in Appendix A.

Require: $\{\mathbf{X}_o\}_{o=1}^T$, $\{\boldsymbol{y}_o\}_{o=1}^T$, α_λ, β_λ, α_ϕ, β_ϕ, α_ψ, β_ψ, and R

1: Initialize $q(\{\mathbf{Q}_o\}_{o=1}^T)$, $q(\{\mathbf{Z}_o\}_{o=1}^T)$, $q(b, \boldsymbol{w})$, and $q(\{\boldsymbol{t}_o\}_{o=1}^T)$ randomly
2: **repeat**
3: Update $q(\{\boldsymbol{\Phi}_o\}_{o=1}^T)$ using Equation 5.2
4: Update $q(\{\mathbf{Q}_o\}_{o=1}^T)$ using Equation 5.3
5: Update $q(\{\mathbf{Z}_o\}_{o=1}^T)$ using Equation 5.4
6: Update $q(\lambda)$ using Equation 5.5
7: Update $q(\boldsymbol{\psi})$ using Equation 5.6
8: Update $q(b, \boldsymbol{w})$ using Equation 5.7
9: Update $q(\{\boldsymbol{t}_o\}_{o=1}^T)$ using Equation 5.8
10: **until** convergence
11: **return** $q(\{\mathbf{Q}_o\}_{o=1}^T)$ and $q(b, \boldsymbol{w})$

Figure 5.2. Inference algorithm of the transfer learning probit classifier.

The variational lower bound of transfer learning probit classification model can be written as

$$\mathcal{L} = \mathrm{E}_{q(\boldsymbol{\Theta},\boldsymbol{\Xi})}[\log p(\{\boldsymbol{y}_o\}_{o=1}^T, \boldsymbol{\Theta}, \boldsymbol{\Xi}|\{\mathbf{X}_o\}_{o=1}^T)] - \mathrm{E}_{q(\boldsymbol{\Theta},\boldsymbol{\Xi})}[\log q(\boldsymbol{\Theta}, \boldsymbol{\Xi})]$$

where the joint likelihood is defined as

$$\begin{aligned} p(\{\boldsymbol{y}_o\}_{o=1}^T, \boldsymbol{\Theta}, \boldsymbol{\Xi}|\{\mathbf{X}_o\}_{o=1}^T) = \\ p(\{\boldsymbol{\Phi}_o\}_{o=1}^T) p(\{\mathbf{Q}_o\}_{o=1}^T|\ \{\boldsymbol{\Phi}_o\}_{o=1}^T) p(\{\mathbf{Z}_o\}_{o=1}^T|\ \{\mathbf{Q}_o\}_{o=1}^T, \{\mathbf{X}_o\}_{o=1}^T) \\ p(\lambda) p(b|\lambda) p(\boldsymbol{\psi}) p(\boldsymbol{w}|\boldsymbol{\psi}) p(\{\boldsymbol{t}_o\}_{o=1}^T|b, \boldsymbol{w}, \{\mathbf{Z}_o\}_{o=1}^T) p(\{\boldsymbol{y}_o\}_{o=1}^T|\{\boldsymbol{t}_o\}_{o=1}^T). \end{aligned}$$

Using these definitions, the variational lower bound becomes

$$
\begin{aligned}
\mathcal{L} = & \underbrace{\mathrm{E}_{q(\{\boldsymbol{\Phi}_o\}_{o=1}^T)}[\log p(\{\boldsymbol{\Phi}_o\}_{o=1}^T)]}_{\mathcal{L}_1} + \underbrace{\mathrm{E}_{q(\{\boldsymbol{\Phi}_o\}_{o=1}^T)q(\{\mathbf{Q}_o\}_{o=1}^T)}[\log p(\{\mathbf{Q}_o\}_{o=1}^T|\{\boldsymbol{\Phi}_o\}_{o=1}^T)]}_{\mathcal{L}_2} \\
& + \underbrace{\mathrm{E}_{q(\{\mathbf{Q}_o\}_{o=1}^T)q(\{\mathbf{Z}_o\}_{o=1}^T)}[\log p(\{\mathbf{Z}_o\}_{o=1}^T|\ \{\mathbf{Q}_o\}_{o=1}^T, \{\mathbf{X}_o\}_{o=1}^T)]}_{\mathcal{L}_3} + \underbrace{\mathrm{E}_{q(\lambda)}[\log p(\lambda)]}_{\mathcal{L}_4} \\
& + \underbrace{\mathrm{E}_{q(\lambda)q(b,\boldsymbol{w})}[\log p(b|\lambda)]}_{\mathcal{L}_5} + \underbrace{\mathrm{E}_{q(\boldsymbol{\psi})}[\log p(\boldsymbol{\psi})]}_{\mathcal{L}_6} + \underbrace{\mathrm{E}_{q(\boldsymbol{\psi})q(b,\boldsymbol{w})}[\log p(\boldsymbol{w}|\boldsymbol{\psi})]}_{\mathcal{L}_7} \\
& + \underbrace{\mathrm{E}_{q(b,\boldsymbol{w})q(\{\boldsymbol{t}_o\}_{o=1}^T)q(\{\mathbf{Z}_o\}_{o=1}^T)}[\log p(\{\boldsymbol{t}_o\}_{o=1}^T|b,\boldsymbol{w},\{\mathbf{Z}_o\}_{o=1}^T)]}_{\mathcal{L}_8} \\
& + \underbrace{\mathrm{E}_{q(\{\boldsymbol{t}_o\}_{o=1}^T)}[\log p(q(\{\boldsymbol{y}_o\}_{o=1}^T)|\{\boldsymbol{t}_o\}_{o=1}^T)]}_{\mathcal{L}_9} - \underbrace{\mathrm{E}_{q(\{\boldsymbol{\Phi}_o\}_{o=1}^T)}[\log q(\{\boldsymbol{\Phi}_o\}_{o=1}^T)]}_{\mathcal{L}_{10}} \\
& - \underbrace{\mathrm{E}_{q(\{\mathbf{Q}_o\}_{o=1}^T)}[\log q(\{\mathbf{Q}_o\}_{o=1}^T)]}_{\mathcal{L}_{11}} - \underbrace{\mathrm{E}_{q(\{\mathbf{Z}_o\}_{o=1}^T)}[\log q(\{\mathbf{Z}_o\}_{o=1}^T)]}_{\mathcal{L}_{12}} - \underbrace{\mathrm{E}_{q(\lambda)}[\log q(\lambda)]}_{\mathcal{L}_{13}} \\
& - \underbrace{\mathrm{E}_{q(\boldsymbol{\psi})}[\log q(\boldsymbol{\psi})]}_{\mathcal{L}_{14}} - \underbrace{\mathrm{E}_{q(b,\boldsymbol{w})}[\log q(b,\boldsymbol{w})]}_{\mathcal{L}_{15}} - \underbrace{\mathrm{E}_{q(\{\boldsymbol{t}_o\}_{o=1}^T)}[\log q(\{\boldsymbol{t}_o\}_{o=1}^T)]}_{\mathcal{L}_{16}}
\end{aligned}
$$

where the exponential form expectations of the distributions in the joint likelihood can be calculated as

$$
\begin{aligned}
\mathcal{L}_1 &= \sum_{o=1}^{T}\sum_{f=1}^{D_o}\sum_{s=1}^{R}\left((\alpha_\phi - 1)\widetilde{\log \phi_{o,s}^f} - \frac{\widetilde{\phi_{o,s}^f}}{\beta_\phi} - \log\Gamma(\alpha_\phi) - \alpha_\phi \log \beta_\phi\right) \\
\mathcal{L}_2 &= \sum_{o=1}^{T}\sum_{s=1}^{R}\left(-\frac{1}{2}\operatorname{tr}(\operatorname{diag}(\widetilde{\boldsymbol{\phi}_{o,s}})\widetilde{\boldsymbol{q}_{o,s}\boldsymbol{q}_{o,s}^\top}) - \frac{1}{2}D_o\log 2\pi + \frac{1}{2}\log|\operatorname{diag}(\widetilde{\boldsymbol{\phi}_{o,s}})|\right) \\
\mathcal{L}_3 &= \sum_{o=1}^{T}\sum_{i=1}^{N_o}\left(-\frac{1}{2}\widetilde{\boldsymbol{z}_{o,i}^\top\boldsymbol{z}_{o,i}} + \boldsymbol{x}_{o,i}^\top\widetilde{\mathbf{Q}}_o\widetilde{\boldsymbol{z}_{o,i}} - \frac{1}{2}\operatorname{tr}(\widetilde{\mathbf{Q}_o\mathbf{Q}_o^\top}\boldsymbol{x}_{o,i}\boldsymbol{x}_{o,i}^\top) - \frac{1}{2}R\log 2\pi\right) \\
\mathcal{L}_4 &= (\alpha_\lambda - 1)\widetilde{\log\lambda} - \frac{\widetilde{\lambda}}{\beta_\lambda} - \log\Gamma(\alpha_\lambda) - \alpha_\lambda\log\beta_\lambda \\
\mathcal{L}_5 &= -\frac{1}{2}\widetilde{\lambda}\widetilde{b^2} - \frac{1}{2}\log 2\pi + \frac{1}{2}\log\widetilde{\lambda} \\
\mathcal{L}_6 &= \sum_{s=1}^{R}\left((\alpha_\psi - 1)\widetilde{\log\psi_s} - \frac{\widetilde{\psi_s}}{\beta_\psi} - \log\Gamma(\alpha_\psi) - \alpha_\psi\log\beta_\psi\right) \\
\mathcal{L}_7 &= -\frac{1}{2}\operatorname{tr}(\operatorname{diag}(\widetilde{\boldsymbol{\psi}})\widetilde{\boldsymbol{w}\boldsymbol{w}^\top}) - \frac{1}{2}R\log 2\pi + \frac{1}{2}\log|\operatorname{diag}(\widetilde{\boldsymbol{\psi}})|
\end{aligned}
$$

$$\mathcal{L}_8 = \sum_{o=1}^{T}\sum_{i=1}^{N_o}\left(-\frac{1}{2}\widetilde{t_{o,i}^2} + \widetilde{t_{o,i}}(\widetilde{\boldsymbol{w}^\top}\widetilde{\boldsymbol{z}_{o,i}} + \widetilde{b}) - \frac{1}{2}\left(\mathrm{tr}(\widetilde{\boldsymbol{w}\boldsymbol{w}^\top}\widetilde{\boldsymbol{z}_{o,i}\boldsymbol{z}_{o,i}^\top}) + 2\widetilde{\boldsymbol{z}_{o,i}^\top}\widetilde{\boldsymbol{w}}\widetilde{b} + \widetilde{b^2}\right) - \frac{1}{2}\log 2\pi\right)$$

$$\mathcal{L}_9 = 0$$

and the negative entropies of the approximate posteriors in the ensemble are given as

$$\mathcal{L}_{10} = \sum_{o=1}^{T}\sum_{f=1}^{D_o}\sum_{s=1}^{R}(-\alpha(\phi_{o,s}^f) - \log\beta(\phi_{o,s}^f) - \log\Gamma(\alpha(\phi_{o,s}^f)) - (1-\alpha(\phi_{o,s}^f))\psi(\alpha(\phi_{o,s}^f)))$$

$$\mathcal{L}_{11} = \sum_{o=1}^{T}\sum_{s=1}^{R}\left(-\frac{1}{2}(D_o+1)(\log 2\pi + 1) - \frac{1}{2}\log|\Sigma(q_{o,s})|\right)$$

$$\mathcal{L}_{12} = \sum_{o=1}^{T}\sum_{i=1}^{N_o}\left(-\frac{1}{2}(R+1)(\log 2\pi + 1) - \frac{1}{2}\log|\Sigma(z_{o,i})|\right)$$

$$\mathcal{L}_{13} = -\alpha(\lambda) - \log\beta(\lambda) - \log\Gamma(\alpha(\lambda)) - (1-\alpha(\lambda))\psi(\alpha(\lambda))$$

$$\mathcal{L}_{14} = \sum_{s=1}^{R}(-\alpha(\psi_s) - \log\beta(\psi_s) - \log\Gamma(\alpha(\psi_s)) - (1-\alpha(\psi_s))\psi(\alpha(\psi_s)))$$

$$\mathcal{L}_{15} = -\frac{1}{2}(R+1)(\log 2\pi + 1) - \frac{1}{2}\log|\Sigma(b, \boldsymbol{w})|$$

$$\mathcal{L}_{16} = \sum_{o=1}^{T}\sum_{i=1}^{N}\left(-\frac{1}{2}(\log 2\pi + \Sigma(t_{o,i})) - \log\mathcal{Z}_{o,i}\right).$$

The posterior expectations needed in order to update approximate posterior distributions and to calculate the lower bound can be given as

$$\widetilde{\phi_{o,s}^f} = \alpha(\phi_{o,s}^f)\beta(\phi_{o,s}^f) \qquad \forall(o,f,s)$$

$$\widetilde{\log\phi_{o,s}^f} = \psi(\alpha(\phi_{o,s}^f)) + \log\beta(\phi_{o,s}^f) \qquad \forall(o,f,s)$$

$$\widetilde{\boldsymbol{\phi}_{o,s}} = \begin{bmatrix}\alpha(\phi_{o,s}^1)\beta(\phi_{o,s}^1) & \alpha(\phi_{o,s}^2)\beta(\phi_{o,s}^2) & \cdots & \alpha(\phi_{o,s}^{D_o})\beta(\phi_{o,s}^{D_o})\end{bmatrix}^\top \qquad \forall(o,s)$$

$$\widetilde{(q_{o,s}^f)^2} = \mu(q_{o,s}^f)^2 + \Sigma(q_{o,s}^f) \qquad \forall(o,f,s)$$

$$\widetilde{\boldsymbol{q}_{o,s}\boldsymbol{q}_{o,s}^\top} = \mu(\boldsymbol{q}_{o,s})\mu(\boldsymbol{q}_{o,s})^\top + \Sigma(\boldsymbol{q}_{o,s}) \qquad \forall(o,s)$$

$$\widetilde{\mathbf{Q}_o} = \begin{bmatrix}\mu(\boldsymbol{q}_{o,1}) & \mu(\boldsymbol{q}_{o,2}) & \cdots & \mu(\boldsymbol{q}_{o,R})\end{bmatrix} \qquad \forall o$$

$$
\begin{aligned}
\widetilde{\mathbf{Q}_o\mathbf{Q}_o^\top} &= \sum_{s=1}^{R}(\mu(\boldsymbol{q}_{o,s})\mu(\boldsymbol{q}_{o,s})^\top + \Sigma(\boldsymbol{q}_{o,s})) && \forall o \\
\widetilde{\boldsymbol{z}_{o,i}} &= \begin{bmatrix}\mu(z^1_{o,i}) & \mu(z^2_{o,i}) & \cdots & \mu(z^R_{o,i})\end{bmatrix}^\top && \forall (o,i) \\
\widetilde{\boldsymbol{z}^s_o} &= \begin{bmatrix}\mu(z^s_{o,1}) & \mu(z^s_{o,2}) & \cdots & \mu(z^s_{o,N_o})\end{bmatrix} && \forall (o,s) \\
\widetilde{\boldsymbol{z}^\top_{o,i}\boldsymbol{z}_{o,i}} &= \sum_{s=1}^{R}(\mu(z^s_{o,i})^2 + \Sigma(z^s_{o,i})) && \forall (o,i) \\
\widetilde{\boldsymbol{z}_{o,i}\boldsymbol{z}^\top_{o,i}} &= \mu(\boldsymbol{z}_{o,i})\mu(\boldsymbol{z}_{o,i})^\top + \Sigma(\boldsymbol{z}_{o,i}) && \forall (o,i) \\
\widetilde{\mathbf{Z}_o} &= \begin{bmatrix}\mu(\boldsymbol{z}_{o,1}) & \mu(\boldsymbol{z}_{o,2}) & \ldots & \mu(\boldsymbol{z}_{o,N_o})\end{bmatrix} && \forall o \\
\widetilde{\mathbf{Z}_o\mathbf{Z}_o^\top} &= \sum_{i=1}^{N_o}(\mu(\boldsymbol{z}_{o,i})\mu(\boldsymbol{z}_{o,i})^\top + \Sigma(\boldsymbol{z}_{o,i})) && \forall o \\
\widetilde{\lambda} &= \alpha(\lambda)\beta(\lambda) \\
\widetilde{\log\lambda} &= \psi(\alpha(\lambda)) + \log\beta(\lambda) \\
\widetilde{\psi_s} &= \alpha(\psi_s)\beta(\psi_s) && \forall s \\
\widetilde{\log\psi_s} &= \psi(\alpha(\psi_s)) + \log\beta(\psi_s) && \forall s \\
\widetilde{\boldsymbol{\psi}} &= \begin{bmatrix}\alpha(\psi_1)\beta(\psi_1) & \alpha(\psi_2)\beta(\psi_2) & \cdots & \alpha(\psi_R)\beta(\psi_R)\end{bmatrix}^\top \\
\widetilde{b} &= \mu(b) \\
\widetilde{b^2} &= \mu(b)^2 + \Sigma(b) \\
\widetilde{w_s^2} &= \mu(w_s)^2 + \Sigma(w_s) \\
\widetilde{\boldsymbol{w}} &= \begin{bmatrix}\mu(w_1) & \mu(w_2) & \cdots & \mu(w_R)\end{bmatrix}^\top \\
\widetilde{\boldsymbol{w}\boldsymbol{w}^\top} &= \mu(\boldsymbol{w})\mu(\boldsymbol{w})^\top + \Sigma(\boldsymbol{w}).
\end{aligned}
$$

From the model definition, the truncation points for each auxiliary variable are defined as

$$
(l_{o,i}, u_{o,i}) = \begin{cases}(-\infty, 0) & \text{if } y_{o,i} = -1 \\ (0, +\infty) & \text{otherwise}\end{cases} \qquad \forall (o,i)
$$

where $l_{o,i}$ and $u_{o,i}$ denote the lower and upper truncation points, respectively. The normalization coefficient, the expectation, and the variance of the auxiliary variables can be calculated as

$$\begin{aligned}
\mathcal{Z}_{o,i} &= \Phi(\beta_{o,i}) - \Phi(\alpha_{o,i}) && \forall(o,i) \\
\widetilde{t_{o,i}} &= \widetilde{\boldsymbol{w}^\top}\widetilde{\boldsymbol{z}_{o,i}} + \widetilde{b} + \frac{\phi(\alpha_i) - \phi(\beta_i)}{\mathcal{Z}_{o,i}} && \forall(o,i) \\
\widetilde{t_{o,i}^2} - (\widetilde{t_{o,i}})^2 &= 1 + \frac{\alpha_{o,i}\phi(\alpha_{o,i}) - \beta_{o,i}\phi(\beta_{o,i})}{\mathcal{Z}_{o,i}} - \frac{(\phi(\alpha_{o,i}) - \phi(\beta_{o,i}))^2}{\mathcal{Z}_{o,i}^2} && \forall(o,i)
\end{aligned}$$

where $\{\alpha_{o,i}, \beta_{o,i}\}$ are defined as

$$\begin{aligned}
\alpha_{o,i} &= l_{o,i} - \widetilde{\boldsymbol{w}^\top}\widetilde{\boldsymbol{z}_{o,i}} - \widetilde{b} && \forall(o,i) \\
\beta_{o,i} &= u_{o,i} - \widetilde{\boldsymbol{w}^\top}\widetilde{\boldsymbol{z}_{o,i}} - \widetilde{b} && \forall(o,i).
\end{aligned}$$

5.2. Prediction

After convergence, we have a separate projection matrix for each data set and a unified set of classification parameters for the projected subspace. For a test data point from a particular data set, we can perform dimensionality reduction and classification using the corresponding projection matrix and the shared classification parameters. $p(\mathbf{Q}_o|\{\mathbf{X}_o\}_{o=1}^T, \{\boldsymbol{y}_o\}_{o=1}^T)$ can be replaced with its approximate posterior distribution $q(\mathbf{Q}_o)$ for the prediction step. We obtain the predictive distribution of the projected instance $\boldsymbol{z}_{o,\star}$ for a new data point $\boldsymbol{x}_{o,\star}$ from a particular data set as

$$p(\boldsymbol{z}_{o,\star}|\boldsymbol{x}_{o,\star}, \{\mathbf{X}_o\}_{o=1}^T, \{\boldsymbol{y}_o\}_{o=1}^T) = \prod_{s=1}^{R} \mathcal{N}(z_{o,\star}^s; \mu(\boldsymbol{q}_{o,s})^\top \boldsymbol{x}_{o,\star}, 1 + \boldsymbol{x}_{o,\star}^\top \Sigma(\boldsymbol{q}_{o,s})\boldsymbol{x}_{o,\star}).$$

The predictive distribution of the auxiliary variable $t_{o,\star}$ can also be found by replacing $p(b, \boldsymbol{w}|\{\mathbf{X}_o\}_{o=1}^T, \{\boldsymbol{y}_o\}_{o=1}^T)$ with its approximate posterior distribution $q(b, \boldsymbol{w})$:

$$p(t_{o,\star}|\{\mathbf{X}_o\}_{o=1}^T, \{\boldsymbol{y}_o\}_{o=1}^T, \boldsymbol{z}_{o,\star}) = \mathcal{N}\left(t_{o,\star}; \mu(b, \boldsymbol{w})^\top \begin{bmatrix} 1 \\ \boldsymbol{z}_{o,\star} \end{bmatrix}, 1 + \begin{bmatrix} 1 & \boldsymbol{z}_{o,\star} \end{bmatrix} \Sigma(b, \boldsymbol{w}) \begin{bmatrix} 1 \\ \boldsymbol{z}_{o,\star} \end{bmatrix}\right)$$

and the predictive distribution of the class label $y_{o,\star}$ can be formulated using the auxiliary variable distribution:

$$p(y_{o,\star} = +1|\boldsymbol{x}_{o,\star}, \{\mathbf{X}_o\}_{o=1}^T, \{\boldsymbol{y}_o\}_{o=1}^T) = \Phi\left(\frac{\mu(t_{o,\star})}{\Sigma(t_{o,\star})}\right).$$

6. EXPERIMENTS

We perform both feature selection and transfer learning experiments on two CRA data sets to test the proposed algorithms. We first explain the data sets and the performance measures used in our experiments. We then report the experimental results and discuss them from different perspectives including predictive accuracy, sparsity, and convergence rate.

6.1. Data Sets

We use two widely used benchmark CRA data sets, namely, `German Credit` and `Australian Credit`, to test the algorithms.

6.1.1. Australian Credit Data Set

`Australian Credit` data set contains 690 data points represented with 14 variables (6 continuous and 8 categorical). When we encode categorical variables as binary features using 1-of-k encoding, there are 38 features in total. This data set is available at `http://archive.ics.uci.edu/ml/datasets/Statlog+(Australian+Credit+Approval)`. The variable information of `Australian Credit` data set is summarized in Table 6.1.

6.1.2. German Credit Data Set

`German Credit` data set contains 1000 data points represented with 20 variables (7 continuous and 13 categorical). When we encode categorical variables as binary features using 1-of-k encoding, there are 59 features in total. This data set is available at `http://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)`. The variable information of `German Credit` data set is summarized in Table 6.2.

Table 6.1. Variable information of `Australian Credit` data set.

Variable	**Type**	**Value**
A1	Categorical	$0, 1$
A2	Continuous	
A3	Continuous	
A4	Categorical	$1, 2, 3$
A5	Categorical	$1, 2, \ldots, 14$
A6	Categorical	$1, 2, \ldots, 9$
A7	Continuous	
A8	Categorical	$0, 1$
A9	Categorical	$0, 1$
A10	Continuous	
A11	Categorical	$0, 1$
A12	Categorical	$1, 2, 3$
A13	Continuous	
A14	Continuous	

6.2. Performance Measures

We report the experimental results in terms of two popular performance measures: classification accuracy and area under *receiver operating characteristic* (ROC) curve. We shortly explain these two measures for binary classification problems such as CRA.

6.2.1. Classification Accuracy

When a binary classification algorithm makes a prediction for a data point, there are four possible outcomes as shown in Table 6.3.

- *True positive* (TP): The tested data point is a positive example and the prediction is also positive.

Table 6.2. Variable information of `German Credit` data set.

Variable	**Information**	**Type**	**Value**
A1	Status of existing checking account	Categorical	$1, 2, 3, 4$
A2	Duration in month	Continuous	
A3	Credit history	Categorical	$0, 1, \ldots, 4$
A4	Purpose	Categorical	$0, 1, \ldots, 10$
A5	Credit amount	Continuous	
A6	Saving account / bonds	Categorical	$0, 1, \ldots, 5$
A7	Present employment since	Categorical	$1, 2, \ldots, 5$
A8	Installment rate in percentage of disposable income	Continuous	
A9	Personal status and sex	Categorical	$1, 2, \ldots, 5$
A10	Other debtors / guarantors	Categorical	$1, 2, 3$
A11	Present residence since	Continuous	
A12	Property	Categorical	$1, 2, 3, 4$
A13	Age in years	Continuous	
A14	Other installment plans	Categorical	$1, 2, 3$
A15	Housing	Categorical	$1, 2, 3$
A16	Number of existing credits at this bank	Continuous	
A17	Job	Categorical	$1, 2, 3, 4$
A18	Number of people being liable to provide maintenance for	Continuous	
A19	Telephone	Categorical	$1, 2$
A20	Foreign worker	Categorical	$1, 2$

- *False negative* (FN): The tested data point is a positive example and the prediction is negative.
- *True negative* (TN): The tested data point is a negative example and the prediction is also negative.
- *False positive* (FP): The tested data point is a negative example and the prediction is positive.

Table 6.3. Confusion matrix for binary classification problems.

		Predicted Class	
		Positive	**Negative**
True Class	**Positive**	*True positive* (TP)	*False negative* (FN)
	Negative	*False positive* (FP)	*True negative* (TN)

Classification accuracy is defined as the ratio between the number of correctly classified data points and the total number of classified data points. It can be calculated from the confusion matrix entries as

$$\frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN}}$$

where the ratio is equal to one for a perfect classifier.

6.2.2. Area Under Curve

An ROC curve illustrates the performance of a binary classifier as the discrimination threshold is varied. It is created by plotting *true positive rate* (TPR) versus *false positive rate* (FPR) at various discrimination thresholds. TPR is defined as the ratio between TP and the number of positive data points

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

and FPR is defined as the ratio between FP and the number of negative data points

$$\frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}.$$

Area under curve (AUC) is defined as the area under the ROC curve and it gives the probability that a classifier will rank a randomly chosen positive data point higher than a randomly chosen negative data point.

6.3. Feature Selection Experiments

We test the probit classifier (PROBIT) and the ℓ_p-norm MKL (ℓ_p-MKL) algorithms on two different CRA data sets explained above. The experimental procedure used and the results obtained are given in the following sections.

6.3.1. Experimental Setup

We implement variational approximation method for PROBIT and alternating optimization procedure for ℓ_p-MKL in Matlab. We take 500 iterations for PROBIT and run ℓ_p-MKL algorithm until the objective function does not improve at least 0.1 per cent between successive iterations.

For both algorithms, we try four different scenarios: (i) non-sparse, (ii) sparse, (iii) grouped non-sparse, and (iv) grouped sparse. Table 6.4 summarizes the parameter values used for these four scenarios. For grouped scenarios, the binary features that are obtained from the categorial variables using 1-of-k encoding are defined as feature groups, whereas each continuous variable is used as a separate feature group. We use the Gaussian kernel on each feature group for ℓ_p-MKL algorithm. The kernel width s is selected as the square root of the number of features used.

Table 6.4. Parameter values for different scenarios of feature selection experiments.

		non-sparse	sparse	grouped non-sparse	grouped sparse
PROBIT	$(\alpha_\lambda, \beta_\lambda)$	$(1, 1)$	$(1, 1)$	$(1, 1)$	$(1, 1)$
	$(\alpha_\psi, \beta_\psi)$	$(1, 1)$	$(10^{-10}, 10^{+10})$	$(1, 1)$	$(10^{-10}, 10^{+10})$
ℓ_p-MKL	p	2	1	2	1
	C	$\{10^{-2}, \ldots, 10^{2}\}$	$\{10^{-2}, \ldots, 10^{2}\}$	$\{10^{-2}, \ldots, 10^{2}\}$	$\{10^{-2}, \ldots, 10^{2}\}$

For both data sets, we take 50 replications, where we randomly select 80 per cent of the data set as the training set and use the remaining as the test set. After constructing the binary features from the categorical variables, the training set is normalized to

have zero mean and unit standard deviation, and the test set is then normalized using the mean and the standard deviation of the training set. We use 5-fold cross-validation on the training set to pick the regularization parameter C from the candidate set. We report generalization performances in terms of mean classification accuracy and mean AUC over 50 replications.

6.3.2. Results

Table 6.5 gives the classification results of PROBIT and ℓ_p-MKL on Australian Credit data set. PROBIT obtains the same level of performance (≈0.855 accuracy and ≈0.925 AUC) with different scenarios. We see that enforcing sparsity does not harm the classification performance at all. However, ℓ_p-MKL achieves slightly lower performance values (≈0.850 accuracy and ≈0.915 AUC). There is again no significant difference between the scenarios in terms of classification performance.

Table 6.5. Classification results on Australian Credit data set in feature selection experiments.

	PROBIT		ℓ_p-MKL	
	Accuracy	**AUC**	**Accuracy**	**AUC**
non-sparse	0.855 ± 0.024	0.923 ± 0.017	0.848 ± 0.020	0.916 ± 0.019
sparse	0.854 ± 0.022	0.927 ± 0.019	0.852 ± 0.022	0.912 ± 0.021
grouped non-sparse	0.856 ± 0.025	0.925 ± 0.016	0.850 ± 0.021	0.916 ± 0.018
grouped sparse	0.854 ± 0.022	0.923 ± 0.020	0.851 ± 0.023	0.912 ± 0.019

We report the selected variables by PROBIT for each scenario on Australian Credit data set in Table 6.6. If we do not enforce feature selection as in non-sparse and grouped non-sparse scenarios, PROBIT uses 11 variables in its decision function. However, when we enforce feature selection as in sparse and grouped sparse scenarios, it only uses nine and seven variables, respectively. PROBIT with grouped sparse scenario achieves the same level of classification performance as other scenarios using only 50 per cent of the input variables. Figure 6.1 shows the feature weights obtained by PROBIT for each scenario on Australian Credit data set. PROBIT can eliminate

some of the input features or feature groups by assigning them zero weights. The effect of sparsity or group sparsity can clearly be seen from the feature weights of `sparse` and `grouped sparse` scenarios.

Table 6.6. Variables selected by `PROBIT` on `Australian Credit` data set in feature selection experiments.

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14
`non-sparse`	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
`sparse`	✗	✗	✗	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	✓
`grouped non-sparse`	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
`grouped sparse`	✗	✗	✗	✓	✗	✗	✓	✓	✓	✓	✗	✓	✗	✓

We report the selected variables by ℓ_p-`MKL` for each scenario on `Australian Credit` data set in Table 6.7. For `non-sparse` and `grouped non-sparse` scenarios, ℓ_p-`MKL` uses 12 variables in its decision function. However, it only uses nine variables for `sparse` and `grouped sparse` scenarios. Different from `PROBIT`, ℓ_p-`MKL` can only eliminate three out of 12 variables. Figure 6.2 shows the feature weights obtained by ℓ_p-`MKL` for each scenario on `Australian Credit` data set. ℓ_p-`MKL` can eliminate some of the input features or feature groups by assigning corresponding kernels zero weights. For `sparse` and `grouped sparse` scenarios, ℓ_p-`MKL` eliminates more of the input kernels, leading to tighter feature and feature group selection.

Table 6.7. Variables selected by ℓ_p-`MKL` on `Australian Credit` data set in feature selection experiments.

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14
`non-sparse`	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
`sparse`	✗	✓	✓	✓	✓	✗	✓	✓	✗	✗	✗	✓	✓	✓
`grouped non-sparse`	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
`grouped sparse`	✗	✓	✓	✓	✓	✗	✓	✓	✗	✗	✗	✓	✓	✓

Table 6.8 gives the classification results obtained by `PROBIT` and ℓ_p-`MKL` on `German Credit` data set. `PROBIT` obtains the same level of performance ($\approx$0.750 accuracy and $\approx$0.780 AUC) with different scenarios. We again see that enforcing sparsity does not

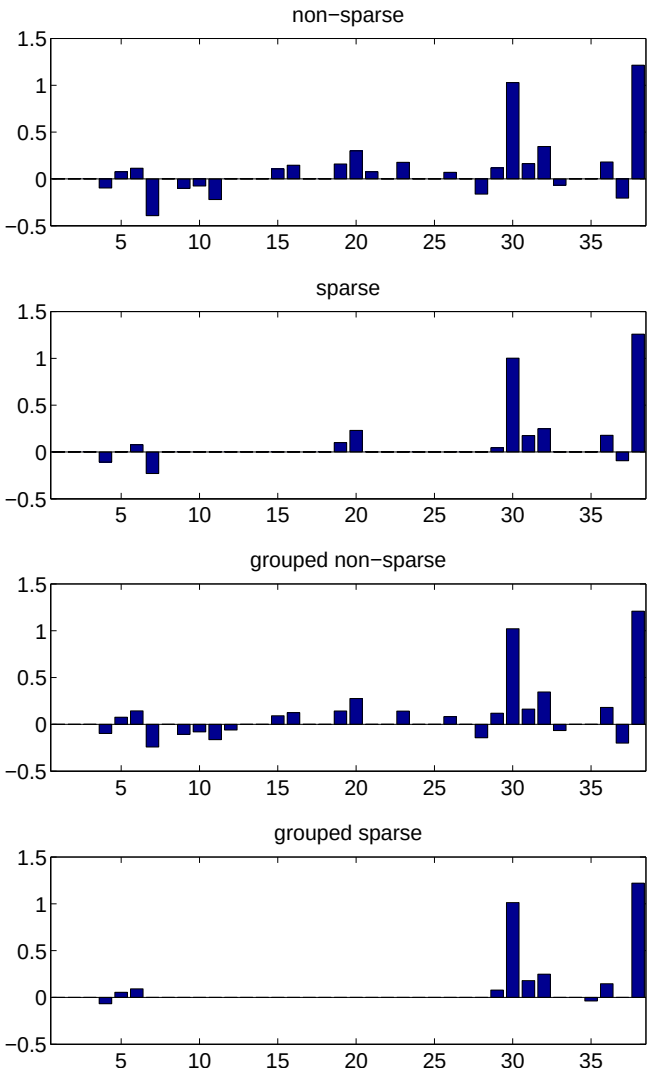


Figure 6.1. Feature weights obtained by `PROBIT` on `Australian Credit` data set in feature selection experiments.

harm the classification performance at all. Different from `Australian Credit` data set, ℓ_p-`MKL` also achieves the same level of classification performance. There is again no significant difference between the scenarios in terms of classification performance.

We report the selected variables by `PROBIT` for each scenario on `German Credit` data set in Table 6.9. If we do not enforce feature selection as in `non-sparse` and `grouped non-sparse` scenarios, `PROBIT` uses 18 variables in its decision function. However, when we enforce feature selection as in `sparse` and `grouped sparse` scenarios, it only uses 15 and 13 variables, respectively. `PROBIT` with `grouped sparse` scenario achieves the same level of classification performance as other scenarios using only 70 per

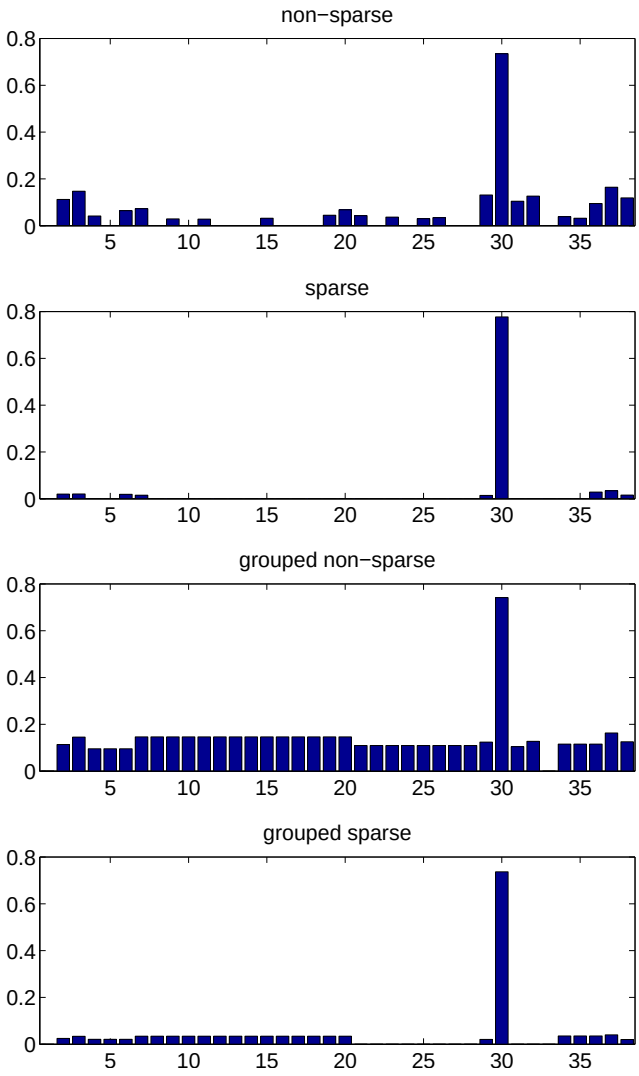


Figure 6.2. Feature weights obtained by ℓ_p-MKL on Australian Credit data set in feature selection experiments.

Table 6.8. Classification results on German Credit data set in feature selection experiments.

	PROBIT		ℓ_p-MKL	
	Accuracy	**AUC**	**Accuracy**	**AUC**
non-sparse	0.752 ± 0.031	0.782 ± 0.030	0.752 ± 0.026	0.778 ± 0.029
sparse	0.746 ± 0.032	0.776 ± 0.033	0.750 ± 0.026	0.776 ± 0.029
grouped non-sparse	0.752 ± 0.031	0.782 ± 0.030	0.754 ± 0.026	0.780 ± 0.030
grouped sparse	0.750 ± 0.030	0.776 ± 0.035	0.752 ± 0.029	0.780 ± 0.031

cent of the input variables. Figure 6.3 shows the feature weights obtained by PROBIT for each scenario on German Credit data set. The effect of sparsity or group sparsity can clearly be seen from the feature weights of sparse and grouped sparse scenarios.

Table 6.9. Variables selected by PROBIT on German Credit data set in feature selection experiments.

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
non-sparse	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗	✓	✓	✓
sparse	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	✓
grouped non-sparse	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗	✓	✓	✓
grouped sparse	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✓

We report the selected variables by ℓ_p-MKL for each scenario on German Credit data set in Table 6.10. For non-sparse and grouped non-sparse scenarios, ℓ_p-MKL uses 18 and 16 variables, respectively, in its decision function. However, it only uses 16 and 13 variables for sparse and grouped sparse scenarios, respectively. Figure 6.4 shows the feature weights obtained by ℓ_p-MKL for each scenario on German Credit data set. For sparse and grouped sparse scenarios, ℓ_p-MKL eliminates more of the input kernels, leading to tighter feature and feature group selection.

Table 6.10. Variables selected by ℓ_p-MKL on German Credit data set in feature selection experiments.

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
non-sparse	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓
sparse	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓
grouped non-sparse	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓
grouped sparse	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗

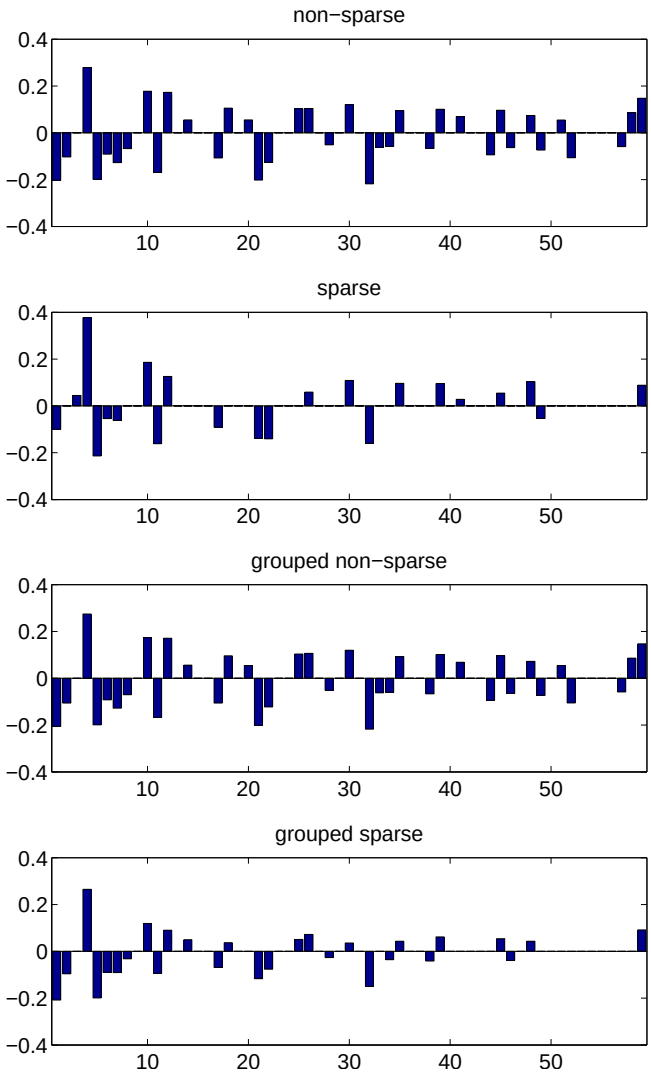


Figure 6.3. Feature weights obtained by `PROBIT` on `German Credit` data set in feature selection experiments.

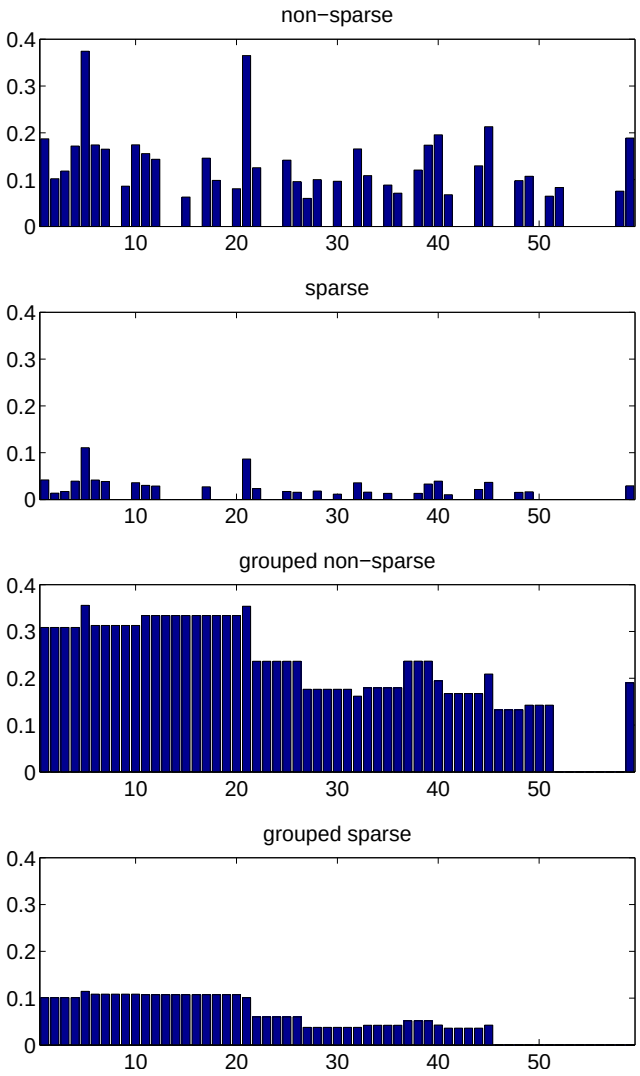


Figure 6.4. Feature weights obtained by ℓ_p-`MKL` on `German Credit` data set in feature selection experiments.

6.3.3. Convergence Analysis

In order to perform a convergence analysis, we monitor the lower bound of PROBIT and the number of iterations performed by ℓ_p-MKL. We report the convergence results on both CRA data sets.

Figure 6.5 shows the lower bounds of four different scenarios for a single replication obtained by PROBIT on Australian Credit data set. PROBIT algorithm converges very quickly (i.e., in tens of iterations) regardless of the parameter setting used.

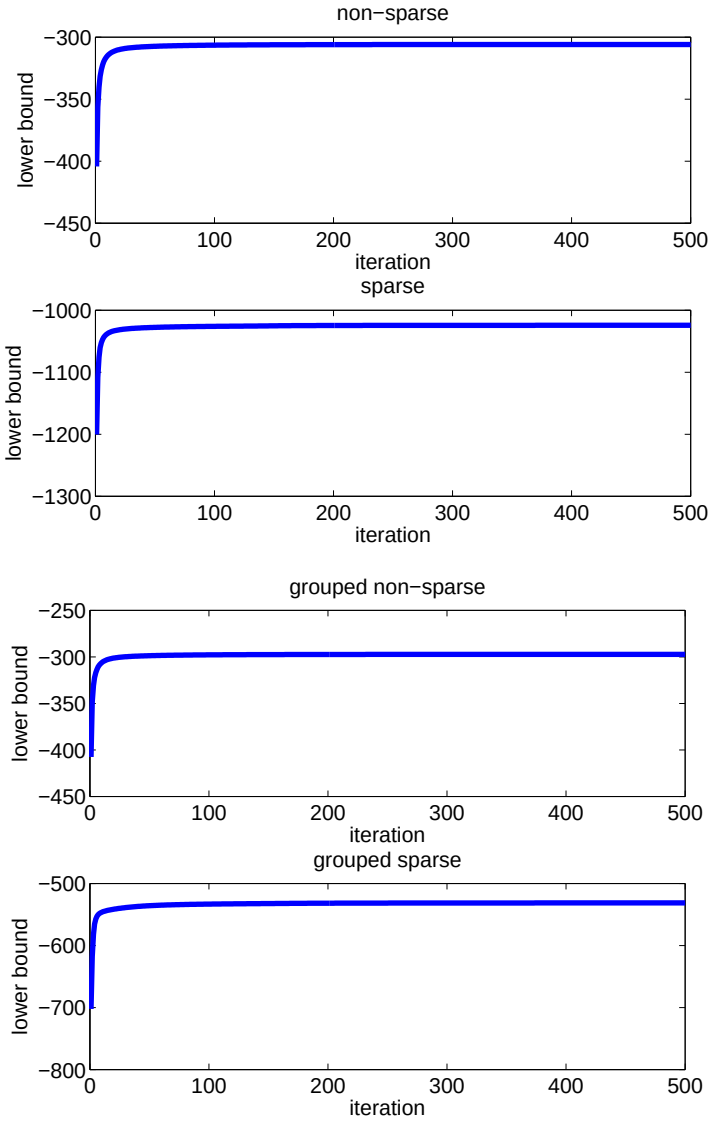


Figure 6.5. PROBIT lower bounds on Australian Credit data set in feature selection experiments.

Note that the lower bounds of these four scenarios are not directly comparable due to differences in the number of variables and distributional assumptions.

Figure 6.6 shows the number of iterations required by ℓ_p-MKL until convergence for four different scenarios on Australian Credit data set. The results are reported for 50 replications separately. We see that ℓ_p-MKL requires less than 10 iterations on the average and the behavior is independent of scenario configuration.

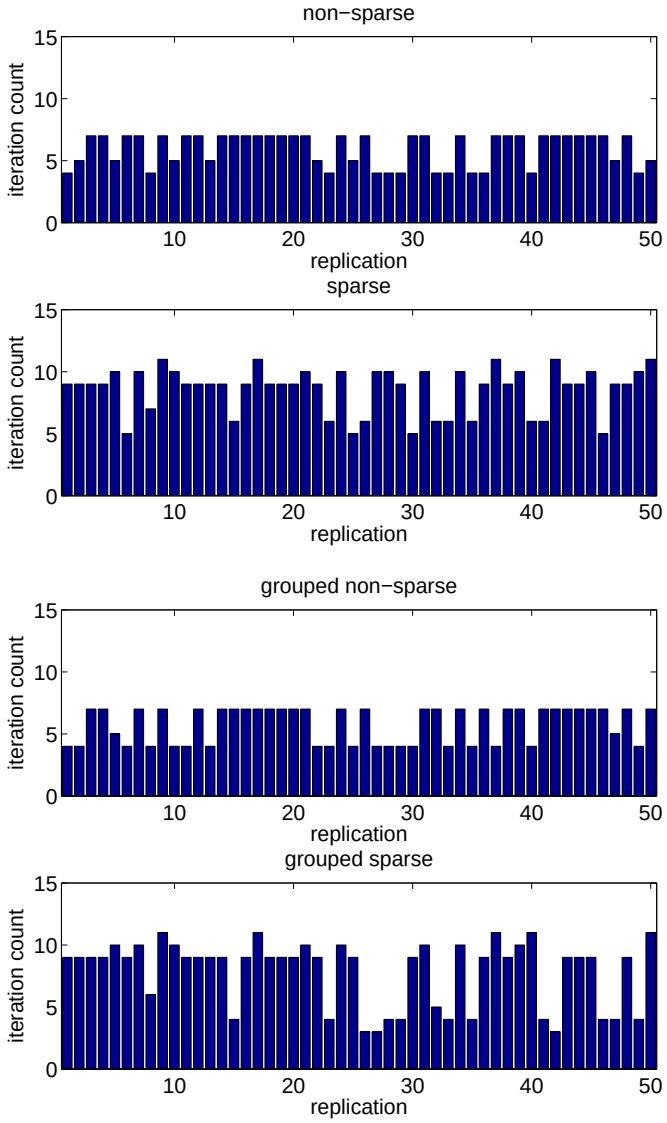


Figure 6.6. ℓ_p-MKL iteration counts on Australian Credit data set in feature selection experiments.

Figure 6.7 shows the lower bounds of four different scenarios for a single replication obtained by PROBIT on German Credit data set. Similar to the results on Australian Credit data set, PROBIT algorithm converges in tens of iterations regardless of the parameter setting used.

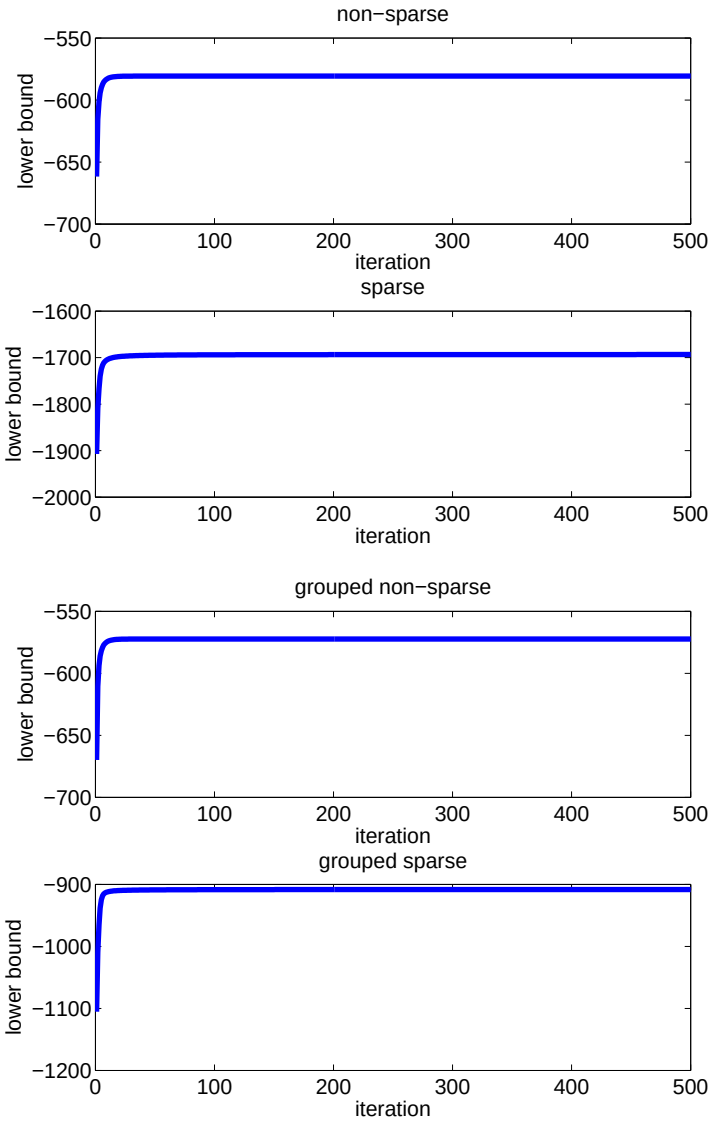


Figure 6.7. PROBIT lower bounds on German Credit data set in feature selection experiments.

Figure 6.8 shows the number of iterations required by ℓ_p-MKL until convergence for four different scenarios on German Credit data set. Similar to the results on Australian Credit data set, ℓ_p-MKL requires less than 10 iterations on the average.

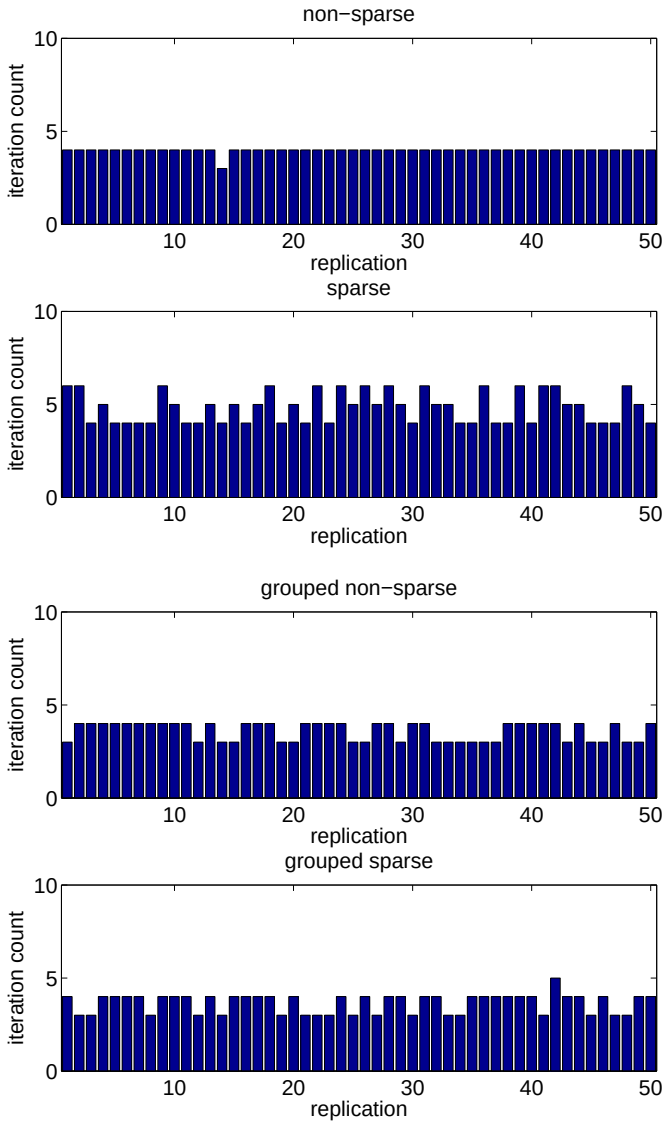


Figure 6.8. ℓ_p-MKL iteration counts on German Credit data set in feature selection experiments.

6.4. Transfer Learning Experiments

We test our proposed transfer learning probit classifier (TRANSFER-PROBIT) on two different CRA data sets explained above. The experimental procedure used and the results obtained are given in the following sections.

6.4.1. Experimental Setup

We try two different scenarios: (i) `German Credit` ⇒ `Australian Credit`, and (ii) `Australian Credit` ⇒ `German Credit`. The arrow shows the direction of information transfer between data sets, where the one on the left is called *source data set* and the other one is called *target data set*.

For the target data set, we use the same train-test splits from the feature selection experiments in order to have comparable results. However, we use 10 or 20 per cent of the training set to clearly see the effect of transfer learning. For the source data set, we use the whole data set to transfer all available information to the target data set. The training sets are normalized to have zero mean and unit standard deviation, and the test set is then normalized using the mean and the standard deviation of the training set. We compare `TRANSFER-PROBIT` with `PROBIT` trained only on the target data set. We take 50 replications and report generalization performances on the target data set in terms of mean AUC over 50 replications.

We also implement variational approximation method for `TRANSFER-PROBIT` in Matlab and take 500 iterations for both `PROBIT` and `TRANSFER-PROBIT`. Table 6.11 summarizes the parameter values used in the transfer learning experiments. We use uninformative priors for precision parameters and experiment with different values for the subspace dimensionality of `TRANSFER-PROBIT`.

Table 6.11. Parameter values of transfer learning experiments.

	$(\alpha_\lambda, \beta_\lambda)$	$(\alpha_\phi, \beta_\phi)$	$(\alpha_\psi, \beta_\psi)$	R
`PROBIT`	$(1, 1)$	—	$(1, 1)$	—
`TRANSFER-PROBIT`	$(1, 1)$	$(1, 1)$	$(1, 1)$	$\{1, \ldots, 10\}$

6.4.2. Results

Figure 6.9 compares the classification results of `PROBIT` and `TRANSFER-PROBIT` on `Australian Credit` data set. We clearly see that `TRANSFER-PROBIT` significantly

outperforms PROBIT in terms of mean AUC value. This shows that when we have a limited amount of training data for the target data set, transfer learning helps us improve our generalization performance using training data from a different but related data set. Using German Credit data set as an additional information source improves the classification results even with one-dimensional unified subspace.

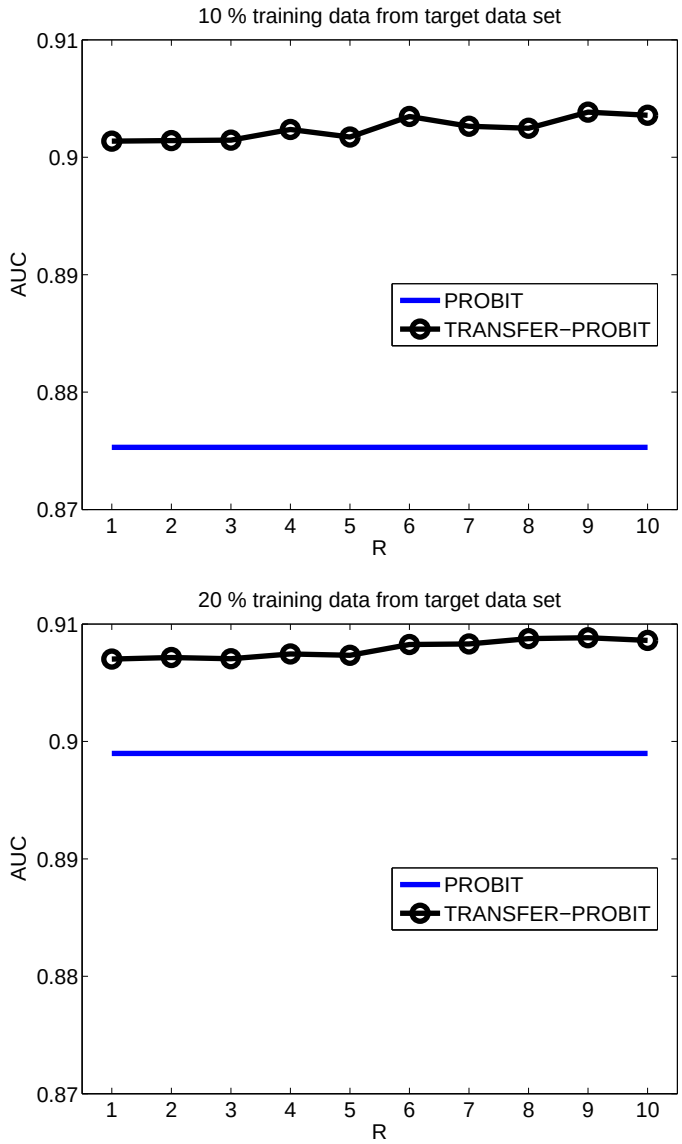


Figure 6.9. Classification results on Australian Credit data set in transfer learning experiments.

Figure 6.10 compares the classification results of PROBIT and TRANSFER-PROBIT on German Credit data set. Similar to the results on Australian Credit data set, TRANSFER-PROBIT significantly outperforms PROBIT in terms of mean AUC value.

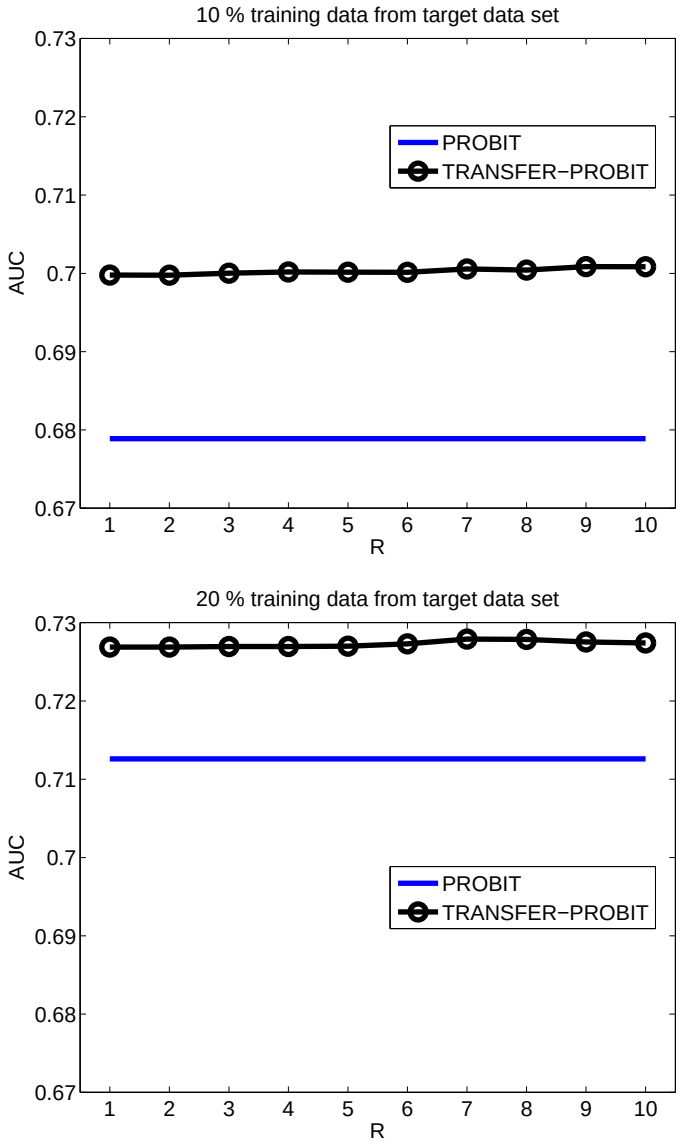


Figure 6.10. Classification results on `German Credit` data set in transfer learning experiments.

7. CONCLUSIONS

In this thesis, we first introduce two different binary classification algorithm variants with group-wise feature selection capability. These variants are derived from the probit classifier using a specific prior structure and from multiple kernel learning using a specific kernel calculation strategy. These two formulations allow us to perform group-wise selection on feature groups of CRA (i.e., features obtained from categorial variables using 1-of-k encoding).

Experimental results on two benchmark CRA data sets show the validity and effectiveness of the proposed binary classification algorithm variants. Our methods use fewer features compared to non-sparse alternatives without sacrificing classification performance in terms of both classification accuracy and AUC. They allow us to decrease data acquisition cost and testing time in addition to better interpretability of the results.

We then extend the probit classifier towards transfer learning. Our method maps data points from different data sets into a unified subspace and learns a shared classification model in that subspace. The proposed transfer learning algorithm can make use of several related data sets and transfer useful information between the data sets to improve the overall generalization performance. This setup is especially useful when a data set has a limited amount of training data and we can a learn better classifier for that data set with the help of other data sets.

We also perform transfer learning experiments on two benchmark CRA data sets to show the performance of our probit classifier variant. We show that the proposed method is able to transfer information from one CRA data set to the other one successfully. Transfer learning setup obtains significantly better results than no-transfer setup even with one-dimensional unified subspace.

The methods developed in this thesis are tested only on two benchmark CRA data sets. An interesting topic for future research is to test them on data sets from different domains. For example, group-wise feature selection algorithms can be tested on bioinformatics data sets that have a natural grouping over the features. Moreover, robotics and computer vision applications are natural candidates for our transfer learning method.

APPENDIX A: MATLAB SOURCE CODES

In this chapter, we give Matlab source codes for variational approximation procedures of the probit classifiers with group-wise feature selection and transfer learning capabilities, respectively. We also provide numerically safer versions of helper functions they need.

```
function state = grouped_probit_variational_train(X, y, parameters)
    rand('state', parameters.seed); %#ok<RAND>
    randn('state', parameters.seed); %#ok<RAND>

    D = size(X, 1);
    N = size(X, 2);
    G = length(unique(parameters.grouping));

    lambda.shape = (parameters.alpha_lambda + 0.5);
    lambda.scale = parameters.beta_lambda;
    tau.shape = (parameters.alpha_psi + 0.5 * histc(parameters.grouping, 1:1:G)');
    tau.scale = parameters.beta_psi * ones(G, 1);
    bw.mean = randn(D + 1, 1);
    bw.covariance = eye(D + 1, D + 1);
    t.mean = zeros(N, 1);
    for i = 1:N
        while 1
            t.mean(i) = randn();
            if t.mean(i) * y(i) > 0
                break;
            end
        end
    end

    XXT = X * X';
    X1 = X * ones(N, 1);
    lower = -1e40 * ones(N, 1);
    lower(y > 0) = 0;
    upper = +1e40 * ones(N, 1);
    upper(y < 0) = 0;

    bounds = zeros(parameters.iteration, 1);

    for iter = 1:parameters.iteration
        if mod(iter, 10) == 0
            fprintf(1, '.');
        end
```

```
if mod(iter, 100) == 0
    fprintf(1, ' %5d\n', iter);
end

%%% update lambda
lambda.scale = 1 / (1 / parameters.beta_lambda + 0.5 * (bw.mean(1)^2 + bw.↩
    covariance(1, 1)));
%%% update tau
for g = 1:G
    indices = 1 + find(parameters.grouping == g);
    tau.scale(g) = 1 / (1 / parameters.beta_psi + 0.5 * sum(bw.mean(indices)↩
        .^2 + diag(bw.covariance(indices, indices))));
end
%%% update b and w
bw.covariance = [N + lambda.shape * lambda.scale, X1'; X1, XXT + diag(tau.↩
    shape(parameters.grouping) .* tau.scale(parameters.grouping))] \ eye(D +↩
     1, D + 1);
bw.mean = bw.covariance * [ones(1, N); X] * t.mean;
%%% update t
t.mean = [ones(1, N); X]' * bw.mean;
t.mean = t.mean + (safenormpdf(lower - t.mean) - safenormpdf(upper - t.mean)↩
    ) ./ (safenormcdf(upper - t.mean) - safenormcdf(lower - t.mean));

lb = 0;
%%% p(lambda)
lb = lb + (parameters.alpha_lambda - 1) * (psi(lambda.shape) + safelog(↩
    lambda.scale)) - lambda.shape * lambda.scale / parameters.beta_lambda - ↩
    gammaln(parameters.alpha_lambda) - parameters.alpha_lambda * safelog(↩
    parameters.beta_lambda);
%%% p(b | lambda)
lb = lb - 0.5 * bw.mean(1).^2 * lambda.shape * lambda.scale - 0.5 * (safelog↩
    (2 * pi) - safelog(lambda.shape * lambda.scale));
%%% p(tau)
lb = lb + sum((parameters.alpha_psi - 1) * (psi(tau.shape) + safelog(tau.↩
    scale)) - tau.shape .* tau.scale / parameters.beta_psi - gammaln(↩
    parameters.alpha_psi) - parameters.alpha_psi * safelog(parameters.↩
    beta_psi));
%%% p(w | tau)
lb = lb - 0.5 * bw.mean(2:D + 1)' * diag(tau.shape(parameters.grouping) .* ↩
    tau.scale(parameters.grouping)) * bw.mean(2:D + 1) - 0.5 * (D * safelog↩
    (2 * pi) - sum(safelog(tau.shape(parameters.grouping) .* tau.scale(↩
    parameters.grouping))));
%%% p(t | b, w, X) p(y | t)
wwT.mean = bw.mean(2:D + 1) * bw.mean(2:D + 1)' + bw.covariance(2:D + 1, 2:D↩
     + 1);
lb = lb - 0.5 * (t.mean' * t.mean + N) + sum(bw.mean(1) * t.mean) + sum(diag↩
    (X' * bw.mean(2:D + 1, :) * t.mean')) - 0.5 * sum(diag(X' * wwT.mean * X↩
    )) - 0.5 * N * (bw.mean(1).^2 + bw.covariance(1, 1)) - sum(X' * (bw.mean↩
```

```
            (2:D + 1) * bw.mean(1) + bw.covariance(2:D + 1, 1))) - 0.5 * N * safelog↩
            (2 * pi);

        %%%% q(lambda)
        lb = lb + lambda.shape + safelog(lambda.scale) + gammaln(lambda.shape) + (1 ↩
            - lambda.shape) * psi(lambda.shape);
        %%%% q(tau)
        lb = lb + sum(tau.shape + safelog(tau.scale) + gammaln(tau.shape) + (1 - tau↩
            .shape) .* psi(tau.shape));
        %%%% q(b, w)
        lb = lb + 0.5 * ((D + 1) * (safelog(2 * pi) + 1) + logdet(bw.covariance));
        %%%% q(t)
        alpha = lower - t.mean;
        beta = upper - t.mean;
        normalization = safenormcdf(beta) - safenormcdf(alpha);
        lb = lb + 0.5 * N * (safelog(2 * pi) + 1) + sum(safelog(normalization)) + ↩
            0.5 * sum((alpha .* safenormpdf(alpha) - beta .* safenormpdf(beta)) ./ ↩
            normalization) - 0.5 * sum((safenormpdf(alpha) - safenormpdf(beta)).^2 ↩
            ./ normalization.^2);

        bounds(iter) = lb;
    end

    state.lambda = lambda;
    state.psi = tau;
    state.bw = bw;
    state.bounds = bounds;
end

function prediction = grouped_probit_variational_test(X, state, parameters)
    rand('state', parameters.seed); %#ok<RAND>
    randn('state', parameters.seed); %#ok<RAND>

    N = size(X, 2);

    t.mean = [ones(1, N); X]' * state.bw.mean;
    t.covariance = diag([ones(1, N); X]' * state.bw.covariance * [ones(1, N); X]) + ↩
        1;

    prediction.p = 1 - safenormcdf(-t.mean ./ t.covariance);
end

function state = transfer_probit_variational_train(X, y, parameters)
    rand('state', parameters.seed); %#ok<RAND>
    randn('state', parameters.seed); %#ok<RAND>

    L = length(y);
    D = zeros(L, 1);
    N = zeros(L, 1);
```

```
for o = 1:L
    D(o) = size(X{o}, 1);
    N(o) = size(X{o}, 2);
end
R = parameters.dimension;

log2pi = safelog(2 * pi);

Phi = cell(1, L);
Q = cell(1, L);
Z = cell(1, L);
for o = 1:L
    Phi{o}.shape = (parameters.alpha_phi + 0.5) * ones(D(o), R);
    Phi{o}.scale = parameters.beta_phi * ones(D(o), R);
    Q{o}.mean = randn(D(o), R);
    Q{o}.covariance = repmat(eye(D(o), D(o)), [1, 1, R]);
    Z{o}.mean = randn(R, N(o));
    Z{o}.covariance = eye(R, R);
end
lambda.shape = (parameters.alpha_lambda + 0.5);
lambda.scale = parameters.beta_lambda;
tau.shape = (parameters.alpha_psi + 0.5) * ones(R, 1);
tau.scale = parameters.beta_psi * ones(R, 1);
bw.mean = randn(R + 1, 1);
bw.covariance = eye(R + 1, R + 1);
t = cell(1, L);
for o = 1:L
    t{o}.mean = zeros(N(o), 1);
end
for o = 1:L
    for i = 1:N(o)
        while 1
            t{o}.mean(i) = randn();
            if t{o}.mean(i) * y{o}(i) > 0
                break;
            end
        end
    end
end

XXT = cell(1, L);
phi_indices = cell(1, L);
for o = 1:L
    XXT{o} = X{o} * X{o}';
    phi_indices{o} = repmat(logical(eye(D(o), D(o))), [1, 1, R]);
end
lower = cell(1, L);
upper = cell(1, L);
```

```
for o = 1:L
    lower{o} = -1e40 * ones(N(o), 1);
    lower{o}(y{o} > 0) = 0;
    upper{o} = +1e40 * ones(N(o), 1);
    upper{o}(y{o} < 0) = 0;
end

bounds = zeros(parameters.iteration, 1);

for iter = 1:parameters.iteration
    if mod(iter, 10) == 0
        fprintf(1, '.');
    end
    if mod(iter, 100) == 0
        fprintf(1, ' %5d\n', iter);
    end

    %%%% update Phi
    for o = 1:L
        Phi{o}.scale = 1 ./ (1 / parameters.beta_phi + 0.5 * (Q{o}.mean.^2 + ↩
            reshape(Q{o}.covariance(phi_indices{o}), D(o), R)));
    end
    %%%% update Q
    for o = 1:L
        for s = 1:R
            Q{o}.covariance(:, :, s) = (XXT{o} + diag(Phi{o}.shape(:, s) .* Phi{↩
                o}.scale(:, s))) \ eye(D(o), D(o));
            Q{o}.mean(:, s) = Q{o}.covariance(:, :, s) * X{o} * Z{o}.mean(s, :)↩
                ';
        end
    end
    %%%% update Z
    for o = 1:L
        Z{o}.covariance = (eye(R, R) + bw.mean(2:R + 1) * bw.mean(2:R + 1)' + bw↩
            .covariance(2:R + 1, 2:R + 1)) \ eye(R, R);
        Z{o}.mean = Z{o}.covariance * (Q{o}.mean' * X{o} + bw.mean(2:end) * t{o↩
            }.mean' - repmat(bw.mean(2:R + 1) * bw.mean(1) + bw.covariance(2:R +↩
             1, 1), 1, N(o)));
    end
    %%%% update lambda
    lambda.scale = 1 / (1 / parameters.beta_lambda + 0.5 * (bw.mean(1)^2 + bw.↩
        covariance(1, 1)));
    %%%% update tau
    tau.scale = 1 ./ (1 / parameters.beta_psi + 0.5 * (bw.mean(2:R + 1).^2 + ↩
        diag(bw.covariance(2:R + 1, 2:R + 1))));
    %%%% update b and w
    Z1 = zeros(R, 1);
    ZZT = zeros(R, R);
```

```
Zt = zeros(R + 1, 1);
for o = 1:L
    Z1 = Z1 + Z{o}.mean * ones(N(o), 1);
    ZZT = ZZT + Z{o}.mean * Z{o}.mean' + N(o) * Z{o}.covariance;
    Zt = Zt + [ones(1, N(o)); Z{o}.mean] * t{o}.mean;
end
bw.covariance = [sum(N) + lambda.shape * lambda.scale, Z1'; Z1, ZZT + diag(↩
    tau.shape .* tau.scale)] \ eye(R + 1, R + 1);
bw.mean = bw.covariance * Zt;
%%%% update t
for o = 1:L
    t{o}.mean = [ones(1, N(o)); Z{o}.mean]' * bw.mean;
    t{o}.mean = t{o}.mean + (safenormpdf(lower{o} - t{o}.mean) - safenormpdf↩
        (upper{o} - t{o}.mean)) ./ (safenormcdf(upper{o} - t{o}.mean) - ↩
        safenormcdf(lower{o} - t{o}.mean));
end

lb = 0;
%%%% p(Phi_o)
for o = 1:L
    lb = lb + sum(sum((parameters.alpha_phi - 1) * (psi(Phi{o}.shape) + ↩
        safelog(Phi{o}.scale)) - Phi{o}.shape .* Phi{o}.scale / parameters.↩
        beta_phi - gammaln(parameters.alpha_phi) - parameters.alpha_phi * ↩
        safelog(parameters.beta_phi)));
end
%%%% p(Q_o | Phi_o)
qqT = cell(1, L);
for o = 1:L
    qqT{o}.mean = zeros(D(o), D(o), R);
    for s = 1:R
        qqT{o}.mean(:, :, s) = Q{o}.mean(:, s) * Q{o}.mean(:, s)' + Q{o}.↩
            covariance(:, :, s);
        lb = lb - 0.5 * sum(diag(diag(Phi{o}.shape(:, s) .* Phi{o}.scale(:, ↩
            s)) * qqT{o}.mean(:, :, s))) - 0.5 * (D(o) * log2pi - sum(↩
            safelog(Phi{o}.shape(:, s) .* Phi{o}.scale(:, s))));
    end
end
%%%% p(Z_o | Q_o, X)
ZZT = cell(1, L);
for o = 1:L
    ZZT{o}.mean = Z{o}.mean * Z{o}.mean' + N(o) * Z{o}.covariance;
    lb = lb - 0.5 * sum(diag(ZZT{o}.mean)) + sum(diag((X{o}' * Q{o}.mean) * ↩
        Z{o}.mean)) - 0.5 * sum(diag(sum(qqT{o}.mean, 3) * XXT{o})) - 0.5 * ↩
        N(o) * R * log2pi;
end
%%%% p(lambda)
lb = lb + (parameters.alpha_lambda - 1) * (psi(lambda.shape) + safelog(↩
    lambda.scale)) - lambda.shape * lambda.scale / parameters.beta_lambda - ↩
```

```
    gammaln(parameters.alpha_lambda) - parameters.alpha_lambda * safelog(↩
    parameters.beta_lambda);
%%%% p(b | lambda)
lb = lb - 0.5 * bw.mean(1).^2 * lambda.shape * lambda.scale - 0.5 * (log2pi ↩
    - safelog(lambda.shape * lambda.scale));
%%%% p(tau)
lb = lb + sum((parameters.alpha_psi - 1) * (psi(tau.shape) + safelog(tau.↩
    scale)) - tau.shape .* tau.scale / parameters.beta_psi - gammaln(↩
    parameters.alpha_psi) - parameters.alpha_psi * safelog(parameters.↩
    beta_psi));
%%%% p(w | tau)
lb = lb - 0.5 * bw.mean(2:R + 1)' * diag(tau.shape .* tau.scale) * bw.mean↩
    (2:R + 1) - 0.5 * (R * log2pi - sum(safelog(tau.shape .* tau.scale)));
%%%% p(t_o | b, w, Z_o) p(y_o | t_o)
wwT.mean = bw.mean(2:R + 1) * bw.mean(2:R + 1)' + bw.covariance(2:R + 1, 2:R↩
     + 1);
for o = 1:L
    lb = lb - 0.5 * (t{o}.mean' * t{o}.mean + N(o)) + sum(bw.mean(1) * t{o}.↩
        mean) + sum(diag(Z{o}.mean' * bw.mean(2:R + 1, :) * t{o}.mean')) - ↩
        0.5 * sum(diag(wwT.mean * ZZT{o}.mean)) - 0.5 * N(o) * (bw.mean(1)↩
        .^2 + bw.covariance(1, 1)) - sum(Z{o}.mean' * (bw.mean(2:R + 1) * bw↩
        .mean(1) + bw.covariance(2:R + 1, 1))) - 0.5 * N(o) * log2pi;
end

%%%% q(Phi_o)
for o = 1:L
    lb = lb + sum(sum(Phi{o}.shape + safelog(Phi{o}.scale) + gammaln(Phi{o}.↩
        shape) + (1 - Phi{o}.shape) .* psi(Phi{o}.shape)));
end
%%%% q(Q_o)
for o = 1:L
    for s = 1:R
        lb = lb + 0.5 * (D(o) * (log2pi + 1) + logdet(Q{o}.covariance(:, :, ↩
            s)));
    end
end
%%%% q(Z_o)
for o = 1:L
    lb = lb + 0.5 * N(o) * (R * (log2pi + 1) + logdet(Z{o}.covariance));
end
%%%% q(lambda)
lb = lb + lambda.shape + safelog(lambda.scale) + gammaln(lambda.shape) + (1 ↩
    - lambda.shape) * psi(lambda.shape);
%%%% q(tau)
lb = lb + sum(tau.shape + safelog(tau.scale) + gammaln(tau.shape) + (1 - tau↩
    .shape) .* psi(tau.shape));
%%%% q(b, w)
lb = lb + 0.5 * ((R + 1) * (log2pi + 1) + logdet(bw.covariance));
```

```
        %%%% q(t_o)
        for o = 1:L
            alpha = lower{o} - t{o}.mean;
            beta = upper{o} - t{o}.mean;
            normalization = safenormcdf(beta) - safenormcdf(alpha);
            lb = lb + 0.5 * N(o) * (log2pi + 1) + sum(safelog(normalization)) + 0.5 ↩
                * sum((alpha .* safenormpdf(alpha) - beta .* safenormpdf(beta)) ./ ↩
                normalization) - 0.5 * sum((safenormpdf(alpha) - safenormpdf(beta))↩
                .^2 ./ normalization.^2);
        end

        bounds(iter) = lb;
    end

    state.Phi = Phi;
    state.Q = Q;
    state.lambda = lambda;
    state.psi = tau;
    state.bw = bw;
    state.bounds = bounds;
end

function prediction = transfer_probit_variational_test(X, state, parameters)
    rand('state', parameters.seed); %#ok<RAND>
    randn('state', parameters.seed); %#ok<RAND>

    L = length(X);
    R = size(state.bw.mean, 1) - 1;
    N = zeros(L, 1);
    for o = 1:L
        N(o) = size(X{o}, 2);
    end

    for o = 1:L
        prediction.Z{o}.mean = zeros(R, N(o));
        prediction.Z{o}.covariance = zeros(R, N(o));
        for s = 1:R
            prediction.Z{o}.mean(s, :) = state.Q{o}.mean(:, s)' * X{o};
            prediction.Z{o}.covariance(s, :) = diag(X{o}' * state.Q{o}.covariance(:,↩
                :, s) * X{o}) + 1;
        end
    end

    prediction.p = cell(1, L);
    for o = 1:L
        t.mean = [ones(1, N(o)); prediction.Z{o}.mean]' * state.bw.mean;
        t.covariance = diag([ones(1, N(o)); prediction.Z{o}.mean]' * state.bw.↩
            covariance * [ones(1, N(o)); prediction.Z{o}.mean]) + 1;
```

```
        prediction.p{o} = 1 - safenormcdf(-t.mean ./ t.covariance);
    end
end

function c = safenormpdf(x)
    x(x < -40) = -40;
    x(x > +40) = +40;
    c = normpdf(x);
end

function c = safenormcdf(x)
    x(x < -40) = -40;
    c = normcdf(x);
end

function y = safelog(x)
    x(x < 1e-323) = 1e-323;
    x(x > 1e+308) = 1e+308;
    y = log(x);
end

function ld = logdet(Sigma)
    U = chol(Sigma);
    ld = 2 * sum(log(diag(U)));
end
```

REFERENCES

Abdou, H., J. Pointon, and A. El-Masry, 2008, "Neural Nets versus Conventional Techniques in Credit Scoring in Egyptian Banking", *Expert Systems with Applications*, Vol. 35, pp. 1275–1292.

Albert, J. H. and S. Chib, 1993, "Bayesian Analysis of Binary and Polychotomous Response Data", *Journal of the American Statistical Association*, Vol. 88, pp. 669–679.

Alpaydın, E., 2010, *Introduction to Machine Learning*, The MIT Press, Cambridge, MA, USA.

Angelini, E., G. Di Tollo, and A. Roli, 2008, "A Neural Network Approach for Credit Risk Evaluation", *The Quarterly Review of Economics and Finance*, Vol. 48, pp. 733–755.

Atiya, A. F., 2001, "Bankruptcy Prediction for Credit Risk Using Neural Networks: A Survey and New Results", *IEEE Transactions on Neural Networks*, Vol. 12, pp. 929–935.

Bach, F. R., 2008, "Consistency of the Group Lasso and Multiple Kernel Learning", *Journal of Machine Learning Research*, Vol. 9, pp. 1179–1225.

Basel I, 1988, *Basel Committee: International Convergence of Capital Measurement and Capital Standards*, Basel Committee of Banking Supervision, Bank for International Settlements.

Basel II, 2004, *International Convergence of Capital Measurement and Capital Standards: A Revised Framework*, Basel Committee of Banking Supervision, Bank for International Settlements.

Basel III, 2011, *A Global Regulatory Framework for More Resilient Banks and Banking Systems*, Basel Committee on Banking Supervision, Bank for International Settlements.

Beal, M. J., 2003, *Variational Algorithms for Approximate Bayesian Inference*, Ph.D. thesis, The Gatsby Computational Neuroscience Unit, University College London.

Brill, J., 1998, "The Importance of Credit Scoring Models in Improving Cash Flow and Collection", *Business Credit*, Vol. 100, pp. 16–17.

Chen, W.-H. and J.-Y. Shih, 2006, "A Study of Taiwan's Issuer Credit Rating Systems Using Support Vector Machines", *Expert Systems with Applications*, Vol. 30, pp. 427–435.

Derelioğlu, G. and F. Gürgen, 2011, "Knowledge Discovery Using Neural Approach for SME's Credit Risk Analysis Problem in Turkey", *Expert Systems with Applications*, Vol. 38, pp. 9313–9318.

Finlay, S., 2009, "Are We Modelling the Right Thing? The Impact of Incorrect Problem Specification in Credit Scoring", *Expert Systems with Applications*, Vol. 36, pp. 9065–9071.

Gelfand, A. E. and A. F. M. Smith, 1990, "Sampling-Based Approaches to Calculating Marginal Densities", *Journal of the American Statistical Association*, Vol. 85, pp. 398–409.

Gönen, M. and E. Alpaydın, 2011, "Multiple Kernel Learning Algorithms", *Journal of Machine Learning Research*, Vol. 12, pp. 2211–2268.

Hsieh, N.-C. and L.-P. Hung, 2010, "A Data Driven Ensemble Classifier for Credit Scoring Analysis", *Expert Systems with Applications*, Vol. 37, pp. 534–545.

Huang, C.-L., M.-C. Chen, and C.-J. Wang, 2007, "Credit Scoring with a Data Mining Approach Based on Support Vector Machines", *Expert Systems with Applications*, Vol. 33, pp. 847–856.

Huang, Y.-M., C.-M. Hung, and H. C. Jiau, 2006, "Evaluation of Neural Networks and Data Mining Methods on a Credit Assessment Task for Class Imbalance Problem", *Nonlinear Analysis: Real World Applications*, Vol. 7, pp. 720–747.

Huang, Z., H. Chen, C.-J. Hsu, W.-H. Chen, and S. Wu, 2004, "Credit Rating Analysis with Support Vector Machines and Neural Networks: A Market Comparative Study", *Decision Support Systems*, Vol. 37, pp. 543–558.

Khashman, A., 2010, "Neural Networks for Credit Risk Evaluation: Investigation of Different Neural Models and Learning Schemes", *Expert Systems with Applications*, Vol. 37, pp. 6233–6239.

Kim, H. S. and S. Y. Sohn, 2010, "Support Vector Machines for Default Prediction of SMEs Based on Technology Credit", *European Journal of Operational Research*, Vol. 201, pp. 838–846.

Kloft, M., U. Brefeld, S. Sonnenburg, and A. Zien, 2011, "ℓ_p-Norm Multiple Kernel Learning", *Journal of Machine Learning Research*, Vol. 12, pp. 953–997.

Kotsiantis, S., D. Tzelepis, E. Koumanakos, and V. Tampakas, 2005, "Efficiency of Machine Learning Techniques in Bankruptcy Prediction", *Proceedings of 2nd International Conference on Enterprise Systems and Accounting.*

Lai, K. K., L. Yu, S. Wang, and L. Zhou, 2006a, "Credit Risk Analysis Using a Reliability-Based Neural Network Ensemble Model", *Proceedings of the 16th International Conference on Artificial Neural Networks.*

Lai, K. K., L. Yu, S. Wang, and L. Zhou, 2006b, "Neural Network Metalearning for Credit Scoring", *Proceedings of the 2006 International Conference on Intelligent Computing*.

Li, J., Z. Chen, and W. Xu, 2006, "Credit Assessment: A Least Squares Support Feature Machine Approach", *Proceedings of the 6th International Conference on Data Mining - Workshops*.

Li, J., L. Wei, G. Li, and W. Xu, 2011, "An Evolution Strategy-Based Multiple Kernels Multi-Criteria Programming Approach: The Case of Credit Decision Making", *Decision Support Systems*, Vol. 51, pp. 292–298.

Lodhi, H., C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, 2002, "Text Classification Using String Kernels", *Journal of Machine Learning Research*, Vol. 2, pp. 419–444.

Min, J. H. and C. Jeong, 2009, "A Binary Classification Method for Bankruptcy Prediction", *Expert Systems with Applications*, Vol. 36, pp. 5256–5263.

Noble, W. S., 2004, "Support Vector Machine Applications in Computational Biology", Schölkopf, B., K. Tsuda, and J.-P. Vert (eds.), *Kernel Methods in Computational Biology*, chap. 3, The MIT Press, Cambridge, MA, USA.

Peng, Y., G. Kou, Y. Shi, and Z. Chen, 2008, "A Multi-Criteria Convex Quadratic Programming Model for Credit Data Analysis", *Decision Support Systems*, Vol. 44, pp. 1016–1030.

Peng, Y., G. Wang, G. Kou, and Y. Shi, 2011, "An Empirical Study of Classification Algorithm Evaluation for Financial Risk Prediction", *Applied Soft Computing*, Vol. 11, pp. 2906–2915.

Schölkopf, B., K. Tsuda, and J.-P. Vert (eds.), 2004, *Kernel Methods in Computational Biology*, The MIT Press, Cambridge, MA, USA.

Shi, Y., 2010, "Multiple Criteria Optimization-Based Data Mining Methods and Applications: A Systematic Survey", *Knowledge and Information System*, Vol. 24, pp. 369–391.

Thomas, L. C., 2000, "A Survey of Credit and Behavioural Scoring: Forecasting Financial Risk of Lending to Consumers", *International Journal of Forecasting*, Vol. 16, pp. 149–172.

Twala, B., 2010, "Multiple Classifier Application to Credit Risk Assessment", *Expert Systems with Applications*, Vol. 37, pp. 3326–3336.

Van Gestel, T., B. Baesens, J. A. K. Suykens, D. Van den Poel, D.-E. Baestaens, and M. Willekens, 2006, "Bayesian Kernel Based Classification for Financial Distress Detection", *European Journal of Operational Research*, Vol. 172, pp. 979–1003.

Vapnik, V., 1998, *The Nature of Statistical Learning Theory*, John Wiley & Sons, New York, NY, USA.

Xu, Z., R. Jin, H. Yang, I. King, and M. R. Lyu, 2010, "Simple and Efficient Multiple Kernel Learning by Group Lasso", *Proceedings of the 27th International Conference on Machine Learning*.

Yao, P., C. Wu, and M. Yang, 2009, "Credit Risk Assessment Model of Commercial Banks Based on Fuzzy Neural Network", *Proceedings of the 6th International Symposium on Neural Networks*.

Yongqiao, W., W. Shouyang, and K. K. Lai, 2005, "A New Fuzzy Support Vector Machine to Evaluate Credit Risk", *IEEE Transactions on Fuzzy Systems*, Vol. 13, pp. 820–831.

Yoon, J. S. and Y. S. Kwon, 2010, "A Practical Approach to Bankruptcy Prediction for Small Businesses: Substituting the Unavailable Financial Data for Credit Card Sales Information", *Expert Systems with Applications*, Vol. 37, pp. 3624–3629.

Zhou, L., K. K. Lai, and L. Yu, 2010, "Least Squares Support Vector Machines Ensemble Models for Credit Scoring", *Expert Systems with Applications*, Vol. 37, pp. 127–133.