Northeastern University
Khoury College of
Computer Sciences

PhD Thesis Approval

Thesis Title: Reduction Methods for Multi-label Classification

Author: Cheng Li

PhD Thesis Approval to complete all degree requirements for the PhD in Computer Science.

Thesis Advisor	9/30/2019 *Date*
Thesis Reader	9/10/2019 *Date*
Thesis Reader	9/10/2019 *Date*
Thesis Reader	9/10/2019 *Date*
Thesis Reader	9/30/2019 *Date*

KHOURY COLLEGE APPROVAL:

Associate Dean for Graduate Programs	10/1/19 *Date*

COPY RECEIVED BY GRADUATE STUDENT SERVICES:

Recipient's Signature	10/15/19 *Date*

Distribution: *Once completed, this form should be scanned and attached to the front of the electronic dissertation document (page 1). An electronic version of the document can then be uploaded to the Northeastern University-UMI Website.*

Page left intentionally blank.

Reduction Methods for Multi-label Classification

A dissertation presented

by

Cheng Li

to

The Khoury College of Computer Sciences

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in the subject of

Computer Science

Northeastern University

Boston, Massachusetts

Committee in charge:

Professor Javed Aslam, Chair
Professor Jennifer Dy
Assistant Professor Jonathan Ullman
Associate Teaching Professor Virgil Pavlu
Assistant Professor Krzysztof Dembczyński

September 2019

Reduction Methods for Multi-label Classification

Abstract

Multi-label classification is an important machine learning task wherein one predicts a set of labels to associate with a given object. For example, an article can belong to multiple categories; an image can be associated with several tags; in medical billing, a patient report is annotated with multiple diagnosis codes. The most commonly used approach, called binary relevance (BR), trains one binary classifier to predict each label separately. BR ignores label dependencies and often makes conflicting predictions, such as tagging `cat` but not `animal` for an image. How to learn label dependencies from data and train classifiers to account for such dependencies is the central question in multi-label classification.

This thesis describes two new approaches to multi-label classification which leverage label dependencies and achieve better classification accuracy than BR and many other sophisticated methods. The first approach, called conditional Bernoulli mixture (CBM), directly estimates the joint probability distribution among all labels with a mixture model. CBM's special model structure allows for efficient training, joint inference and marginal inference procedures designed to optimize different metrics.

The second approach, named BR-rerank, seeks to improve both BR's confidence estimation and prediction through post calibration and reranking procedures. BR-rerank takes the BR predicted set of labels and its product score as features, extracts more features from the prediction itself to capture label constraints, and applies Gradient Boosted Trees (GB) as a calibrator to map these features into a calibrated confidence score. The GB calibrator not only produces well-calibrated scores (*aligned with accuracy* and *sharp*), but also models label interactions, correcting a critical flaw in BR. Experiment results show that reranking label sets by the new calibrated confidence makes accurate set predictions on par with state-of-the-art multi-label classifiers—yet calibrated, simpler, and faster.

Both CBM and BR-rerank are reduction methods: they transform a complex multi-label classification problem to a series of standard binary classification, multi-class classification and regression problems, which are easier to solve.

Contents

List of Figures

List of Tables

Acknowledgments

The work presented in this thesis would not have been possible without the help and support of many people.

First and foremost, I am tremendously grateful for my adviser Javed Aslam for his continuous support and guidance throughout my Ph.D. study. In the early years of my study, by taking his courses, I learned a lot from him, especially his intuitive way of explaining things and his pursuit of mathematical elegance. Throughout my entire Ph.D. journey, I have always had a lot of fun in our collaborations. Sometimes, we thought about difficult problems together in his office. At other times, he gave me the freedom to pursue my own interests, though, always being there to support and advise. As I am now joining the startup company he co-founded, I am looking forward to working with him again and tackling technical challenges together.

I am also really grateful for Virgil Pavlu, who was like a second advisor to me. Over the past few years, I literally grew up under Virgil's guidance and supervision. He spent so many days with me discussing technical details and debugging programs in his house, and so many sleepless nights with me editing papers before conference deadlines. He always does the best things for his students. I truly respect him.

I want to thank the rest of my thesis committee members, Krzysztof Dembczyński, Jennifer Dy, and Jonathan Ullman. Special thanks to Krzysztof, who read my thesis carefully and gave me detailed and constructive comments even when he was busy moving to a new country.

I want to thank my labmates Bingyu Wang, Kechen Qin, Yuyu Xu, Pavel Metrikov, Jesse Anderton, and Maryam Aziz. I have had tremendous joy in all the conversations we had and all the whiteboards that we filled together.

I feel incredibly fortunate to have had the opportunity to deploy, test, and improve my algorithms in a real production environment by working with people from CodaMetrix. I want to thank my company colleagues Hamid Tabatabaie, Kevin Schmitt, Jeff Woodward, Phill Mell-Davies, Lisa Boyle, Amir Tahmasebi, Afshin Moshrefi, Ethan Morgan, and Zhenming Bi, among others. I also want to thank Michael Mercurio from CodaMetrix and MGPO for providing several years' of research funding.

Finally, I would like to thank my grandparents, my parents, my brother, and my wife, for always being supportive.

Chapter 1

Introduction

1.1 Multi-label Classification Problem

Many machine learning applications require solving the problem of identifying which category (categories) a given instance belongs to. For example, in spam detection, one decides whether an email is a spam email or not. In handwritten digit recognition, one recognizes a handwritten digit as one of $\{0, 1, ..., 9\}$. In image tagging, one finds all the relevant tags, such as `dog`, `animal` and `person`, to describe objects in a given image.

All these problems are classification problems. In a classification problem, there is a list of predefined candidate categories, and we need to choose the proper category (categories) for the given instance. Depending on the number of candidate categories provided and the number of categories one is allowed choose, classification problems can be divided into three different types: binary classification, multi-class classification and multi-label classification.

The simplest form of classification is *binary classification*, which chooses one of the two given categories. For example, in spam detection, each email is an instance to be classified, and the two possible categories are `spam` and `not spam`. Another classical example of binary classification is to determine whether a given patient has certain disease.

Many classification problems involve more than two candidate categories. For example, handwritten digit recognition involves 10 possible categories $\{0, 1, 2, ..., 9\}$. Such problems are formulated as *multi-class classification* which selects one category out of multiple possible categories.

☐ airport
☐ animal
☑ clouds
☐ book
☑ lake
☑ sunset
☑ sky
☐ cars
☑ water
☑ reflection
☐ bear
☐ buildings
☐ castle
...

Figure 1.1: A tagged image from the NUSWIDE dataset [25]. Tags associated with this image = {clouds, lake, sunset, sky, water, reflection}. All candidate tags in the NUSWIDE dataset = {airport, animal, beach, bear, birds, boats, book, bridge, buildings, cars, castle, cat, cityscape, clouds, computer, coral, cow, dancing, dog, earthquake, elk, fire, fish, flags, flowers, food, fox, frost, garden, glacier, grass, harbor, horses, house, lake, leaf, map, military, moon, mountain, nighttime, ocean, person, plane, plants, police, protest, railroad, rainbow, reflection, road, rocks, running, sand, sign, sky, snow, soccer, sports, statue, street, sun, sunset, surf, swimmers, tattoo, temple, tiger, tower, town, toy, train, tree, valley, vehicle, water, waterfall, wedding, whales, window, zebra}.

There are also problems which not only define multiple candidate categories, but also allow each instance to be associated to multiple categories at the same time. For example, an image can contain multiple objects and can have multiple tags. The image shown in Figure 1.1 is tagged with clouds, lake, sunset, sky, water, reflection. These tags are a subset of the 81 candidate tags defined in NUSWIDE image tagging dataset [25]. Similarly, a movie can belong to multiple genres. The movie "Harry Potter and the Sorcerer's Stone" appears under the genres Adventure, Family, and Fantasy on the IMDB website, among the 28 different genres defined by IMDB (see Figure 1.2). Such classification tasks are called *multi-label classification*, which associates a given instance to multiple categories, chosen from the list of candidate categories. Generally speaking, there is no restriction on how many

Figure 1.2: A screenshot of the IMDB webpage for the movie "Harry Potter and the Sorcerer's Stone". Genre tags associated with this movie = {`Adventure, Family, Fantasy`}. All genre tags available on IMDB = {`Drama, Comedy, Romance, Thriller, Crime, Action, Horror, Adventure, Documentary, Mystery, Sci-Fi, Fantasy, Family, Biography, War, Animation, History, Music, Musical, Western, Short, Sport, Film-Noir, News, Adult, Talk-Show, Game-Show, Reality-TV`}. From `https://www.imdb.com/title/tt0241527/`. Screenshot by author.

categories an instance can match. An instance can potentially match none of the given categories or all of them (although this is rare in practice). Besides image tagging and movie genre classification, multi-label classification also has many other applications, such as Reuters news categorization, ICD/CPT medical billing, Youtube video tagging, and patent classification (Figure 1.3).

Conceptually, multi-class classification is a generalization of binary classification. Multi-label classification is a further generalization of multi-class classification. Table 1.1 summarizes these three types of classifications.

Table 1.1: Binary vs. multi-class vs. multi-label classifications.

task	description
binary classification	choose 1 label out of 2 candidate classes
multi-class classification	choose 1 label out of many candidate classes
multi-label classification	choose many labels out of many candidate classes

Figure 1.3: Applications of multi-label classification.

Formally, in a classification problem, we are given a finite list of predefined categories (or class or tags or labels) $\mathcal{L}$. These categories are typically encoded as integers $\mathcal{L} = \{1, 2, ..., L\}$. The number of candidate categories $|\mathcal{L}|$ is 2 for binary classification, and could be bigger than 2 for multi-class classification and multi-label classification. Each input instance is represented as a D dimensional feature vector $\mathbf{x} = (x_1, x_2, ..., x_D) \in \mathcal{R}^D$, with each dimension measuring certain aspect of the instance. In binary and multi-class classification, we aim to build a classifier h that maps each $\mathbf{x}$ to a single category $y \in \mathcal{L}$. In multi-label classification, we aim to build a classifier h that maps each $\mathbf{x}$ to a set of categories $\mathbf{y} \subset \mathcal{L}$. The classification output can also be written as a binary vector with each bit indicating the presence or absence of the corresponding label. For binary and multi-class classification, only one bit will be 1 and all other bits will be 0, and such a binary vector is sometimes called a one-hot encoding vector. For multi-label classification, multiple bits could be 1. For example, in a multi-class classification problem with 10 candidate categories, if the third category is assigned to an instance, then the output can be written as $y = 3$ or $y = (0, 0, 1, 0, 0, 0, 0, 0, 0, 0)$. In a multi-label classification problem with 10 candidate categories, if the first, the third, and the eighth labels are assigned to an

instance, the output can be written as $\mathbf{y} = \{1, 3, 8\}$ or $\mathbf{y} = (1, 0, 1, 0, 0, 0, 0, 1, 0, 0)$. Sometimes it is more convenient to work with the integer notation and sometimes it is more convenient to work with the binary vector notation. In this thesis, we will switch between these two notations and use whichever more convenient in the context.

This thesis focuses on multi-label classification, which has many important applications and unique algorithmic challenges, as we shall describe in the next section.

1.2 Challenges in Multi-label Classification

Compared to binary and multi-class problems, multi-label problems have some unique challenges. As a specific motivating example, consider the problem of medical billing and reporting where all patient symptoms, diagnoses, and procedures recorded in conjunction with hospital care are encoded with a set of CPT codes (for procedures) and a set of ICD codes (for symptoms and diagnoses).[1] The use of such codes is mandated for medical reporting and billing in the US and beyond. From the standpoint of multi-label prediction, the size of these code sets is extremely large and growing; for example, there are over 69,000 ICD-10 diagnosis codes in current use as compared to about 14,000 such codes in ICD-9. Most hospitals employ dedicated human medical coders to parse medical records manually and assign codes. Human annotation is inherently prone to error, and this is especially true when confronted with tens-of-thousands of potential codes. It is reported that nearly 50% of the manually billed medical records have some coding error (especially missing codes), and this high error rate has caused a 26% increase of total cost for patients and health care providers [1]. Developing an automated medical billing and reporting system to make these processes more efficient and accurate has immediate benefit for the whole society. From a machine learning point of view, the task is to build a multi-label classifier in order to predict the diagnosis/procedure codes for a patient visit, given the visit notes and certain attributes like the provider, care facility and patient's general information.

We have observed several challenges and requirements in building such a multi-label classification system for medical billing and reporting during our collaborating with Partners Healthcare and Massachusetts General Hospital, and

[1] CPT stands for "Current Procedural Terminology" and ICD stands for "International Classification of Diseases".

most of these challenges and requirements also arise in other multi-label classification tasks, such as Flickr image tagging, Wikipedia categorization and YouTube video classification.

Label dependencies. There are constraints among codes. For example, in medical coding, the codes `M25.562` ("Pain in left knee") and `M25.569` ("Pain in unspecified knee") should not be used together. In image tagging, an imaged tagged with `cat` should also be tagged with `animal`. One major challenge in multi-label classification is that predicting each label independently (called binary relevance (BR) [107]) does not work well. Certain labels are hard to predict directly; also individual label predictions can often be conflicting. It is known that exploiting label dependencies improves the overall performance [32], and most of the progress in multi-label classification accounts for label dependencies. In the absence of any prior information about label relations (domain knowledge, ontology, etc.) the challenge is to *learn* the label dependencies from data while optimally *training* classifiers to account for such dependencies, a daunting task if the label set is large enough so that the possible label subsets are in the millions or billions or even larger.

Large scale data. Datasets with large numbers of instances/features/labels are becoming increasingly common. Many of the research datasets have hundreds of labels and hundreds of thousands of instances. The data that industrial companies are dealing with are even larger: The Open Image dataset from Google [2] contains 9 million images and 5,000 labels. The YouTube-8M dataset [3] contains 8 million videos, 1024 frame level features, 1,152 video level features, and 4,716 labels. The WikiLSHTC-325K dataset [4] contains 2 million Wikipedia pages, 1,617,899 unigram features and 325,056 labels. How to train a multi-label classifier efficiently on large scale data while capturing label dependencies is a challenge. For example, the pair-wise CRF model [44] considers pair-wise label interactions and has a complexity quadratic in number of labels. Such methods may have difficulties scaling to large datasets with many labels. Also, storing the trained models may take a lot of disk space. Techniques that compress model structure without losing accuracy are very useful.

Inference under task metric. Different multi-label applications often require different forms of predictions and employ different evaluation metrics. Commonly used evaluation metrics include *subset accuracy*, *F1 score*, *Jaccard index*, *Hamming loss*, *Mean Average Precision* and *NDCG*. The theoretical results in [32, 61] show that a multi-label classifier designed for optimizing one measure may be suboptimal under other measures. How to achieve the optimal inference under Hamming loss, subset accuracy, instance/micro/macro averaged F1 has been studied in [32, 61]. It is shown that the optimal inference for the subset accuracy measure and the instance

averaged F1 measure requires the classifier to maintain a joint probability estimation among all labels; the optimal inference for Hamming loss and micro/macro averaged F1 requires the classifier to provide marginal label probability estimations. To support all these inference procedures, a multi-label model should be versatile. The classifier model should have some carefully designed structure that allows for efficient joint and marginal inferences.

Calibrated confidence. The ability to produce a calibrated confidence associated with the prediction is a natural requirement in many industrial applications, but is often overlooked by algorithm designers in academia since it does not necessarily affect raw testing performance. The confidence indicates the likelihood of the prediction $\mathbf{y}$ being correct. It is called "calibrated" if the confidence aligns with the empirical accuracy: for all the predictions with confidence around, say, 60%, roughly 60% of them should be correct. In medical billing, each medical note is tagged with multiple billing codes and these billing codes are sent off to insurance company. The hospital annotation must reach a required accuracy, say, 90% to prevent additional checks or lawsuits from the insurer. A multi-label classifier may only have 70% accuracy on this task so it cannot alone send predicted billing codes. But the classifier is still quite helpful if its reported confidence with each prediction is well calibrated: send off automatically the prediction with confidence greater than 90%, and ask for human intervention on lower-confidence predictions. This work-flow can significantly reduce billing annotation costs, but only works well if the confidence is calibrated. Note that the confidence score is defined for the whole predicted label set $\mathbf{y}$, as opposed to individual labels $y_1, y_2, ..., y_L$. Because operationally, asking a human medical expert to verify one label takes roughly the same time as verifying a set of labels (it is reading the medical document that takes most time). So there is a natural incentive to automate the entire set prediction if it is confident enough, rather than some individual label predictions.

Label noise. Another difficulty is that multi-label data often exhibits high levels of noise [86, 78, 16, 125, 131]. Most of the data we work with (such as patient records) are annotated by humans, so mistakes are inevitable. There are typically hundreds or thousands of candidate labels and the number of relevant labels for an instance is unknown, thus it is hard for human annotators to go through the list and identify all relevant labels. Minor mistakes in a multi-label setup are more likely than in a multi-class setup where each data object has precisely one label. This labeling noise poses significant challenges for multi-label training and confidence calibration.

A successful multi-label classification system should deal with all the above mentioned challenges. Such a system should: (1) respect the dependencies among the labels; (2) trains efficiently on large scale data; (3) make various forms of predictions

depending on the task metric and scenario; (4) robust to the noise in annotations; (5) have some well calibrated confidence scores associated the predictions.

1.3 Related Work

There are many existing works on multi-label classification. In this section, we review some representitive ones and we divide them into three topics: 1) how to design the classifier structure to capture label dependencies; 2) how to deal with large scale data; 3) how to deal with label noise.

1.3.1 Label Dependencies and Classifier Design

It is known that exploiting label dependencies improves the overall performance [32], and most of the progress in multi-label classification accounts for label dependencies. Here we review a few representative approaches, and discuss their advantages and disadvantages.

Binary Relevance. The simplest approach is to apply one binary classifier (e.g., binary logistic regression or support vector machine) to predict each label separately. This approach is called *binary relevance* (BR) [107] and is widely used due to its simplicity and speed. BR's training time grows linearly with the number of labels, which is considerably lower than many methods that seek to model label dependencies, and this makes BR run reasonably fast on commonly used datasets. (Admittedly, BR may still fail to scale to datasets with extremely large number of labels, in which case specially designed multi-label classifiers with sub-linear time complexity should be employed instead. But in this thesis, we shall not consider such *extreme* multi-label classification problem.)

BR has two well-known drawbacks. First, BR neglects label dependencies and this often leads to prediction errors: some BR predictions are incomplete, such as tagging `cat` but not `animal` for an image, and some are conflicting, such as predicting both the code `Pain in left knee` and the code `Pain in unspecified knee` for a medical note. Second, the confidence score or probability (we shall use "confidence score" and "probability" interchangeably) BR associates to its overall set prediction $\mathbf{y}$ is often misleading, or uncalibrated. BR computes the overall set prediction confidence score as the product of the individual label confidence scores, i.e., $p(\mathbf{y}|\mathbf{x}) = \prod_{l=1}^{L} p(y_l|\mathbf{x})$. This overall confidence score often does not reflect reality: among all the set predictions on which BR claims to have roughly 80% confidence,

maybe only 60% of them are actually correct (a predicted set is considered "correct" if it matches the ground truth set exactly). Having such uncalibrated prediction confidence makes it hard to integrate BR directly into a decision making pipeline where not only the predictions but also the confidence scores are used in downstream tasks.

Power-Set. At the other extreme is the *Power-Set* (PowSet) approach [107], which treats each label subset as a class, and trains a multi-class classifier. As a consequence, one would be restricted in practice to predicting only label subsets seen in the training data and always assigning zero probabilities to unseen subsets; even for many of the subsets observed there would likely be a scarcity of training data. Power-Set is handling label dependencies and its predictions are coherent, but the method is often infeasible on the exponential number of label sets. The Random k-label-sets (RAKEL) [108] is proposed to reduce the computational cost associated with Power-Set method. It trains K multi-class classifiers, each on a random subset of labels and uses all K models to make the final predictions.

Classifier Chains. Classifier Chains (CC) [97] decomposes the joint probability $p(\mathbf{y}|\mathbf{x})$ into a product of conditionals $p(y_1|\mathbf{x})p(y_2|\mathbf{x}, y_1) \cdots p(y_L|\mathbf{x}, y_1, .., y_{L-1})$, based on the chain rule. This reduces a multi-label learning problem to L binary learning problems, each of which learns a new label given all previous labels. During prediction, finding the exact joint mode is intractable. Classifier Chains classify labels greedily in a sequence: label y_ℓ is decided by maximizing $p(y_\ell|\mathbf{x}, y_1, .., y_{\ell-1})$, and becomes a feature to be used in the prediction for label $y_{\ell+1}$. This greedy prediction procedure has several issues: 1) the chain order must be decided in advance, 2) the predicted subset can be far away from the joint mode [33]; 3) errors in early label predictions propagate to subsequent label predictions; 4) the overall prediction depends on the chain order. To address the first two issues, *Probabilistic Classifier Chains* (PCC) replace the greedy search strategy with some more accurate search strategies, such as exhaustive search [24], ϵ-approximate search [34], Beam Search [64, 65], or A* search [77]. To address the last issue, Ensemble of Classifier Chains (ECC) [97] averages several predictions made by different chains, wherein the averaging can take place at either the individual label level (ECC-label) or the label subset level (ECC-subset). Using dynamic programming to find the optimal chain order [73] has been proposed recently.

Conditional Dependency Network. A similar reduction method named *Conditional Dependency Networks* (CDN) estimates $p(\mathbf{y}|\mathbf{x})$ based on full conditionals and Gibbs sampling [50]. During learning, one binary classifier is trained for each full conditional $p(y_\ell|\mathbf{x}, y_1, .., y_{\ell-1}, y_{\ell+1}, ..., y_L)$. During prediction, Gibbs sampling is used to find the mode of the joint. The method's major limitation

is that it cannot handle perfectly correlated or anti-correlated labels: consider binary classification as multi-label problem with only 2 exhaustive and exclusive labels. A perfect model for $p(y_1 = 1|\mathbf{x}, y_2)$ is $1 - y_2$; in other words, the feature information is completely ignored. The same applies to $p(y_2|\mathbf{x}, y_1)$. So the prediction will inevitably fail. In general, Gibbs sampling may fail in the presence of perfect correlations or anti-correlations since the resulting stochastic process is not ergodic; when relations are imperfect but very strong, Gibbs sampling may need a very long time to converge.

Conditional Random Fields. *Conditional Random Fields* (CRF) [102] offers a general framework for structured prediction problems based on undirected graphical models. In multi-label classification, labels can form densely connected graphs, so restrictions are imposed in order to make training and inference tractable. The pair-wise conditional random field (CRF) [44] uses pair-wise potential functions to model interactions between pairs of labels. It defines a log-linear model in the form

$$
\begin{aligned}
p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\{\sum_{l=1}^{L}\sum_{d=1}^{D} w_{ld} x_d \mathbb{1}[y_l = 1] \\
+ \sum_{l=1}^{L}\sum_{m=1}^{L} (w_{lm1}\mathbb{1}[y_l = 0, y_m = 0] + w_{lm2}\mathbb{1}[y_l = 0, y_m = 1] \\
+ w_{lm3}\mathbb{1}[y_l = 1, y_m = 0] + w_{lm4}\mathbb{1}[y_l = 1, y_m = 1])\}
\end{aligned}
$$

where $Z(\mathbf{x})$ is the normalization constant summing over all possible $\mathbf{y} \in \{0, 1\}^L$, and all w are learned weights. The CRF model has well known limitations. First since the model assigns one dedicated parameter to each label interaction term, modeling higher order label dependencies requires too many parameters, usually infeasible. Second, the linear formulation in CRF makes it hard to incorporate features which do not linearly correlate with the prediction target. One such critical feature is the cardinality of the predicted label set $\mathbf{y}$: for most of the datasets, predicting a label set of size 2 to 5 is more likely to be correct than a set of size 0 or 20 (even if the number of labels in tens of thousands, like the medical ICD10 diagnosis codes, a report typically has 2 or 3 such diagnosis). To incorporate such a feature, the model has to be non-linear. The third limitation of CRF is that both computing the partition function $Z(\mathbf{x})$ and making predictions require handling an exponentially large output space. Proposed solutions include *support inference* and binary pruned inference [44]. Support inference restricts $\mathbf{y}$ sets only to those observed in training. A consequence is that no new label sets can ever be predicted. From a probabilistic perspective, support inference imposes a very harsh 0/1 prior over label sets: all unobserved sets have zero prior and all observed sets have uniform non-zero prior.

Apart from the pair-wise CRF [44], there is another CRF model designed

specifically to capture exclusive or hierarchical label relations [35]; this works only when label dependencies are *strict* and *a priori* known.

Stacking methods. Stacking methods perform predictions in two stages. The first stage generates some initial estimations and the second stage combines these estimations and produces a final prediction. Two most well-known stacking methods in the literature are called 2BR [47, 106] and DBR [80]. The stage-1 models in 2BR and DBR are standard BR models. The stage-2 models in 2BR and DBR work differently. In 2BR, the stage-2 model predicts each label ℓ with a separate binary classifier which takes as input the original instance feature vector $\mathbf{x}$ as well as all label probabilities predicted by the stage-1 model. In DBR, the stage-2 model predicts each label ℓ with a separate binary classifier which takes as input the original instance feature vector $\mathbf{x}$ as well as the binary absence/presence information of all other labels. The absence/presence of label ℓ itself is not part of the input to avoid learning a trivial mapping. During training, the absence/presence information is obtained from the ground truth; during prediction, it is obtained from the stage-1 model's prediction. Clearly for DBR there is some inconsistency on how stage-2 inputs are obtained. 2BR does not suffer from such inconsistency. However, both 2BR and DBR have a critical flaw: when their stage-2 models make the final decision for a particular label, they do not really take into account the *final decisions* made for other labels by the stage-2 model; they instead only consider the *initial estimations* on other labels made by the stage-1 model, which can be quite different. As a result, the final set predictions made by 2BR and DBR may not respect the label dependencies/constraints these models have learned.

Neural Networks. Several popular deep neural networks have been applied to multi-label prediction. Examples include recurrent neural network (RNN) [115], convolutional neural networks [71] and auto-encoders [113]. Because RNN is original designed for sequence prediction task, one has to transform sets to sequences during training and transform sequences to sets during prediction. The order in which labels appear in a sequence has an impact on model performance [82, 95].

The Structured Prediction Energy Network (SPEN) [15] uses deep neural networks to efficiently encode arbitrary relations between labels, which to large degree avoids parameterization issue associated with pair-wise CRF, but it cannot generate a confidence score for its MAP prediction as computing the normalization constant is intractable.

Deep value network (DVN) [52] trains a neural network to evaluate prediction candidates and then uses back-propagation to find the prediction that leads to the maximum score.

Other models. There are many other approaches that seek to model label dependencies in different ways [88, 53, 23, 132, 121, 31, 19]. For example, the probabilistic classifier tree method [31] is designed to estimate the joint probability $p(\mathbf{y}|\mathbf{x})$ by following a path from the root to a leaf node in a tree hierarchy using the chain rule. It reduces the training and prediction problem into many simple subproblems and has lower training complexity compared to BR. The Constraint-and-Predict method (CP) [19] is proposed to specifically incorporate cardinality constraint into learning and prediction. Several approaches adapt existing machine learning models, such as Bayesian network [129], and determinantal point process [119]. The survey [45] covers many other methods.

1.3.2 Learning on Large Scale Data

One way to scale up model training is to exploit the sparsity in the feature matrix or the label matrix. The PDsparse method [124] employs a primal and dual sparse approach to train BR. It reports a compression ratio of 1000 in the model size using elastic-net regularization. DiSMEC [14] trains BR in a distributed fashion. However, these methods do not capture any model dependencies. There are two main families of algorithms that estimate label dependencies and simultaneously achieve sub-linear training time: embedding methods and tree based methods. Embedding methods [17, 27] assume some low rank structure of the label matrix and find a low dimensional embedding of the labels. One scoring function is trained for each dimension in the embedding space. However, the low rank assumption is often violated in real data due to the long tail of labels [120]. Tree based methods [93, 57] partition both the feature space and the label space repeatedly.

1.3.3 Dealing with Label Noise

Existing approaches that tackle label noise make different assumptions about noise: [125, 118, 117] assume that the human annotation has three possibilities: "positive", "negative" and "missing". In this setup, the "positive" and "negative" annotations are noise free, and the "missing" label annotation can be either positive or negative. Note that the annotator needs to explicitly indicate that he/she forgets to judge those "missing" labels. Methods designed for this setup have only been tested on artificial data and seem to have only theoretical interests, as in practice, datasets are rarely labeled this way. In almost all the real multi-label datasets, each instance is only labeled with some "positive" labels. The rest of the labels that were not added to the instance may contain the labels that truly do not apply as well as the labels

that should apply but were missed by the annotators. Sometimes even those marked as "positive" could be unreliable. In this realistic setup, the learning algorithms do not know *a priori* which annotations are reliable and which are not and have to detect incorrect annotations on the fly. Admittedly, learning in this setup is very difficult. Using dedicated model structure to model noise or using robust training objectives typically has noticeable but limited performance improvement [78]. On some multi-label tasks, the noise can be assumed to be one sided — that annotators are not malicious; the "positive" labels they put are in general correct, but they could miss some positive labels. This additional assumption simplifies learning, and many such algorithms have been developed, under the general name of "PU" learning [57, 59]. The PfastXML method [57] assumes that the missing rate of each label can be obtained from an external source. Some methods specifically deal with image annotation noise [38, 70]. Recently, researchers have observed several difficulties and limitations of training robust classifiers purely on noisy annotations and start to explore the possibility of leveraging an additional set of noise-free data [110, 111, 55].

1.4 Summary of Contributions

This thesis considers the problem of multi-label classification and introduces two new classification methods named conditional Bernoulli mixture (CBM) and BR-rerank. Both methods seek to improve multi-label classification accuracy by leveraging label dependencies, and they tackle the challenge of label dependency estimation by reducing a multi-label classification problem to simpler binary classification, multi-class classification, or regression problems. Our contributions are as follows:

- We introduce a new multi-label model, named conditional Bernoulli mixture (CBM) model, which directly approximates the conditional joint label probability by a mixture of binary relevance models. CBM reduces a complex multi-label classification problem into a multi-class classification problem and a series of binary classification problems, each of which is easier to solve.

- We derive an Expectation-Maximization (EM) based training procedure for CBM, and show that the training procedure has a modular design and can easily incorporate existing binary and multi-class base learners training procedures as subroutines. We also develop tricks to speed up CBM training by leveraging the sparsity in the model structure as well as in the data.

- We develop three different prediction methods that aim to optimize different evaluation metrics: set accuracy, Hamming loss and F1 score. To optimize set accuracy, we develop a dynamic programming procedure to efficiently find the most probable label set.
- We introduce another new multi-label method, named BR-rerank, which employs a post-calibration and reranking procedure to improve the prediction accuracy and prediction confidence of the widely used binary relevance model. BR-rerank captures label dependencies in the post-processing step and reduces a complex multi-label classification problem into a regression problem and a series of binary classification problems, each of which is easier to solve.
- We identify several features that are critical in multi-label prediction confidence calibration and propose to use gradient boosting as the calibrator due to its effectiveness in modeling feature and label interactions.
- We develop a new way to train calibrator in the presence of label noise which gives unbiased estimation of the calibration confidence.
- Our analysis shows that both CBM and BR-rerank avoid many issues associated with existing multi-label methods. Experimental results also show the effectiveness of the proposed methods against competitive alternatives on benchmark datasets.
- We implement the proposed method CBM and BR-rerank together with many baseline methods such as BR, PCC, CRF, DBR and make the implementations publicly available at `https://github.com/cheng-li/pyramid`.

The work in this thesis is a revised and extended presentation of research developed through several co-authored papers. Chapter 3 is based on a paper in ICML 2016 [68] and a paper in ICMLA 2018 [114]. Chapter 4 is based on a paper in ECML-PKDD 2019 [67].

Chapter 2

Foundations

In this chapter, we describe some basic concepts, theories and algorithms which will be used in later chapters.

2.1 Evaluation Metrics for Multi-label Classification

One distinct feature of multi-label classification is that there are many different evaluation metrics. In this section, we describe several commonly used metrics.

Let $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ be a multi-label dataset with N instances and L candidate labels. The n-th instance has feature vector $\mathbf{x}_n$ and ground truth labels $\mathbf{y}_n$. Suppose $\hat{\mathbf{y}}_n$ is the set prediction made for the n-th instance by some classifier. Then we have the following categorization for each binary label prediction result made for each instance:

$$c_{nl} = \begin{cases} tp, & \text{if } \hat{y}_{nl} = 1, y_{nl} = 1 \\ fp, & \text{if } \hat{y}_{nl} = 1, y_{nl} = 0 \\ tn, & \text{if } \hat{y}_{nl} = 0, y_{nl} = 0 \\ fn, & \text{if } \hat{y}_{nl} = 0, y_{nl} = 1 \end{cases} \tag{2.1}$$

where tp, fp, tn, fn stand for true-positive, false-positive, true-negative, and false-negative predictions, respectively.

One can summarize the prediction results by defining three N-by-L matrices $\mathbf{Y}$, $\hat{\mathbf{Y}}$ and $\mathbf{C}$. In all three matrices, each row corresponds to an instance, and

each column corresponds to a label. $\mathbf{Y}$ is the ground-truth matrix, and each entry $y_{nl} \in \{0, 1\}$ indicates whether the l-th label is truly relevant for the n-th instance. $\hat{\mathbf{Y}}$ is the prediction matrix, and each entry $\hat{y}_{nl} \in \{0, 1\}$ indicates whether the l-th label is predicted for the n-th instance. $\mathbf{C}$ is the correctness matrix, and each entry $c_{nl} \in \{tp, fp, tn, fn\}$ indicates whether the decision made for the n-th instance w.r.t. the l-th label is correct. For example, these matrices defined for 5 instances and 4 labels may look like the following:

$$\mathbf{Y} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}, \quad \hat{\mathbf{Y}} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} tp & fp & tp & fn \\ tn & tp & fp & fn \\ fp & tn & tn & tn \\ fn & tp & tp & tp \\ tp & tn & tp & tn \end{bmatrix}$$

There are many different evaluation metrics for multi-label classification. Most of them are computed based solely on the $\mathbf{C}$ matrix. Depending on how the statistics are aggregated during the computation, those metrics can be categorized into instance-averaged, label-averaged (also called macro-averaged) and micro-averaged metrics [61].

There are also metrics defined not in terms of entries in the $\mathbf{C}$ matrix. Examples include instance-averaged (or label-averaged) precision at K, instance-averaged (or label averaged) NDCG. These are ranking based metrics, which require the underlying multi-label classifier to provide a ranking of instances (or labels) according to their relevance w.r.t. a label (or instance). In this thesis, we will focus on metrics defined by the $\mathbf{C}$ matrix.

2.1.1 Instance-Averaged Metrics

An instance-averaged metrics first computes a performance score for each instance (each row in the $\mathbf{C}$ matrix), and then averages scores across all instances. It measures the expected classification performance of a randomly picked instance.

First we define the total number of tp, fp, tn, fn entries in the n-th row of the

$\mathbf{C}$ matrix as TP_n, FP_n, TN_n, FN_n, respectively.

$$\mathrm{TP}_n = \sum_{l=1}^{L} \mathbb{I}[c_{nl} = tp] \tag{2.2}$$

$$\mathrm{FP}_n = \sum_{l=1}^{L} \mathbb{I}[c_{nl} = fp] \tag{2.3}$$

$$\mathrm{TN}_n = \sum_{l=1}^{L} \mathbb{I}[c_{nl} = tn] \tag{2.4}$$

$$\mathrm{FN}_n = \sum_{l=1}^{L} \mathbb{I}[c_{nl} = fn] \tag{2.5}$$

where $\mathbb{I}[\cdot]$ is the indicator function. We then define the following instance-averaged metrics:

- instance-averaged precision:

$$\frac{1}{N} \sum_{n=1}^{N} \frac{\mathrm{TP}_n}{\mathrm{TP}_n + \mathrm{FP}_n} \tag{2.6}$$

  It measures, on average for each instance, what fraction of the predicted labels are correct.

- instance-averaged recall:

$$\frac{1}{N} \sum_{n=1}^{N} \frac{\mathrm{TP}_n}{\mathrm{TP}_n + \mathrm{FN}_n} \tag{2.7}$$

  It measures, on average for each instance, what fraction of the relevant labels are correctly predicted.

- instance-averaged F1:

$$\frac{1}{N} \sum_{n=1}^{N} \frac{2\mathrm{TP}_n}{2\mathrm{TP}_n + \mathrm{FP}_n + \mathrm{FN}_n} \tag{2.8}$$

  The F1 score defined for each instance $\frac{2\mathrm{TP}_n}{2\mathrm{TP}_n+\mathrm{FP}_n+\mathrm{FN}_n}$ is the harmonic mean of precision $\frac{\mathrm{TP}_n}{\mathrm{TP}_n+\mathrm{FP}_n}$ and recall $\frac{\mathrm{TP}_n}{\mathrm{TP}_n+\mathrm{FN}_n}$. Instance-averaged F1 can also be written as

$$\frac{1}{N} \sum_{n=1}^{N} \frac{2|\mathbf{y}_n \cap \hat{\mathbf{y}}_n|}{|\mathbf{y}_n| + |\hat{\mathbf{y}}_n|} \tag{2.9}$$

where $\mathbf{y}_n$ and $\hat{\mathbf{y}}_n$ are treated as set of labels, $\cap$ is the intersection operator, and $|\cdot|$ standards for the cardinality (size) of the set. Note that for each instance, its F1 score achieves the maximum value of 1 when the predicted set equals the ground truth set, and the minimum value of 0 when none of the predicted labels are in the ground truth set. F1 score lies between 0 and 1 when the predicted set is partially correct.

- instance-averaged Jaccard index:

$$\frac{1}{N}\sum_{n=1}^{N}\frac{\mathrm{TP}_n}{\mathrm{TP}_n+\mathrm{FP}_n+\mathrm{FN}_n} \tag{2.10}$$

Jaccard index can also be written as

$$\frac{1}{N}\sum_{n=1}^{N}\frac{|\mathbf{y}_n\cap\hat{\mathbf{y}}_n|}{|\mathbf{y}_n\cup\hat{\mathbf{y}}_n|} \tag{2.11}$$

where $\mathbf{y}_n$ and $\hat{\mathbf{y}}_n$ are treated as set of labels, and $\cup$ is the union operator. Jaccard index is quite similar to F1 score in the sense that they both measure the degree of overlap between the predicted set and the ground truth set.

- instance-averaged Hamming loss:

$$\frac{1}{N}\sum_{n=1}^{N}\frac{\mathrm{FP}_n+\mathrm{FN}_n}{L} \tag{2.12}$$

It looks at all the binary decisions made on all labels, and calculates what fraction of binary decisions are incorrect.

- subset accuracy (also called set accuracy):

$$\frac{1}{N}\sum_{n=1}^{N}\mathbb{I}[\mathrm{FP}_n+\mathrm{FN}_n=0] \tag{2.13}$$

It counts the fraction of instances with perfect predictions. This is the most stringent metric – partially correct set predictions do not receive any credit. It can also be written as

$$\frac{1}{N}\sum_{n=1}^{N}\mathbb{I}[\mathbf{y}_n=\hat{\mathbf{y}}_n] \tag{2.14}$$

In the literature, 1 minus subset accuracy is referred to as 0/1 loss.

2.1.2 Label-Averaged Metrics

A label-averaged (macro averaged) metrics first computes a performance score for each label (each column in the $\mathbf{C}$ matrix), and then averages scores across all labels. It measures the expected classification performance of a randomly picked label.

The definitions of label-averaged metrics are symmetric to the definitions of instance-averaged metrics. First we define the total number of tp, fp, tn, fn entries in the l-th column of the $\mathbf{C}$ matrix as TP_l, FP_l, TN_l, FN_l, respectively.

$$\mathrm{TP}_l = \sum_{n=1}^{N} \mathbb{I}[c_{nl} = tp] \tag{2.15}$$

$$\mathrm{FP}_l = \sum_{n=1}^{N} \mathbb{I}[c_{nl} = fp] \tag{2.16}$$

$$\mathrm{TN}_l = \sum_{n=1}^{N} \mathbb{I}[c_{nl} = tn] \tag{2.17}$$

$$\mathrm{FN}_l = \sum_{n=1}^{N} \mathbb{I}[c_{nl} = fn] \tag{2.18}$$

We then define the following label-averaged metrics:

- label-averaged precision:

$$\frac{1}{L} \sum_{l=1}^{L} \frac{\mathrm{TP}_l}{\mathrm{TP}_l + \mathrm{FP}_l} \tag{2.19}$$

  It measures the precision of each binary label classifier (what fraction of instanced predicted to have this label truly have this label), and averages across all labels.

- label-averaged recall:

$$\frac{1}{L} \sum_{l=1}^{L} \frac{\mathrm{TP}_l}{\mathrm{TP}_l + \mathrm{FN}_l} \tag{2.20}$$

  It measures the recall of each binary label classifier (what fraction of instances with this label are predicted to have this label), and averages across all labels.

- label-averaged F1:

$$\frac{1}{L} \sum_{l=1}^{L} \frac{2\mathrm{TP}_l}{2\mathrm{TP}_l + \mathrm{FP}_l + \mathrm{FN}_l} \tag{2.21}$$

The F1 score defined for each label $\frac{2\text{TP}_l}{2\text{TP}_l+\text{FP}_l+\text{FN}_l}$ is the harmonic mean of precision $\frac{\text{TP}_l}{\text{TP}_l+\text{FP}_l}$ and recall $\frac{\text{TP}_l}{\text{TP}_l+\text{FN}_l}$.

- label-averaged Jaccard index:

$$\frac{1}{L}\sum_{l=1}^{L}\frac{\text{TP}_l}{\text{TP}_l+\text{FP}_l+\text{FN}_l} \tag{2.22}$$

- label-averaged Hamming loss:

$$\frac{1}{L}\sum_{l=1}^{L}\frac{\text{FP}_l+\text{FN}_l}{N} \tag{2.23}$$

This is the same as instance-averaged Hamming loss.

Note that label-averaged metrics implicitly assume that all labels are equally important, which is often not true in practice, as some labels are far more popular than others. Label-averaged metrics often suffer from two issues: long tail and instability. 1) The long tail issue: Most of the real-world multi-label datasets exhibit a long tail: there are often a few popular labels with many matched instances, and a large number of rare labels with few matched instances. Those few popular labels could occur more often than all rare labels combined. But because of the sheer number of rare labels, label-averaged metrics are dominated by the performance on rare labels. 2) The instability issue: many rare labels only appear a few times in the test set, some only appear once, and some do not appear at all. This sparsity makes metrics defined on rare labels unstable to compute. Consider a rare label that only appears once in the test set. The classifier's recall on this label is either 1 (if the classifier finds this single occurrence) or 0 (if the classifier misses it).

2.1.3 Micro-Averaged Metrics

A micro-averaged metrics first aggregates tp, fp, tn, fn from all entries of the $\mathbf{C}$ matrix, and then defines a performance score using these 4 aggregated numbers. It measures the expected classification performance of a randomly picked instance-label pair.

First we define the total number of tp, fp, tn, fn entries in the entire $\mathbf{C}$ matrix

as TP, FP, TN, FN, respectively.

$$\text{TP} = \sum_{n=1}^{N} \sum_{l=1}^{L} \mathbb{I}[c_{nl} = tp] \tag{2.24}$$

$$\text{FP} = \sum_{n=1}^{N} \sum_{l=1}^{L} \mathbb{I}[c_{nl} = fp] \tag{2.25}$$

$$\text{TN} = \sum_{n=1}^{N} \sum_{l=1}^{L} \mathbb{I}[c_{nl} = tn] \tag{2.26}$$

$$\text{FN} = \sum_{n=1}^{N} \sum_{l=1}^{L} \mathbb{I}[c_{nl} = fn] \tag{2.27}$$

We then define the following micro-averaged metrics:

- micro-averaged precision:

$$\frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.28}$$

  It looks at all the predicted labels for all instances and counts what fraction of them are actually correct.

- micro-averaged recall:

$$\frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.29}$$

  It looks at all the relevant labels for all instances and counts what fraction of them are predicted.

- micro-averaged F1:

$$\frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} \tag{2.30}$$

- micro-averaged Jaccard index:

$$\frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}} \tag{2.31}$$

- micro-averaged Hamming loss:

$$\frac{\text{FP} + \text{FN}}{NL} \tag{2.32}$$

  This is the same as instance-averaged or label-averaged Hamming loss.

2.2 Optimization for Target Metric

In multi-label classification, there are two general ways to improve prediction performance w.r.t. an evaluation metric: a) improve the model or the training method to better fit the data; b) make predictions that are tailored to the metric. The first one is true for all types of classification problems (including binary and multi-class classifications) and is easy to understand. The second one is particularly true for multi-label classification because in multi-label classification, one prediction strategy that works well under one evaluation metric may not work well under another evaluation metric. In general, given a conditional joint distribution $p(\mathbf{y}|\mathbf{x})$ learned from the data, one need to apply different prediction strategies in order to optimize different evaluation metrics. In this section, we present some theoretical results on the optimal predictions w.r.t. different evaluation metrics. These results are from [32, 30, 112, 61].

Hamming loss. Hamming loss decomposes over labels and this greatly simplified the problem. The optimal prediction w.r.t. Hamming loss is made by thresholding each marginal label probability at 0.5.

$$h^*(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), ..., h_L(\mathbf{x})) \tag{2.33}$$

where

$$h_\ell(\mathbf{x}) = \begin{cases} 1 \text{ if } p(y_\ell|\mathbf{x}) > 0.5 \\ 0 \text{ otherwise} \end{cases} \tag{2.34}$$

The result shows that theoretically there is no need to consider label dependencies if the end goal is to optimize Hamming loss.

Set accuracy. The optimal prediction w.r.t. set accuracy is given by the mode of the conditional joint distribution:

$$h^*(\mathbf{x}) = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \tag{2.35}$$

Intuitively, for the predicted set to be correct as a whole, all labels in the set need to be considered together. One cannot decide each label regardless of other labels. Searching for the most probable label combination is a computationally difficult problem in general as the search space is exponentially large. To facilitate the search procedure, one could model the joint distribution with some easy-to-manipulate distribution. Also, instead of looking for the exact optimal solution, one could return a solution that is close enough to the optimal one.

Label-averaged F1. The optimal prediction w.r.t. label-averaged F1 is made by thresholding each marginal label probability at a different value.

$$h^*(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), ..., h_L(\mathbf{x})) \tag{2.36}$$

where

$$h_\ell(\mathbf{x}) = \begin{cases} 1 \text{ if } p(y_\ell|\mathbf{x}) > \delta_\ell \\ 0 \text{ otherwise} \end{cases} \tag{2.37}$$

and each $\delta_\ell \in (0, 1)$ is some threshold for a label. This result shows that to optimize label-averaged F1, one can tune a threshold using validation data for each label separately to optimize the F1 on the label, and then apply these thresholds during prediction. As in the case of Hamming loss, it suffices to only estimate the marginal probabilities and there is no need to capture label dependencies.

Micro-averaged F1. The optimal prediction w.r.t. micro-averaged F1 is made by thresholding all marginal label probabilities at the same value.

$$h^*(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), ..., h_L(\mathbf{x})) \tag{2.38}$$

where

$$h_\ell(\mathbf{x}) = \begin{cases} 1 \text{ if } p(y_\ell|\mathbf{x}) > \delta \\ 0 \text{ otherwise} \end{cases} \tag{2.39}$$

and $\delta \in (0, 1)$ is some threshold. This result shows that to optimize micro-averaged F1, one can tune a single threshold to optimize the micro-averaged F1 on the validation data, and then apply the threshold during prediction. As in the case of label-averaged F1, it suffices to only estimate the marginal probabilities. However, unlike label-averaged F1 optimization which does not consider label dependencies at all, micro-averaged F1 optimization incorporates some weak form of label dependencies through the shared threshold tuning.

Instance-averaged F1. Instance-averaged F1 is one of the most widely used metrics and how to optimize it has attracted a lot of attentions. Generally speaking, there are two different approaches. The empirical utility maximization (EUM) approach trains a classifier to directly maximize the F1 score on the training data (or through cross-validation). The Decision Theoretical Analysis (DTA) approach first builds a general probabilistic model $p(\mathbf{y}|\mathbf{x})$ using training data without concerning the F1 metric, and then runs additional inference step to maximize the expected F1

score during prediction:

$$\begin{aligned} h^*(\mathbf{x}) &= \arg\max_{\mathbf{y}'} \mathbf{E}_{\mathbf{y}\sim p(\mathbf{y}|\mathbf{x})}[F(\mathbf{y},\mathbf{y}')] \\ &= \arg\max_{\mathbf{y}'} \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \cdot F(\mathbf{y},\mathbf{y}') \end{aligned} \tag{2.40}$$

where $\mathbf{y}$ is the (unknown) ground truth, $\mathbf{y}'$ is a candidate prediction and $F(\mathbf{y},\mathbf{y}') = \frac{2\sum_{l=1}^{L} y_l y'_l}{\sum_{l=1}^{L} y_l + \sum_{l=1}^{L} y'_l}$ is the F1 score of the prediction. Note that the ground truth $\mathbf{y}$ is unknown at prediction time, and is treated as a random variable whose distribution $p(\mathbf{y}|\mathbf{x})$ is estimated by the classifier.

Previous empirical studies [83] suggest that each of the two approaches above has some advantaged and disadvantages. On one hand, if the model class is severely misspecified and the probabilistic estimations are inaccurate, EUM can be more robust than DTA. On the other hand, if the test distribution differs from the training one (a common scenario is when $p(\mathbf{x})$ changes but $p(\mathbf{y}|\mathbf{x})$ does not), DTA is more robust than EUM. The survey paper [91] summarizes many theoretical results and algorithms aimed at maximizing the F1 metric.

The simplest EUM style classifier consists of a set of marginal probability estimators and a shared threshold tuned to optimize instance-averaged F1 score on held-out data [61]. It has exactly the same form as the micro-averaged F1 optimizer:

$$h^*(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), ..., h_L(\mathbf{x})) \tag{2.41}$$

where

$$h_\ell(\mathbf{x}) = \begin{cases} 1 \text{ if } p(y_\ell|\mathbf{x}) > \delta \\ 0 \text{ otherwise} \end{cases} \tag{2.42}$$

and $\delta \in (0,1)$ is some threshold.

There are also some more complex EUM style classifiers. One example is the structured support vector machines (SSVMs) method, which minimizes a convex upper bound of the F1 loss (1 minus F1 score) [105, 89, 90].

The most representative DTA approach is the General F-measure Maximizer (**GFM**) [112], which is an efficient inference algorithm that finds the F1-optimal prediction for a given instance based on some probability estimations (see Algorithm 2.1). The direct input to GFM is not a joint estimation $p(\mathbf{y}|\mathbf{x})$, but rather, some marginal distributions of the form $p(y_l = 1, |\mathbf{y}| = s\,|\mathbf{x})$, $\forall l, s \in \{1, ..., L\}$, where $|\mathbf{y}|$ stands for the number of relevant labels in $\mathbf{y}$. The paper [112] proposed two ways of obtaining these L^2 marginals: (1) a model which directly estimates L^2

marginals from data, and (2) the use of a probabilistic joint classifier/estimator $p(\mathbf{y}|\mathbf{x})$ and sampling to generate the required L^2 probabilities. The GFM algorithm has the complexity $\Theta(L^{2.376})$, where L is the number of all possible labels. When the maximum number of labels per instance T is far less than L, one can replace the $L \times L$ matrix P in the first line of the algorithm by a $L \times T$ matrix and reduce the complexity to $\Theta(LT^2)$. Typically T is less than 10 even when L could be on the order of hundreds or thousands. This can lead to significant speedup on large datasets.

Algorithm 2.1 General F-Measure Maximizer [112]

1: **Input:** $p(\mathbf{y} = \mathbf{0}|\mathbf{x})$ and L by L Matrix P with elements $p_{ls} = p(y_\ell = 1, |\mathbf{y}| = s\,|\,\mathbf{x})$
2: Define L by L Matrix W with elements $w_{sk} = \frac{2}{s+k}$
3: Compute L by L Matrix $\Delta = PW$
4: **for** $s = 1, 2, ..., L$ **do**
5: The best prediction $\mathbf{y}^s$ with s labels is given by including the s labels l with the highest Δ_{ls}
6: Compute the expected F measure $\mathbf{E}[F(\mathbf{y}, \mathbf{y}^s)] = \sum_{\ell=1}^{L} y_l^s \Delta_{\ell s}$
7: **end for**
8: The expected F measure for the empty prediction $y^0 = \mathbf{0}$ is $\mathbf{E}[F(\mathbf{y}, \mathbf{y}^0)] = p(\mathbf{y} = \mathbf{0}|\mathbf{x})$.
9: **Output:** optimal $\mathbf{y}^* = \text{argmax}_{0 \le s \le L}\, \mathbf{E}[F(\mathbf{y}, \mathbf{y}^s)]$

2.3 Commonly Used Base Learners

Many multi-label classifiers, including the ones we propose in this thesis, use some regressors, binary classifiers or multi-class classifiers as base learners. Here we describe several popular and representative base learners.

2.3.1 Linear Regression

Linear regression is the simplest regression method. It models the continuous output variable as a linear function of the input variables:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_D x_D \tag{2.43}$$

Given a regression dataset $\{(\mathbf{x}_n, y_n, w_n)\}_{n=1}^N$ where each instance $\mathbf{x}_n$ has label y_n and weight w_n, we can train a linear regression model by minimizing the weighted square loss plus some regularization penalty term:

$$\min_{\beta_0,\boldsymbol{\beta}} \frac{1}{2\sum_{n=1}^{N} w_n} \sum_{n=1}^{N} w_n (y_n - \beta_0 - \sum_{d=1}^{D} \beta_d x_{nd})^2 + \lambda[\alpha||\boldsymbol{\beta}||_1 + (1-\alpha)\frac{1}{2}||\boldsymbol{\beta}||_2^2] \quad (2.44)$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_D)$, $||\cdot||_1$ is the L1 norm, $||\cdot||_2$ is the L2 norm, λ is the overall elastic-net penalty strength, and α is the L1 ratio [39].

The optimization problem (2.44) can be solved in several different ways. One popular solution is the coordinate descent method [39], which successively minimizes the overall loss along each coordinate while having all other coordinates fixed:

$$\beta_0 \leftarrow \frac{\sum_{n=1}^{N} w_n (y_n - \sum_{d=1}^{D} \beta_d x_{nd})}{\sum_{n=1}^{N} w_n} \quad (2.45)$$

$$\beta_d \leftarrow \frac{S(\frac{1}{\sum_{n=1}^{N} w_n} \sum_{n=1}^{N} w_n x_{nd}(y_n - \tilde{y}_n^{(d)}), \lambda\alpha)}{\frac{1}{\sum_{n=1}^{N} w_n} \sum_{n=1}^{N} w_n x_{nd}^2 + \lambda(1-\alpha)} \quad (2.46)$$

where

- $\tilde{y}_n^{(d)} = \beta_0 + \sum_{j \neq d} \beta_j x_{nj}$ is the fitted value excluding the contribution from x_{nd}.
- $S(z, \gamma)$ is the soft-thresholding operator with value

$$\text{sign}(z)(|z| - \gamma)_+ = \begin{cases} z - \gamma & \text{if } z > 0 \text{ and } \gamma < |z| \\ z + \gamma & \text{if } z < 0 \text{ and } \gamma < |z| \\ 0 & \text{if } \gamma \geq |z| \end{cases} \quad (2.47)$$

The overall training procedure for elastic-net regularized linear regression is summarized in Algorithm 2.2.

2.3.2 Multi-Class Logistic Regression

Logistic regression is one of the most widely used linear classifiers. Here we describe the multi-class version of the logistic regression model (also called multinomial logistic regression), as it is quite general and includes the binary version as a special case. For a multi-class problem with K classes, logistic regression assigns a set of weights $\boldsymbol{\beta}_k = (\beta_{k1}, ..., \beta_{kD})$ and a bias term β_{k0} to each class k, and models the class conditional probability as

$$p(y = k|\mathbf{x}) = \frac{e^{\beta_{k0} + \sum_{d=1}^{D} \beta_{kd} x_d}}{\sum_{l=1}^{K} e^{\beta_{l0} + \sum_{d=1}^{D} \beta_{ld} x_d}} \quad (2.48)$$

Algorithm 2.2 Training Elastic-net Regularized Linear Regression by Coordinate Descent [39]

Input: weighted regression dataset $\{(\mathbf{x}_n, y_n, w_n)\}_{n=1}^N$, initial model parameters $\beta_0, \boldsymbol{\beta}$

1: **repeat**
2: Update β_0 as in (2.45)
3: **for** $d = 1, 2, ..., D$ **do**
4: Update β_d as in (2.46)
5: **end for**
6: **until** convergence

Output: model parameters $\beta_0, \boldsymbol{\beta}$

Consider a general multi-class dataset $\{(\mathbf{x}_n, y_n, w_n)\}_{n=1}^N$ in which each instance $\mathbf{x}_n$ has instance weight w_n and soft target label y_n. Here w_n is a non-negative number, and $y_n = (y_{n1}, y_{n2}, ..., y_{nK})$ standards for the target multinomial distribution satisfying $y_{nk} \in [0, 1]$ and $\sum_{k=1}^K y_{nk} = 1$.

Logistic regression can be trained by minimizing the regularized loss function

$$\min_{\{\beta_{k0}, \boldsymbol{\beta}_k\}_{k=1}^K} \frac{1}{\sum_{n=1}^N w_n} \sum_{n=1}^N w_n \mathbb{KL}(y_n || p(y|\mathbf{x}_n)) + \lambda \sum_{k=1}^K [\alpha ||\boldsymbol{\beta}_k||_1 + (1-\alpha)\frac{1}{2}||\boldsymbol{\beta}_k||_2^2] \tag{2.49}$$

where $\mathbb{KL}(y_n||p(y|\mathbf{x}_n)) = \sum_{k=1}^K y_{nk}[\log y_{nk} - \log p(y = k|\mathbf{x}_n)]$ is the KL divergence between the target class distribution and the predictive class distribution, $||\cdot||_1$ is the L1 norm, $||\cdot||_2$ is the L2 norm, λ is the overall elastic-net penalty strength, and α is the L1 ratio [39].

This loss function (2.49) can be minimized by repeatedly forming a quadratic approximation to the loss function and solving the resulting least square linear regression problem for each class k:

$$\min_{\beta_{k0}, \boldsymbol{\beta}_k} \frac{1}{2\sum_n^N w'_{nk}} \sum_{n=1}^N w'_{nk}(z_{nk} - \beta_{k0} - \sum_{d=1}^D \beta_{kd} x_{nd})^2 + \lambda[\alpha ||\boldsymbol{\beta}_k||_1 + (1-\alpha)\frac{1}{2}||\boldsymbol{\beta}_k||_2^2] \tag{2.50}$$

where

$$z_{nk} = \beta_{k0} + \sum_{d=1}^D \beta_{kd} x_{nd} + \frac{y_{nk} - p(y = k|\mathbf{x}_n)}{p(y = k|\mathbf{x}_n)(1 - p(y = k|\mathbf{x}_n))} \tag{2.51}$$

$$w'_{nk} = w_n p(y = k|\mathbf{x}_n)(1 - p(y = k|\mathbf{x}_n)) \tag{2.52}$$

which are computed based on current model parameters and fixed during the optimization of (2.50).

The overall training procedure for elastic-net regularized multi-class logistic regression is summarized in Algorithm 2.3.

Algorithm 2.3 Training Elastic-net Regularized Multi-Class Logistic Regression by Coordinate Descent [39]

Input: weighted multi-class dataset $\{(\mathbf{x}_n, y_n, w_n)\}_{n=1}^N$, initial model parameters $\{\beta_{k0}, \boldsymbol{\beta}_k\}_{k=1}^K$

1: **repeat**
2: **for** $k = 1, 2, ..., K$ **do**
3: Form the quadratic approximation (2.50) to the loss function (2.49)
4: Treat (2.50) as a weighted-least-square linear regression problem with training dataset $\{(\mathbf{x}_n, z_{nk}, w'_{nk})\}_{n=1}^N$ and model parameters $\beta_{k0}, \boldsymbol{\beta}_k$ and solve it using Algorithm 2.2
5: **end for**
6: **until** convergence

Output: model parameters $\{\beta_{k0}, \boldsymbol{\beta}_k\}_{k=1}^K$

2.3.3 Gradient Boosting Regressor

Gradient Boosting (GB, or GBDT) is a general framework for training tree ensembles to optimize given loss functions. A tree ensemble is a list of regression trees that together produce an overall output:

$$F(\mathbf{x}) = h_1(\mathbf{x}) + h_2(\mathbf{x}) + \cdots + h_T(\mathbf{x}) \tag{2.53}$$

where $F(\mathbf{x})$ is the overall ensemble output, and each $h_t(\mathbf{x})$ is a regression tree output.

Given a regression/classification dataset $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ and a suitable loss function $L(y, F(\mathbf{x}))$, gradient boosting builds the tree ensemble in a greedy, stage-wise fashion (see Algorithm 2.4).

For regression tasks, one can easily modify the general procedure (Algorithm 2.4) to optimize squared loss $L(y, F(\mathbf{x})) = (y - F(\mathbf{x}))^2$, as shown in Algorithm 2.5.

Algorithm 2.4 Generic Gradient Boosting Training [40]

Input: dataset $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$, loss function L, initial model $F(\mathbf{x})$

for iteration $t = 1, 2, ..., T$ **do**
  for $n = 1, 2, ..., N$ **do**
    compute the gradient of the loss L w.r.t. the current ensemble score $F(\mathbf{x}_n)$: $g_n = \frac{\partial L(y_n, F(\mathbf{x}_n))}{\partial F(\mathbf{x}_n)}$
  end for
  fit a regression tree h_t to the regression dataset $\{(\mathbf{x}_n, -g_n)\}_{n=1}^N$
  shrink the regression tree leaf output values by ρ: $h_t \leftarrow \rho h_t$
  add the new regression tree h_t to the ensemble: $F = h_1 + h_2 + \cdots + h_t$
end for

Output: final ensemble $F = h_1 + h_2 + \cdots + h_T$

Algorithm 2.5 Gradient Boosting Regressor Training [40]

Input: regression dataset $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$, initial model $F(\mathbf{x})$

for iteration $t = 1, 2, ..., T$ **do**
  for $n = 1, 2, ..., N$ **do**
    compute the gradient of the loss L w.r.t. the current ensemble score $F(\mathbf{x}_n)$: $g_n = \frac{\partial L(y_n, F(\mathbf{x}_n))}{\partial F(\mathbf{x}_n)} = F(\mathbf{x}_n) - y_n$
  end for
  fit a regression tree h_t to the regression dataset $\{(\mathbf{x}_n, -g_n)\}_{n=1}^N$
  shrink the regression tree leaf output values by ρ: $h_t \leftarrow \rho h_t$
  add the new regression tree h_t to the ensemble: $F = h_1 + h_2 + \cdots + h_t$
end for

Output: final ensemble $F = h_1 + h_2 + \cdots + h_T$

2.3.4 Binary Gradient Boosting Classifier

The ensemble score $F(\mathbf{x})$ provided by gradient boosting model can be easily turned into a probability using the sigmoid transformation $p(y = 1|\mathbf{x}) = \frac{1}{1+e^{-F(\mathbf{x})}}$. This allows GB to be trained on binary classification datasets using maximum likelihood estimation.

Given a binary classification dataset $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$, we define the loss function on each instance to be the negative log likelihood or KL-divergence $L(y_n, F(\mathbf{x}_n)) = -\mathbb{I}[y_n = 0] \log p(y = 0|\mathbf{x}) - \mathbb{I}[y_n = 1] \log p(y = 1|\mathbf{x})$. One can easily modify the general procedure (Algorithm 2.4) to optimize this loss function, as shown in Algorithm 2.6.

Algorithm 2.6 Binary Gradient Boosting Classifier Training [40]

Input: binary classification dataset $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$, initial model $F(\mathbf{x})$

1: **for** iteration $t = 1, 2, ..., T$ **do**
2: **for** $n = 1, 2, ..., N$ **do**
3: compute the gradient of the loss L w.r.t. the current ensemble score $F(\mathbf{x}_n)$: $g_n = \frac{\partial L(y_n, F(\mathbf{x}_n))}{\partial F(\mathbf{x}_n)} = p(y = 1|\mathbf{x}_n) - y_n$
4: **end for**
5: fit a regression tree h_t to the regression dataset $\{(\mathbf{x}_n, -g_n)\}_{n=1}^N$
6: shrink the regression tree leaf output values by ρ: $h_t \leftarrow \rho h_t$
7: add the new regression tree h_t to the ensemble: $F = h_1 + h_2 + \cdots + h_t$
8: **end for**

Output: final ensemble $F = h_1 + h_2 + \cdots + h_T$

2.3.5 Multi-Class Gradient Boosting Classifier

Gradient Boosting can also be applied to multi-class classification. It maintains K tree ensembles $F_1, F_2, ..., F_K$ for a K-class problem. Each ensemble F_k is responsible for providing the score $F_k(\mathbf{x})$ for a class k.

$$F_k(\mathbf{x}) = h_{k1}(\mathbf{x}) + h_{k2}(\mathbf{x}) + \cdots + h_{kT}(\mathbf{x}), k = 1, 2, ..., K \tag{2.54}$$

Similar to multi-class LR, multi-class GB computes the class probabilities using the softmax transformation

$$p(y = k|\mathbf{x}) = \frac{e^{F_k(\mathbf{x})}}{\sum_{j=1}^K e^{F_j(\mathbf{x})}} \tag{2.55}$$

Given a general multi-class dataset $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ with soft target labels ($y_{nk} \in [0, 1]$ and $\sum_{k=1}^K y_{nk} = 1$), GB can be trained by minimizing the KL-divergence between the target distribution and the predictive distribution, defined as $\mathbb{KL}(y_n||p(y|\mathbf{x}_n)) = \sum_{k=1}^K y_{nk}[\log y_{nk} - \log p(y = k|\mathbf{x}_n)]$. Notice that multi-class GB and LR have the same training objective and similar design. The major difference is that in LR each class score is a liner function of the features while in GB it is a non-linear function implemented by a tree ensemble. In this sense, multi-class GB can be viewed as a non-linear generalization of multi-class LR.

The multi-class GB training procedure is derived from the generic GB training procedure Algorithm 2.4. But since multi-class GB contains K ensembles, the training needs to have an additional loop over ensembles. The overall training procedure is summarized in Algorithm 2.7.

Algorithm 2.7 Multi-Class Gradient Boosting Classifier Training [40]

Input: multi-class dataset $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$, initial models $F_1, F_2, ..., F_K$

for iteration $t = 1, 2, ..., T$ **do**
  for $k = 1, 2, ..., K$ **do**
    for $n = 1, 2, ..., N$ **do**
      compute the gradient of the loss L w.r.t. the current ensemble score $F_k(\mathbf{x}_n)$: $g_{nk} = \frac{\partial L(y_n, F_1(\mathbf{x}_n), F_2(\mathbf{x}_n), ..., F_K(\mathbf{x}_n))}{\partial F_k(\mathbf{x}_n)} = p(y = k|\mathbf{x}_n) - y_{nk}$
    end for
    fit a regression tree h_{kt} to the regression dataset $\{(\mathbf{x}_n, -g_{nk})\}_{n=1}^N$
    shrink the regression tree leaf output values by ρ: $h_{kt} \leftarrow \rho h_{kt}$
    add the new regression tree h_{kt} to the ensemble F_k: $F_k = h_{k1} + h_{k2} + \cdots + h_{kt}$
  end for
end for

Output: final K ensembles $F_1, F_2, ..., F_K$

2.3.6 Isotonic Regression

Isotonic regression [98] is a technique for fitting a non-decreasing curve to a sequence of observations. Given a sequence of one dimensional inputs $x_1 \leq x_2 \leq \cdots \leq x_N$ and their labels $y_1, y_2, ..., y_N$, isotonic regression fits a curve $g(x)$ with the property $g(x_n) \leq g(x_{n+1})$. See Figure 2.1 for an illustration.

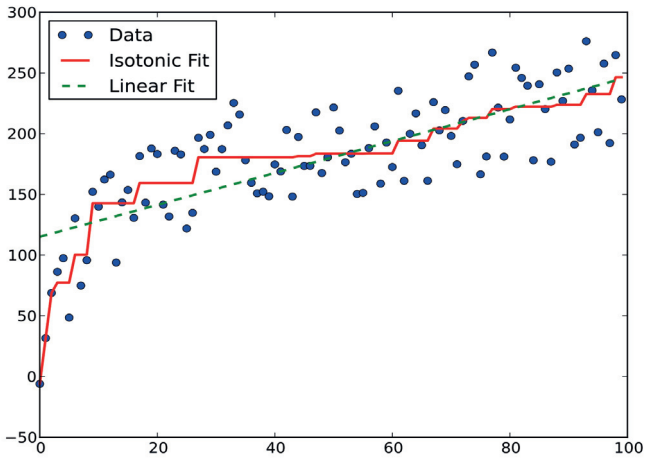


Figure 2.1: Isotonic regression vs. linear regression. By Alexeicolin - Own work, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=23732999`

Isotonic regression is trained by minimizing squared error

$$\min_{g} \sum_{n=1}^{N} (y_n - g(x_n))^2 \text{ subject to } g(x_n) \leq g(x_{n+1}) \tag{2.56}$$

This constrained optimization problem (2.56) can be solved efficiently in linear time using the pool-adjacent-violators (PAV) algorithm [13].

After training, isotonic regression can be used for prediction: for a new input x that lies between x_n and x_{n+1}, isotonic regression outputs the interpolated value[1] $g(x) = \frac{x-x_n}{x_{n+1}-x_n}[g(x_{n+1}) - g(x_n)] + g(x_n)$.

Isotonic regression is often used as a calibrator for classifiers. Consider a classifier that has made N predictions on a dataset. Let $s_1 \leq s_2 \leq \cdots \leq s_N$ be the (uncalibrated) confidence scores the classifier assigned to its predictions (assuming the predictions are already sorted by scores), and let $v_1, v_2, ..., v_N \in \{0, 1\}$ be the binary correctness of these predictions compared to ground truth. One could partition the inputs into K (e.g., 100) fixed size buckets and compute the average score x_k and average accuracy y_k in each bucket k. If the scores the classifier produced indicate the likelihood of making correct predictions, x_k should be close to y_k. If this is not the case, one can train an isotonic regression g on $x_1, x_2, \cdots, x_K$ with labels $y_1, y_2, \cdots, y_K$ and use g as a calibrator to map any uncalibrated score x to a calibrated one $g(x)$. Note that because of the square loss training objective, one can also directly train the calibrator g on $s_1, s_2, \cdots, s_N$ with labels $v_1, v_2, ..., v_N$ and obtain the same regression model.

[1] Another option is to output $g(x_n)$ and the resulting regression curve is piece-wise constant.

Chapter 3

Conditional Bernoulli Mixtures

In the previous chapter, we have described many different evaluation metrics for multi-label classification and also presented different prediction strategies that are tailored for different metrics. Some of these prediction methods only need the model to provide marginal label probabilities $p(y_\ell|\mathbf{x})$; while others require the model to estimate the joint probability among all labels $p(\mathbf{y}|\mathbf{x})$. For example, prediction methods designed for Hamming loss and label-averaged F1 score only performs marginal inference while prediction methods designed for set accuracy and instance F1 require joint inference. Clearly for a multi-label model to be generally useful for all types of inference, it needs to provide a joint probability estimation and it should also allow for efficient marginalization based on the joint. This chapter describes conditional Bernoulli mixture (CBM), a new probabilistic model for multi-label classification. It uses a mixture structure to estimate the joint distribution. This structure allows for efficient training, joint inference and marginal inference. CBM is also a reduction method: it transforms a multi-label classification problem to a multi-class classification problem and a series of binary classification problems. This allows CBM to use existing binary and multi-class models as building blocks.

3.1 Label Dependencies and Joint Estimations

The simplest approach to estimate the joint probability is to estimate the marginal probabilities and then take the product, i.e., $p(\mathbf{y}|\mathbf{x}) = \prod_{\ell=1}^{L} p(y_\ell|\mathbf{x})$, which effectively assumes conditional independence among labels and ignores label dependencies. This is the *Binary Relevance* (BR) [107] approach which reduces a multi-label problem into L independent (probabilistic) binary classification problems, and is

widely used due to its simplicity. Its disadvantage, as mentioned earlier, is that its predictions based only on marginal probabilities can be conflicting or incomplete. For example, a medical note may be predicted with both the code `Pain in left knee` and the code `Pain in unspecified knee`; an image may be tagged with `cat` but not `animal`. Such predictions may not seem very problematic if one uses instance F1 score, Hamming loss or Jaccard index as the evaluation metric – predicting 3 out of 4 codes right will give a F1 score of 0.86 (please refer to Section 2.1 for the definitions of these metrics). However, if one uses subset accuracy for evaluations, any predictions with minor mistakes will be deemed wrong and receive zero credit. In computing subset accuracy, a predicted subset is considered correct only when it matches the true subset exactly. Admittedly, this metric is very stringent in evaluation. However, optimizing for subset accuracy is a very interesting algorithmic design and research problem, as it encourages the classifiers to output coherent and complete predictions.

At the other extreme is the *Power-Set* (PowSet) approach [107], treating each label subset $\mathbf{y}$ as a class, and trains a (probabilistic) multi-class classifier. As a consequence, one would be restricted in practice to predicting only label subsets seen in the training data and always assigning zero probabilities to unseen subsets; even for many of the subsets observed there would likely be a scarcity of training data. Power-Set is handling label dependencies and its predictions are coherent, but the method is often infeasible on the exponential number of label sets.

3.2 Bernoulli Mixtures

Mixture models offer a flexible and powerful framework for general multivariate density estimation problems. A mixture generally has the form

$$p(\mathbf{y}) = \sum_{k=1}^{K} \pi^k p(\mathbf{y}; \boldsymbol{\beta}^k), \tag{3.1}$$

which approximates a complex joint $p(\mathbf{y})$ by a weighted combination of K component densities $p(\mathbf{y}; \boldsymbol{\beta}^k)$, each of which typically takes some simple density form parametrized by $\boldsymbol{\beta}^k$. The Expectation Maximization (EM) algorithm can be employed to train such mixture models by iterating between estimating the mixture memberships and fitting component densities.

Bernoulli mixtures (BM) are classic models for multi-dimensional binary variable density estimation [66, 76, 18], where the learnability is realized by assuming independence of variables within each mixture component. Thus each component

density $p(\mathbf{y}; \boldsymbol{\beta}^k)$ is simply a product of Bernoulli densities and the overall model has the form

$$p(\mathbf{y}) = \sum_{k=1}^{K} \pi^k \prod_{\ell=1}^{L} \mathrm{Ber}(y_\ell; \mu_\ell^k), \tag{3.2}$$

where $\mathrm{Ber}(y_\ell; \mu_\ell^k)$ denotes the Bernoulli distribution with head probability μ_ℓ^k. See Figure 3.1 for an illustration of a simple Bernoulli mixture with 3 components.

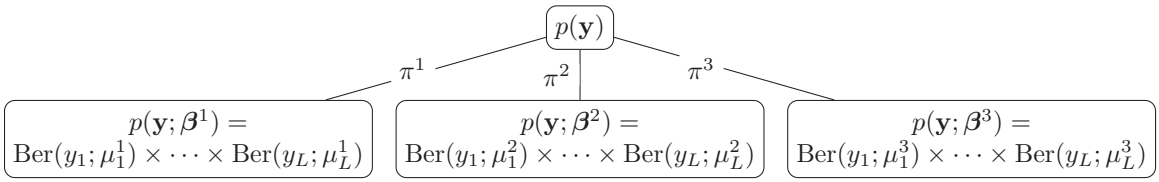


Figure 3.1: An illustration of Bernoulli mixture with 3 components.

The intuition behind Bernoulli mixture is that globally dependent labels can be less dependent in a given context. BM can be regarded as a soft clustering algorithm which groups correlated labels into the same cluster. The probability that label ℓ belongs to cluster k is μ_ℓ^k. For instance, Figure 3.2 shows three sample label clusters found by the BM algorithm on the MSCOCO image dataset [5]. In particular, the labels `car` and `truck` are globally highly correlated and they appear in the same cluster with high probabilities. If a person is told that an image contains a truck and is asked to guess what else might be in the same image, a reasonable guess could be a `car` since the topic of the image might be road traffic. Therefore knowing that there is a `truck` raises one's belief about `car`. Mathematically, $p(\texttt{car} \mid \texttt{truck}) > p(\texttt{car} \mid \text{no } \texttt{truck})$. However, if the person is already told that the topic of the image is road traffic, then whether the image

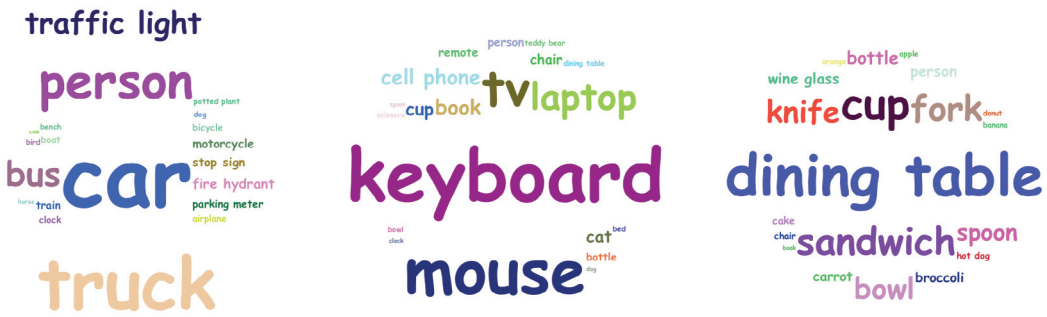


Figure 3.2: Three label clusters found by Bernoulli mixture on MSCOCO image dataset. For each label in a cluster, the font size is proportional to its probability of belonging to the cluster.

contains a `truck` or not should not influence his belief on `car`. Mathematically, $p(\texttt{car} \mid \texttt{truck}, \text{topic} = \text{road traffic}) \approx p(\texttt{car} \mid \text{no } \texttt{truck}, \text{topic} = \text{road traffic})$. In other words, the two globally dependent labels `car` and `truck` become roughly independently under this given topic, and this can be written mathematically as $\texttt{car} \perp \texttt{truck} \mid \text{topic} = \text{road traffic}$.

BM has a few attractive properties. First, *dependencies*: although the L variables are assumed to be independent inside each component, they are in general dependent in the mixture (they are forced to be independent only when $K = 1$). In other words, $p(\mathbf{y}) \neq \prod_{\ell=1}^{L} p(y_\ell)$. This can also be verified by computing the following covariance matrix and observing that it is non-diagonal for $K \geq 2$ [18].

$$\text{Cov}[\mathbf{y}] = \sum_{k=1}^{K} \pi^k [\boldsymbol{\Sigma}^k + \boldsymbol{\mu}^k (\boldsymbol{\mu}^k)^\top] - \mathbb{E}(\mathbf{y}) \mathbb{E}(\mathbf{y})^\top, \tag{3.3}$$

where $\mathbb{E}(\mathbf{y}) = \sum_{k=1}^{K} \pi^k \boldsymbol{\mu}^k$, $\boldsymbol{\Sigma}^k = \text{diag}\{\mu_\ell^k (1 - \mu_\ell^k)\}$, and $\boldsymbol{\mu}^k = (\mu_1^k, \mu_2^k, ..., \mu_L^k)^\top$. Second, *complexity*: specifying the full joint of L variables requires 2^L parameters and is unmanageable. By contrast, a BM model with K components uses $KL + K$ parameters to approximate the joint, and is far more manageable. Third, *efficiency*: the full factorization of each component makes fitting BM efficient via the EM algorithm.

In this work, we propose a new multi-label classification method which approximates the conditional joint $p(\mathbf{y}|\mathbf{x})$ based on Bernoulli mixtures. The method simultaneously *learns* the label dependencies from data and *trains* classifiers to account for such dependencies.

3.3 Conditional Bernoulli Mixtures

For multi-label classification, we model the conditional joint $p(\mathbf{y}|\mathbf{x})$ with a discriminative extension of BM, capturing conditional dependencies among binary labels given features. The analysis in [32] shows that labels could be largely conditionally independent given features (i.e., $p(\mathbf{y}|\mathbf{x}) \approx \prod_{\ell=1}^{L} p(y_\ell|\mathbf{x})$), even when labels are strongly marginally dependent (i.e., $p(\mathbf{y}) \neq \prod_{\ell=1}^{L} p(y_\ell)$), as long as each label is highly predictable from features. Therefore it is not necessary to capture correlations among easy-to-predict labels; it is sufficient to only capture those correlations which involve some hard-to-predict labels. Thus conditioning on features greatly reduces the need to estimate label correlations and makes learning much easier; *the conditional on* $\mathbf{x}$ *is essential for good approximation and effective training*,

without making prior assumptions about the form of label dependencies (as many other methods do).

Making both mixture coefficients and Bernoulli distributions conditional on $\mathbf{x}$, we obtain our proposed model, conditional Bernoulli mixtures (CBM):

$$p(\mathbf{y}|\mathbf{x}) = \sum_{k=1}^{K} \pi(z = k|\mathbf{x}; \boldsymbol{\alpha}) \prod_{\ell=1}^{L} b(y_\ell|\mathbf{x}; \boldsymbol{\beta}_\ell^k), \tag{3.4}$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}_\ell^k$ $(\ell = 1, 2, ..., L; k = 1, 2, ..., K)$ are model parameters to be learned, and z is a hidden categorical indicator variable, such that $z = k$ if the data point is assigned to component k. Here we use π and b to represent the conditional mixture membership distribution and the conditional binary label distribution, respectively.

CBM reduces a multi-label problem to a multi-class problem and several binary problems, approaching $p(\mathbf{y}|\mathbf{x})$ akin to *divide and conquer*: the categorical distribution $\pi(z|\mathbf{x}; \boldsymbol{\alpha})$ assigns each instance $\mathbf{x}$ to 1 out of K components probabilistically. Its goal is to divide the feature space into several regions such that each region only has weak conditional label correlations and can be approximated by a simple component. This gating function $\pi(z|\mathbf{x}; \boldsymbol{\alpha})$ can be instantiated by any multi-class classifier which provides probability estimations, such as a multinomial logistic regression with parameters $\boldsymbol{\alpha}$.

Inside each region k, the local conditional joint density is approximated by a product of conditional marginal densities. Every local binary label predictor $b(y_\ell|\mathbf{x}; \boldsymbol{\beta}_\ell^k)$ estimates the probability of getting label y_ℓ from component k for data point $\mathbf{x}$, and can be instantiated by any binary classifier which provides probability estimations, such as a binary logistic regression with parameters $\boldsymbol{\beta}_\ell^k$. All K components together with the gating function are learned jointly in order to break the global label correlation into simple forms.

On one extreme, if CBM has only one component (and hence $\pi(z|\mathbf{x}; \boldsymbol{\alpha})$ plays no role), all labels are conditionally independent and CBM degenerates to Binary Relevance. On the other extreme, if CBM assigns one component to each unique label subset and fixes each $b(y_\ell = 1|\mathbf{x}; \boldsymbol{\beta}_\ell^k)$ to be the corresponding binary constant (1 if label ℓ is in the subset and 0 otherwise), then the overall CBM model simply selects one label subset from all possible subsets, which is conceptually the same as the Power-Set approach. By varying the number of components and the complexity of each component, CBM can provide a continuous spectrum between these two extremes (see Figure 3.3). Just as Binary Relevance and Power-Set, CBM is purely a reduction method. The main advantage of reduction methods compared to new models specifically designed for multi-label problem is that the reduction approach

makes it easier to incorporate and reuse many well-developed multi-class and binary classifiers. In Section 3.4, we will demonstrate two different instantiations of CBM, one with logistic regressions, and the other with gradient boosted trees.

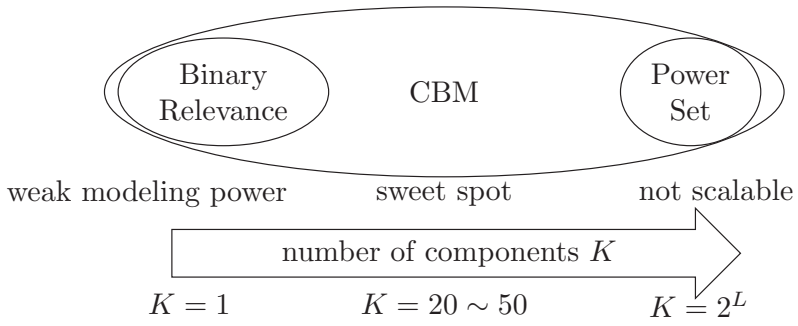


Figure 3.3: CBM vs. Binary Relevance vs. Power Set

An illustrative example. Before diving into the model training details, we illustrate how CBM performs joint label classifications with an example. Figure 3.4 shows a test image in the NUSWIDE dataset, for which the matched label `reflection` is missed by Binary Relevance but is captured by CBM.

For this image, the most influential components produced by CBM are shown in Figure 3.5. Simply averaging individual label probabilities by component mixing weights gives the conditional marginals $p(\texttt{lake}|\mathbf{x}) = 0.56$, $p(\texttt{water}|\mathbf{x}) = 0.69$, $p(\texttt{sunset}|\mathbf{x}) = 0.66$, and $p(\texttt{reflection}|\mathbf{x}) = 0.32$, which indicate that unlike `lake`, `water` and `sunset`, the label `reflection` by itself is not deemed as probable by CBM. However from the CBM joint density $p(\mathbf{y}|\mathbf{x})$ we can also infer Pearson correlation coefficients $\rho_{\texttt{reflection,lake}} = 0.50$, $\rho_{\texttt{reflection,water}} = 0.40$, $\rho_{\texttt{reflection,sunset}} = 0.17$ and observe that `reflection` is positively correlated with `lake`, `water`, and `sunset`. In fact, the joint probability asserts that the subset {`clouds, lake, sky, sunset, water, reflection`} is the most probable one, with probability 0.09. By contrast, the subset {`clouds, lake, sky, sunset, water`} has a lower probability 0.06. Therefore, although individually unlikely, the label `reflection` is correctly added to the mostly likely subset due to label correlations.

3.4 Training CBM with EM

CBM can be trained by maximum likelihood estimation on a given dataset $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$. Below we first derive a generic EM algorithm that works for any instantiation of the components, and then consider two concrete instantiations.

Figure 3.4: A test image from the NUSWIDE multi-label dataset [25] for which BR gives incorrect predictions and CBM gives correct predictions.. Independent binary logistic regressions predict the labels {`clouds, lake, sky, sunset, water`}. Our proposed method correctly predicts {`clouds, lake, sky, sunset, water, reflection`}, because it captures the dependencies between `reflection` and the other labels.

To make the mathematical derivation clear, we use different symbols for random variables and values of random variables. We use $\mathbf{Y}_n$ when treating the labeling of $\mathbf{x}_n$ as an unknown random variable and $\mathbf{y}_n$ when referring to its specific labeling assignment given in the training set. The likelihood for the dataset is

$$\prod_{n=1}^{N}\{\sum_{k=1}^{K}[\pi(z_n=k|\mathbf{x}_n;\boldsymbol{\alpha})\prod_{\ell=1}^{L}b(y_{n\ell}|\mathbf{x}_n;\boldsymbol{\beta}_\ell^k)]\}.$$

Since the model contains hidden variables, we use EM to minimize an upper bound of the negative log likelihood. Denoting the posterior membership distribution $p(z_n|\mathbf{x}_n,\mathbf{y}_n)$ as $\Gamma(z_n)=(\gamma_n^1,\gamma_n^2,...,\gamma_n^K)$, the upper bound can be written as

$$\begin{aligned}&\sum_{n=1}^{N}\mathbb{KL}(\Gamma(z_n)||\pi(z_n|\mathbf{x}_n;\boldsymbol{\alpha}))\\&+\sum_{k=1}^{K}\sum_{\ell=1}^{L}\sum_{n=1}^{N}\gamma_n^k\mathbb{KL}(\mathrm{Ber}(Y_{n\ell};y_{n\ell})||b(Y_{n\ell}|\mathbf{x}_n;\boldsymbol{\beta}_\ell^k)),\end{aligned}\tag{3.5}$$

where $\mathrm{Ber}(Y_{n\ell};y_{n\ell})$ is the Bernoulli distribution with head probability $y_{n\ell}$.

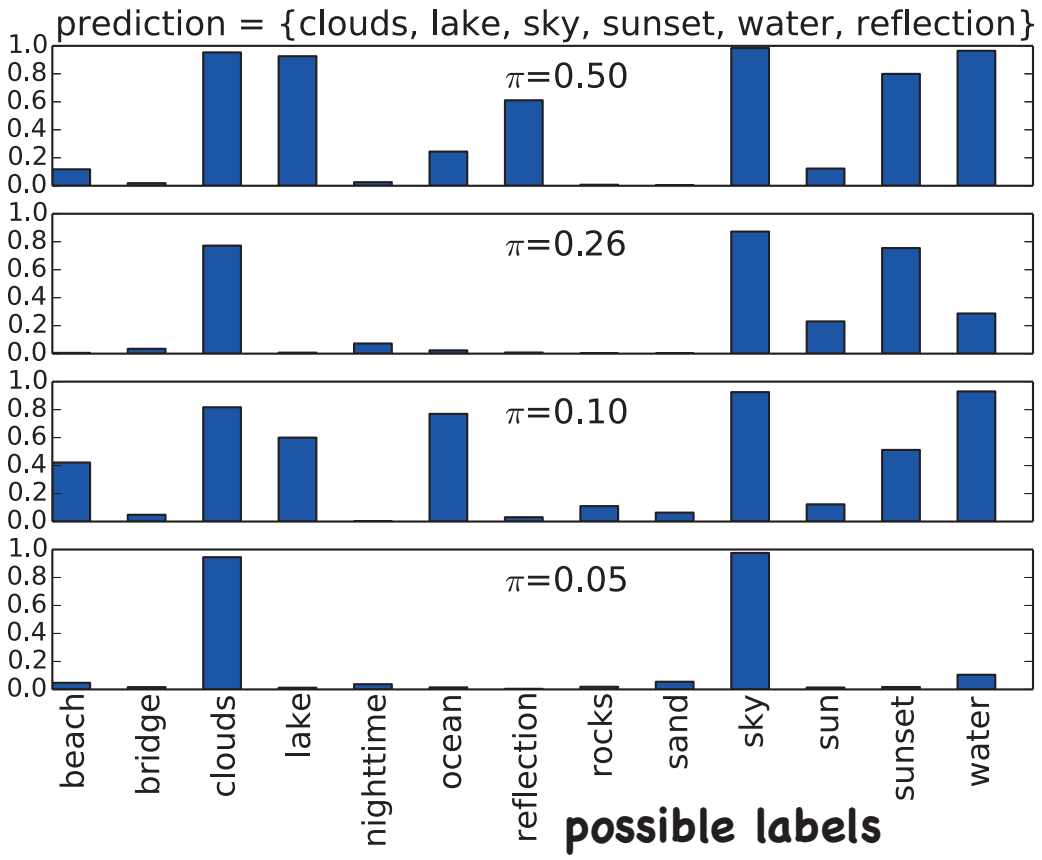


Figure 3.5: The top 4 most influential components for the test image in Figure 3.4. π values indicate component mixing coefficients. Each bar represents an individual label probability in one component. Labels with near zero probabilities are not displayed here.

E Step: Re-estimate the posterior membership probability of each data point belonging to each component:

$$\gamma_n^k = \frac{\pi(z_n = k|\mathbf{x}_n;\boldsymbol{\alpha}) \prod_{\ell=1}^{L} b(y_{n\ell}|\mathbf{x}_n;\boldsymbol{\beta}_\ell^k)}{\sum_{j=1}^{K} \pi(z_n = j|\mathbf{x}_n;\boldsymbol{\alpha}) \prod_{\ell=1}^{L} b(y_{n\ell}|\mathbf{x}_n;\boldsymbol{\beta}_\ell^j)}. \tag{3.6}$$

M Step: Update all model parameters. The bound (3.5) shows that the optimization of all parameters can be nicely decomposed into a series of separate optimization problems:

$$\boldsymbol{\alpha}_{new} = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \sum_{n=1}^{N} \mathbb{KL}(\Gamma(z_n)||\pi(z_n|\mathbf{x}_n;\boldsymbol{\alpha})), \tag{3.7}$$

$$\boldsymbol{\beta}_{\ell\,new}^k = \underset{\boldsymbol{\beta}_\ell^k}{\operatorname{argmin}} \sum_{n=1}^{N} \gamma_n^k \mathbb{KL}(\mathrm{Ber}(Y_{n\ell};y_{n\ell})||b(Y_{n\ell}|\mathbf{x}_n;\boldsymbol{\beta}_\ell^k)). \tag{3.8}$$

The optimization problem defined in (3.7) is a multi-class classification problem with target class distribution (soft labels) $\Gamma(z_n) = (\gamma_n^1, \gamma_n^2, ..., \gamma_n^K)$ for $\mathbf{x}_n$. The training goal for $\pi(z|\mathbf{x}; \boldsymbol{\alpha})$ is to assign (probabilistically) each data point to a component based only on its features, such that the assignment matches the posterior component membership determined by both features and true labels.

The optimization problem defined in (3.8) is a weighted binary classification problem. $\mathbf{x}_n$ has target label $y_{n\ell}$ and weight γ_n^k. The predictor $b(y_\ell|\mathbf{x}; \boldsymbol{\beta}_\ell^k)$ is trained as a binary classifier for label ℓ in component k, based only on training data within (soft) component k.

To train CBM, we iterate between the E step and the M step, until the upper bound (3.5) converges (see Algorithm 3.1). In practice, the EM algorithm can

Algorithm 3.1 Generic Training for CBM

repeat
  E Step
  for $n = 1, 2, ..., N$; $k = 1, 2, ..., K$ **do**
    update γ_n^k as in (3.6)
  end for
  M Step
  update $\boldsymbol{\alpha}$ as in (3.7)
  for $k = 1, 2, ..., K$; $\ell = 1, 2, ..., L$ **do**
    update $\boldsymbol{\beta}_\ell^k$ as in (3.8)
  end for
until convergence

get stuck in local optima, so careful initializations are often necessary for good performance. Due to the natural connection between CBM and BM, one simple way of initializing CBM is to first fit a BM just for label density estimation without looking at features. BM can be trained very quickly by a simpler EM algorithm [18]. We can use several random starts for a set of BM models, and select the one with the best training objective score to initialize the CBM training procedure.

3.4.1 CBM with Logistic Regression Learners

In the simplest form all underlying models in CBM are linear. We employ a multinomial logistic regression for the gating function π, and a binary logistic regression for each predictor b. The outer loop of the training procedure is the EM algorithm described above. In each M step, all objectives defined in (3.7) and (3.8)

are convex (although the overall EM optimization is non-convex). To optimize each logistic regression, we compute the gradient of the corresponding objective w.r.t. its parameters and update all parameters using gradient based optimization methods. Here we choose L-BFGS method [85], which is the state-of-the-art optimization method for large scale logistic regressions. We can start with the existing model, and perform a few incremental update steps until it converges. To avoid over-fitting, we also add L_2 regularizations (Gaussian priors) to all parameters.

CBM+LR complexity. The overall complexity of the training procedure for CBM is dominated by the updates in M steps and depends on the complexities of the binary and multi-class learning algorithms used. For CBM with logistic regression (LR) learners, we can measure the complexity of each L-BFGS (or gradient descent) update for all model parameters. If N is the number of instances, D the number of features, L the number of labels, C the number of unique label subsets in the training set, and K the number of CBM components, then each CBM+LR update takes $\mathcal{O}(KLDN)$. For comparison, each update takes $\mathcal{O}(LDN)$ in Binary Relevance with LR, and takes $\mathcal{O}(CDN)$ in Power-Set with LR. For large datasets, typically $KL < C$, and CBM is slower than Binary Relevance but faster than Power-Set. In Section 3.5, we will develop a few tricks to speed up CBM training so that its complexity does not grow linearly with K.

3.4.2 CBM with Gradient Boosting Learners

Many datasets require non-linear decision boundaries, in which case the CBM model with logistic regressions may not have enough explanation power. To make CBM non-linear, we use gradient boosted trees (GB) for both π and b. The original gradient boosted trees algorithm described in [40] is designed for standard multi-class problems, and does not take label distributions or instance weights as inputs, thus some modifications are necessary. The target label distribution is easy to deal with: one still computes the functional gradient of the objective w.r.t. each ensemble scoring function, where the objective is defined by the target distribution. To handle instance weights, one ignores them when calculating functional gradients, and performs a weighted least squares fit when fitting each regression tree to the gradients. Unlike logistic regression, where all parameters can be easily updated, gradient boosting introduces new trees to the ensemble while keeping old trees untouched. Thus the M step here is slightly different from before: rather than re-adjusting all parameters to optimize the objective, boosting improves the objective by adding a few more trees. This amounts to a partial M step, and the resulting EM algorithm is usually called the generalized EM algorithm [51]. We emphasize that the

use of boosting as an underlying non-linear classifier here is just for demonstration purposes. Other classifiers such as neural networks could also be used. This is in direct contrast to AdaBoost.MH and Adaboost.MR [100] which specifically employ boosting to optimize Hamming loss and rank loss.

3.5 Speeding up Training with Sparse Structures

One of the intended application of the developed method is large scale multi-label text classification, in which the number of labels/features/instances can be enormous. The original CBM training and prediction complexity grows linearly with the number of labels and the number of mixture components, which could be prohibitively large. Also storing many mixture components each with many classifiers can be difficult.

We seek to reduce the complexity with two approximations that leverage the sparse structure naturally arising in CBM.

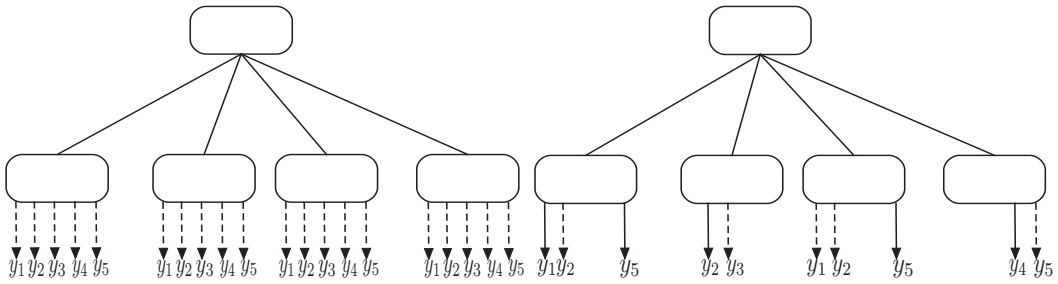


Figure 3.6: CBM with dense structure vs. CBM with sparse structure. Mixture components are the leafs; dashed arrows represent binary label classifiers; solid arrow are labels always predicted in the component. Missing arrows indicate labels never predicted in the component.

The first approximation is to skip some instances in each binary classifier training. We say an instance is active in a component if its membership degree γ_n^k is not close to 0 (greater than some small threshold δ). When the number of components is large enough, it is often the case that some γ_n^k values are extremely close to 0 (but are not exactly 0 due to the softmax operation in the multi-class classifier π). Notice that γ_n^k acts as an instance weight in the binary classifier training. Therefore instances with very small γ_n^k have very little impact on the actual classifier training and can be safely skipped. So for each binary classifier in a component k, we only train it on the instances that are active in the component.

The second approximation is to skip some binary classifiers in each component.

We say a label ℓ is active in a component k if the fraction of instances in the component matching this label is not close to either 0 or 1, i.e., $\epsilon < \sum_{n:y_{nl}=1} \gamma_n^k < 1-\epsilon$, for some small constant ϵ. If $\sum_{n:y_{nl}=1} \gamma_n^k <= \epsilon$, then effectively no data in the component contains this label. If $\sum_{n:y_{nl}=1} \gamma_n^k >= 1-\epsilon$, then effectively all data in the component contains this label. In both cases, there is no need to train a dedicated classifier – constant outputs suffice. In practice, when the number of components is large enough, only a few labels are actually active in each component (see Figure 3.2). Most of the labels almost never occur; while a few others may almost always occur. Therefore, the number of binary classifiers we need to train in each component is far smaller than the total number of labels L. See Figure 3.6 for an illustration.

Analyzing the theoretical complexity of the sparse CBM with these approximation tricks is not straightforward. Here we provide some empirical results. Our preliminary experiments show that by only allocating classifiers for active labels greatly speeds up training and prediction without damaging accuracy. On the RCV1-2K dataset with 2,456 labels, the dense CBM with 50 components contains 122,800 binary classifiers, while the sparse CBM can skip 107,643 binary classifiers and only train the remaining 15,157 classifiers. That reduces both the training time and the memory usage by 90%. We also notice that training a sparse CBM with many components can sometime be even faster than training a BR model, because with many components, each binary classifier training is done on a subset of data within the component, and thus can be much faster. On RCV1-2K dataset, training sparse CBM is 3 times faster than training BR.

3.6 Making Predictions to Optimize Set Accuracy

So far we have described CBM's model structure and its training method. After training is done, CBM produces a joint distribution $p(\mathbf{y}|\mathbf{x})$, which allows us to perform various inferences. We are particularly interested in prediction methods that optimize a given evaluation metric. In Section 3.6, we derive a dynamic programming based prediction method to optimize set accuracy. In Section 3.7, we feed the joint distribution to an existing procedure named GFM to optimize instance-averaged F1. In Section 3.8, we perform marginal based prediction to optimize Hamming loss.

According to [32], making the optimal prediction in terms of subset accuracy for a given $\mathbf{x}$ requires finding the most probable label subset $\mathbf{y}^* = \text{argmax}_{\mathbf{y}}\, p(\mathbf{y}|\mathbf{x})$. There are 2^L label subset candidates, and it is intractable to evaluate the probability for each of them. Many multi-label methods suffer from this intractability for exact

inference (see Section 3.11). Fortunately for CBM, its special structure allows it to make predictions efficiently, with either *sampling* or *dynamic programming*. As a common preprocessing step, for a given $\mathbf{x}$, we can first compute $\pi^k = \pi(z = k|\mathbf{x}; \boldsymbol{\alpha})$ and $\mu_\ell^k = b(y_\ell = 1|\mathbf{x}; \boldsymbol{\beta}_\ell^k)$, for all $k = 1, 2, .., K$ and $\ell = 1, 2, ..., L$.

3.6.1 Prediction by Sampling

The CBM density form (3.4) suggests a natural sampling strategy for a label subset $\mathbf{y}$. We first sample a component k according to the mixture coefficients $\pi^1, ..., \pi^K$. Then from this component, we sample each label y_ℓ independently with probability μ_ℓ^k. The procedure can be repeated multiple times to generate a set of $\mathbf{y}$ candidates, from which we pick the most probable one. The sampling strategy works best when the most probable label subset has a high probability. Sampling is easy to implement, but does not guarantee that the predicted $\mathbf{y}$ is indeed the optimal one.

3.6.2 Prediction by Dynamic Programming

In order for the overall probability $p(\mathbf{y}|\mathbf{x})$ to be high, there must exist a component k for which the component probability $\prod_{\ell=1}^{L} b(y_\ell|\mathbf{x}; \boldsymbol{\beta}_\ell^k)$ is high. On the other hand, one can show that the $\mathbf{y}^*$ maximizing the overall probability does not necessarily maximize any component probability. To find $\mathbf{y}^*$, we design a dynamic programing procedure FIND-NEXT-HIGHEST() that enumerates label subsets in a decreasing probability order in each component, and then we iterate round-robin across components until we are certain that the unchecked subsets will never produce a high overall probability. We define an operation FIND-NEXT-HIGHEST() as in Algorithm 3.2. For a component k, the $\mathbf{y}$ with the highest component probability is the set containing precisely all the ℓ with $\mu_\ell^k \geq 1/2$. To produce a ranked list, we

Algorithm 3.2 Find-Next-Highest()

1: $\mathbf{y} = Q^k.deque()$
2: **for** $l = 1, 2, ..., L$ **do**
3: Generate a new candidate $\mathbf{y}'$ by flipping the l-th bit of $\mathbf{y}$
4: **if** $\mathbf{y}'$ has not been added to Q^k before **then**
5: $Q^k.enqueue(\mathbf{y}')$
6: **end if**
7: **end for**
8: **Output: y**

use a max priority queue Q^k to store candidate label subsets and their associated component probabilities. Initially, Q^k contains the $\mathbf{y}$ with the highest component probability. The t-th call to Q^k.FIND-NEXT-HIGHEST() returns the $\mathbf{y}$ with the t-th highest component probability.

The overall prediction algorithm (Algorithm 3.3) iterates over all components, checks the next candidate recommended by each component (Lines 9-10), updates the best candidate found so far (Lines 11-13), computes an upper bound for the probability of any unvisited candidate (Lines 14-15). The algorithm terminates when no remaining candidate can possibly have a higher probability than the existing best candidate. In practice, we observe that the algorithm rarely visits elements deeper than rank 10 in the component ranked lists.

Algorithm 3.3 Prediction by Dynamic Programming

1: **Input:** π^k, μ_ℓ^k, $k = 1, 2, ..., K$, $\ell = 1, 2, ..., L$
2: Initialize the maximum overall probability $M = -\infty$
3: **for** $k = 1, 2, ..., K$ **do**
4: Initialize the latest component probability $G^k = +\infty$
5: Initialize the priority queue Q^k
6: **end for**
7: **while true do**
8: **for** $k = 1, 2, ..., K$ **do**
9: $\mathbf{y} = Q^k$.FIND-NEXT-HIGHEST()
10: Let $p = \sum_{m=1}^{K} \pi^m \prod_{\ell=1}^{L} \text{Ber}(y_\ell; \mu_\ell^m)$
11: **if** $p > M$ **then**
12: Set $M = p$ and $\mathbf{y}^* = \mathbf{y}$
13: **end if**
14: Set $G^k = \prod_{\ell=1}^{L} \text{Ber}(y_\ell; \mu_\ell^k)$
15: Compute threshold $\tau = \sum_{m=1}^{K} \pi^m G^m$
16: **if** $M \geq \tau$ **then**
17: **Output** $\mathbf{y}^*$ and **terminate**
18: **end if**
19: **end for**
20: **end while**

Algorithm 3.3 is a special case of the well-known Threshold algorithm [36], which finds the globally best candidate by aggregating several sorted lists[1]. The

[1]Our first version of Algorithm 3.3 published in [68] uses a different stop condition, which makes the algorithm less efficient than the current version. Both versions produce identical outputs.

Threshold algorithm has the following general setup: each candidate is given K scores by K subsystems, and the overall score of each candidate is computed by a fixed monotonic aggregation function of these K scores. Each subsystem supports both sequential access (where all candidates are visited in decreasing score order) and random access (where the score of a given candidate is requested). In our CBM prediction task, each candidate is a label set, each subsystem is a mixture component, each score is a component probability, and the aggregation function is the weighted average where the weights are mixture coefficients. The dynamic programming algorithm (Algorithm 3.2) provides the sorted lists.

Dealing with empty predictions. Predicting empty label subsets could be undesirable when it is known a priori that each instance matches at least one label. The dynamic programming prediction algorithm can be easily modified to output the most probably non-empty subsets. In our experiments, we allow CBM to predict empty sets only when the training set contains empty sets. This strategy is shown to improve the test performance slightly. Occasionally, this could make CBM with 1 component perform slightly differently from BR.

3.6.3 Experiment Results

We perform experiments on five commonly used multi-label datasets: SCENE, TMC2007, MEDIAMILL, NUSWIDE and RCV1. For the sake of reproducibility, we adopt the train/test splits provided by the datasets. Datasets details are provided in Appendix A.

We compare conditional Bernoulli mixtures (CBM) with the following methods which estimate $p(\mathbf{y}|\mathbf{x})$: Binary Relevance (BR), Power-Set (PowSet), Classifier Chains (CC), Probabilistic Classifier Chains with Beam Search (PCC), Ensemble of Classifier Chains with label voting (ECC-label) and subset voting (ECC-subset), Conditional Dependency Networks (CDN) and pair-wise Conditional Random Fields (pairCRF). See Appendix B for the implementations used. For all methods which require a base learner, we employ logistic regression as a common base learner. Additionally, we test gradient boosted trees (GB) as a non-linear base learner in conjunction with BR, PowSet and CBM. Both LR and pairCRF are L_2 regularized. Hyper parameter tuning is done by cross-validation on the training set (see Appendix C for details). For methods involving random initializations or sampling, the reported results are averaged over 3 runs.

Test subset accuracy on five datasets is shown in Table 3.1, grouped by the base learner. On four datasets, the highest subset accuracy is achieved by one of

Table 3.1: Comparison between CBM and other methods in terms of set accuracy. All numbers are in percentages. Best performances are bolded.

Method	Learner	SCENE	RCV1	TMC2007	MEDIAMILL	NUSWIDE
BR	LR	51.5	40.4	25.3	9.6	24.7
PowSet	LR	68.1	**50.2**	28.2	9.0	26.6
CC	LR	62.9	48.2	26.2	10.9	26.0
PCC	LR	64.8	48.3	26.8	10.9	26.3
ECC-label	LR	60.6	46.5	26.0	11.3	26.0
ECC-subset	LR	63.1	49.2	25.9	11.5	26.0
CDN	LR	59.9	12.6	16.8	5.4	17.1
pairCRF	linear	68.8	46.4	28.1	10.3	26.4
CBM	LR	69.7	49.9	**28.7**	13.5	**27.3**
BR	GB	59.3	30.1	25.4	11.2	24.4
PowSet	GB	**70.5**	38.2	23.1	10.1	23.6
CBM	GB	**70.5**	43.0	27.5	**14.1**	26.5

the CBM instantiations; on the other dataset, CBM is close to the best one. This demonstrates the effectiveness of CBM for optimizing subset accuracy. On the SCENE and MEDIAMILL datasets, CBM+GB performs better than CBM+LR, which shows the benefit of being able to incorporate non-linear components. By contrast, pairCRF is restricted to work with linear functions, lacking the flexibility of CBM.

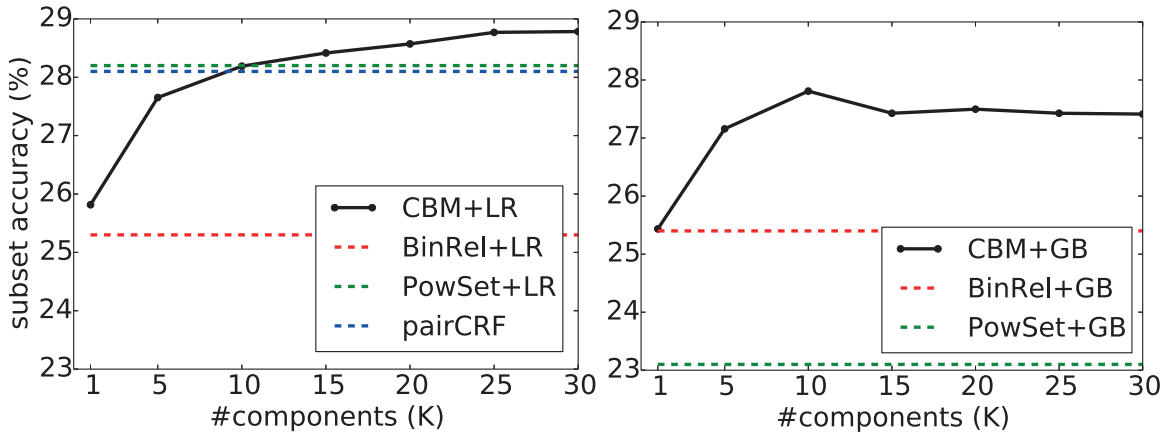


Figure 3.7: Test subset accuracy on TMC dataset with varying number of components K for CBM+LR (left) and CBM+GB (right) compared with BR, PowSet, and pairCRF.

Impact of number of components. Figure 3.7 shows how increasing the number of components K affects test performance for CBM on TMC dataset. When $K = 1$, CBM only estimates marginals and is equivalent to BR (the slight difference in performance is due to the handling of empty set in CBM prediction). Increasing K from 1 to 15 makes CBM quickly become a better joint estimator and outperform BR. Increasing K further gives smaller improvement for CBM and the performance asymptotes as K reaches about 30. Other datasets show similar trends.

3.7 Making Predictions to Optimize F1 Score

In practice, and particularly in industry, the instance-averaged F-measure—which gives partial rewards for subset predictions based on overlap with the correct subset—is much better suited for many label set classification tasks than strict subset-accuracy (see Section 2.1 for the precise definitions of these metrics). For example, in a medical note, a patient may present with multiple illnesses or undergo a procedure with multiple billing codes; predicting five out of six codes correctly is a considerable help to medical billing systems. Multi-label competitions organized by industrial companies, such as the Yelp business categorization challenge [123] and the Greek Media challenge [116], employ F-measure for evaluation. In this section, we take the joint estimation $p(\mathbf{y}|\mathbf{x})$ provided by CBM and develop a different prediction algorithm that seek to optimize instance F1 as opposed to set accuracy.

3.7.1 Prediction with GFM

We rely on the General F-measure Maximizer (**GFM**) [112] algorithm described earlier in Section 2.2 (see Algorithm 2.1) to find the prediction $\mathbf{y}^*$ which maximizes the expected F1-measure:

$$\begin{aligned}\mathbf{y}^* &= \arg\max_{\mathbf{y}'} \mathbf{E}_{\mathbf{y}\sim p(\mathbf{y}|\mathbf{x})}[F(\mathbf{y}, \mathbf{y}')] \\ &= \arg\max_{\mathbf{y}'} \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \cdot F(\mathbf{y}, \mathbf{y}') \end{aligned} \tag{3.9}$$

where $\mathbf{y}$ is the (unknown) ground truth label vector, $\mathbf{y}'$ is a candidate prediction vector and $F(\mathbf{y}, \mathbf{y}') = \frac{2\sum_{l=1}^{L} y_l y'_l}{\sum_{l=1}^{L} y_l + \sum_{l=1}^{L} y'_l}$ is the F1 score of the prediction. Note that the ground truth $\mathbf{y}$ is unknown at prediction time, and is treated as a random variable whose distribution $p(\mathbf{y}|\mathbf{x})$ is estimated by the classifier.

The direct input to GFM is not a joint estimation $p(\mathbf{y}|\mathbf{x})$, but rather, some

marginal distributions of the form $p(y_l = 1, |\mathbf{y}| = s \mid \mathbf{x}),\ \forall l, s \in \{1, ..., L\}$, where $|\mathbf{y}|$ stands for the number of relevant labels in $\mathbf{y}$. This formula can be read as, for example, "the probability of the given document having $s = 5$ relevant labels and `cat` is one of them". There are (no more than) L^2 probabilities in this form, per instance. The paper [112] proposed two ways of obtaining these L^2 marginals probabilities: (1) a model which directly estimates L^2 marginals from data, and (2) the use of a probabilistic joint classifier/estimator $p(\mathbf{y}|\mathbf{x})$ and sampling to generate the required L^2 probabilities.

Support inference. We follow the idea (2) but make a critical change: after training the joint estimator $p(\mathbf{y}|\mathbf{x})$, we derive the required L^2 marginals using support inference as opposed to sampling. Sampling is easy when one could take advantage of the classifier structure: For BR each label can be sampled independently; in PCC one can sample labels one by one, using ancestral sampling; in CBM one can first sample a component, and from the component sample each label independently. However sampling is ineffective for large L and low-confidence ("flat") joint density that spreads probability mass over many label combinations.

Support inference only considers probabilities of label combinations seen in training. We enumerate all seen label combinations, evaluate their probabilities and then marginalize. These marginals are then fed into GFM to produce the F1-optimal prediction. At first glance, support inference seems to have the limitation of not considering unseen label combinations. In reality this limitation only appears *during marginalization*, and is largely mitigated by GFM in the prediction step. It is not hard to show that although support inference only considers existing combinations, support inference + GFM can output unseen combinations. Thus support inference provides a regularized probability estimation by only assigning probability mass to observed combinations, and GFM takes this regularized probability estimation as the input and outputs F optimal prediction, which could potentially be an unobserved label combination. We observe this strategy to work remarkably well for many classifiers and datasets. Our proposed solution is simple to implement and highly effective.

Calibration. It is often the case that the probability estimations given by the classifiers are *uncalibrated*, meaning that the probabilities do not align well with the actual prediction accuracy. One can further calibrate these probabilities on a validation set using some calibration method such as Isotonic Regression [98]. For each instance $\mathbf{x}$ in the validation set, we use support inference to generate $p(y_l = 1, |\mathbf{y}| = s \mid \mathbf{x}),\ \forall l, s \in \{1, ..., L\}$. Then for each (l, s) pair we train an isotonic regression calibrator using the input scores $p(y_l = 1, |\mathbf{y}| = s \mid \mathbf{x})$ and the regression targets $\mathbb{I}[y_l = 1, |\mathbf{y}| = s \mid \mathbf{x}]$ from all $\mathbf{x}$ in the validation set. When making predictions

Table 3.2: Comparison between CBM+GFM and other methods in terms of F1 score. '-' indicates failed runs with 56 core and 256GB RAM.

Method	BIBTEX	IMDB	OHSUMED	RCV1	WISE	WIPO
BR+GFM	48.1	63.8	71.0	76.1	80.1	68.0
CRF+GFM	49.5	**67.1**	70.5	76.1	79.4	**72.5**
CBM+GFM	**50.4**	66.2	**72.6**	**78.7**	**81.5**	71.3
BR	39.8	59.6	63.6	73.8	72.8	69.5
CRF	46.5	63.0	66.4	74.4	77.7	70.3
CBM	45.3	62.2	69.5	77.3	79.8	69.6
LIFT	31.5	-	54.4	70.2	-	61.6
SPEN	39.0	61.1	61.7	65.3	-	65.9
PDsparse	40.7	62.3	67.3	75.0	74.5	67.5
CFT	23.5	-	-	53.5	-	62.7
CLEMS	42.5	-	52.6	72.4	-	67.1
LSF	43.9	59.8	65.0	73.6	76.7	71.1

on the test set, we use the Isotonic regression calibrated probabilities. We find that calibrating the L^2 marginal probabilities produced by support inference helps GFM make better predictions. Note that to speed up computation, many of the (l, s) pairs can be safely skipped. See the discussion on GFM in Section 2.2 for more details.

3.7.2 Experiment Results

We test the proposed CBM+GFM method on several datasets (details of these datasets are shown in Appendix A) and compare it with several baseline methods. We adopt the given train/test split whenever it is provided; otherwise we use a random 20% of the data as the test set. We regularize CBM with LR base learners by elastic-net penalty $\lambda\{\alpha||w||_1 + (1-\alpha)||w||_2^2\}$, and we tune the overall strength λ and the L1 ratio α. Experiment results are shown in Table 3.2.

First we compare two CBM prediction methods: predicting by maximum probability (the "CBM" row in the table) vs. predicting with GFM (the "CBM+GFM" row in the table). It is clear that plugging in GFM improves F1 score on all datasets. We did similar tests for BR and CRF and show that GFM also helps BR and CRF (see BR vs. BR+GFM and CRF vs. CRF+GFM). Among the three methods BR+GFM, CRF+GFM and CBM+GFM, our proposed method CBM+GFM achieves the overall best performance.

Next we compare our algorithm to another approach which directly estimates

the L^2 probabilities required by the GFM algorithm, as suggested in [112], and we shall refer to it as **LSF**. It may appear that if the end goal is to produce these L^2 probabilities as input to the GFM algorithm, it should be more straightforward to estimate these L^2 probabilities directly rather than going through a joint estimation $p(\mathbf{y}|\mathbf{x})$ first, which by itself is quite challenging (the joint evolves 2^L probabilities in general). We can estimate each $p(y_l = 1, |\mathbf{y}| = s \,|\, \mathbf{x})$ using a binary logistic regression. Another option is to estimate $p(y_l = 1|\mathbf{x})$ with a binary logistic regression, and then $p(|\mathbf{y}| = s \mid \mathbf{x}, y_l = 1)$ with another multinomial logistic regression and then multiply their probabilities. However, in practice, we observe that this direct estimation approach does not perform well (see Table 3.2). Our speculation is that directly predicting the number of relevant labels $|\mathbf{y}|$ by a classifier is a very hard and unnatural task. Each category here ("matching s labels") does not have a fixed meaning, like `sport` or `economy` do in typical topical classification. As "matching 2 labels" can mean many different things, for example (`sport`, `basketball`) but also (`business`, `industry`), it is hard to establish a relation between "matching 2 labels" and feature representation. This is especially so for linear models like logistic regression, where probabilities are monotonic functions of scores, which in turn are monotonic functions of features. We believe having a joint estimation first and then inferring these marginals from the joint is a more natural choice.

We also tested many other methods and found them to perform worse than our proposed method. We describe these methods below and list their results in Table 3.2.

The **PD-Sparse** method [124] is recently proposed for extremely large scale multi-label classification. It employs a Dual Fully-Corrective Block-Coordinate Frank-Wolfe algorithm that exploits both primal and dual sparsity to achieve high efficiency. However, PD-Sparse only computes a non-probabilistic score for each label and ranks labels by scores. It does not offer a straightforward way of predicting a set of labels for each instance. The original implementation provided by the authors ask the users to provide the desired number of labels per instance and returns the top labels with highest scores as predictions. Because the correct number of labels varies greatly from instance to instance, predicting a fixed number of labels for all instances results in sometimes low precision (when the specified number of labels is more than necessary), sometimes low recall (when the specified number of labels is less than necessary), and overall low F1-measure. Since PD-Sparse does not provide probability estimations, GFM cannot be plugged in to predict optimal F1. We tried to make the PD-Sparse predictions more adaptive by tuning the threshold of the label scores to maximize the F1-measure, but PD-Sparse still performs much worse than our proposed methods.

There are also several neural network based multi-label classification methods [15, 81, 26, 79]. We run the code associated with the recently proposed Structured Prediction Energy Networks (**SPEN**) [15] with carefully tuned hyper parameters as suggested by the authors and observe that SPEN's performance to be less competitive, possibly due to over-fitting in high dimensional data with neural network's high model capacity. It is worth mentioning that GFM is recently applied to convolutional neural networks to optimize F measure and the experiment confirms that GFM gives better performance than simple thresholding [29].

The **LIFT** algorithm [128] constructs features specific to each label by conducting clustering analysis on its positive and negative instances, and then performs training and testing by querying the clustering results. We run the code provided by the authors and follow the suggested hyper parameters. LIFT does not perform well and could not finish on two datasets.

There are several approaches that seek to optimize the F-measure directly during training. [91] provides an up-to-date overview on different F-measure maximization methods. [90] uses a graph-cut algorithm and has poor scalability on high dimensional text datasets. There are three methods that use a cost-sensitive approach to optimize F-measure score during training [69, 54, 87]. We tested the Condensed Filter Tree method (**CFT**) [69] and the cost-sensitive label embedding with multidimensional scaling method (**CLEMS**) [54] and found both to perform poorly and their training to be also slow. [42] studies F-measure maximization with conditionally independent label subsets. This method has a strong assumption which makes it hard to apply to real data.

3.8 Making Predictions to Optimize Hamming Loss

3.8.1 Prediction with Marginals

There are cases when the predictions only depends on the marginal probabilities. The theoretical results in [32, 61] show that the optimal prediction for Hamming loss is achieved by thresholding each label probability independently at 0.5. It is shown that Macro F1 and Micro F1 optimal inference also only depends on marginal probabilities.

There are also multi-label tasks where the goal is not to produce a set of labels, but rather, to rank all labels according to their relevance w.r.t. an instance, or to

rank all instances according to their relevance w.r.t. the query label. Both tasks only depends on the label marginal probability $p(y_l|\mathbf{x})$. Label ranking and instance ranking are usually evaluated with MAP. For CBM, the marginal probability can be easily computed as $p(y_\ell|\mathbf{x}) = \sum_{k=1}^{K} \pi(z = k|\mathbf{x}; \boldsymbol{\alpha}) b(y_\ell|\mathbf{x}; \boldsymbol{\beta}_\ell^k)$, and thus CBM can be readily used in tasks where marginal inference is needed.

3.8.2 Experiment Results

Here we focus on the set prediction task where the evaluation metric is Hamming loss. We test CBM with independent predictions by thresholding marginals and compare it with BR. Both CBM and BR use L2 regularized LR as base learners. The results are shown in Table 3.3. Interestingly CBM also achieves slightly better Hamming loss than BR on most of the datasets. Since both models make independent predictions based on marginals, the improvement CBM achieves is likely due to its increased model complexity which allows CBM to better estimate marginals. Estimating marginal probabilities with mixture models has also been reported to be effective in the large scale multi-label Youtube video classification task [10].

Table 3.3: Comparison between CBM and BR in terms of Hamming loss

	SCENE	RCV1	TMC2007	MEDIAMILL	NUSWIDE
CBM	**0.0877**	**0.0137**	0.0657	**0.0298**	**0.0202**
BR	0.1015	0.0143	**0.0649**	0.0309	0.0207

3.9 Comparison between Different CBM Prediction Methods

In Sections 3.6, 3.7 and 3.8 we have derived three different CBM prediction methods to optimize set accuracy, instance F1 and Hamming loss, respectively. For each prediction method, we have tested it on real datasets and compared it with other baselines. It would also be interesting to compare these three prediction methods themselves. Unfortunately due to different experimental setups, the results from previous sections are not directly comparable. In particular, CBM hyper parameters were tuned w.r.t. different metrics which led to different CBM models. In this section, we train only one CBM model (using L2 regularized LR as base learners) on each dataset and then vary only the prediction methods. The results are summarized

in Tables 3.4, 3.5, and 3.6. We can clearly see that the prediction method designed to optimize a particular metric indeed achieves the best performance w.r.t. that metric.

Table 3.4: Comparing different CBM prediction methods in terms of set accuracy

	SCENE	RCV1	TMC2007	MEDIAMILL	NUSWIDE
set-accuracy optimized	**0.6973**	**0.4986**	**0.2824**	**0.1306**	**0.2723**
F1 optimized	0.5944	0.4830	0.2659	0.1029	0.2255
Hamming loss optimized	0.6279	0.4730	0.2724	0.1098	0.2578

Table 3.5: Comparing different CBM prediction methods in terms of F1 score

	SCENE	RCV1	TMC2007	MEDIAMILL	NUSWIDE
set-accuracy optimized	0.7501	0.7577	0.6106	0.5070	0.3665
F1 optimized	**0.7709**	**0.7682**	**0.6368**	**0.5786**	**0.4366**
Hamming loss optimized	0.6762	0.7406	0.5993	0.5471	0.3685

Table 3.6: Comparing different CBM prediction methods in terms of Hamming loss

	SCENE	RCV1	TMC2007	MEDIAMILL	NUSWIDE
set-accuracy optimized	0.0891	0.0138	0.0671	0.0343	0.0211
F1 optimized	0.1018	0.0139	0.0690	0.0329	0.0244
Hamming loss optimized	**0.0877**	**0.0137**	**0.0657**	**0.0298**	**0.0202**

Figure 3.8 provides a conceptual summary of different CBM prediction methods designed for different metrics. In particular, to optimize Hamming loss and MAP which are based on marginal probabilities alone, one has two options. The first option is to infer the marginals from the joint estimated by CBM. The second option is to directly estimate marginals using BR. One should note, however, the BR model is not capable of optimizing set accuracy and F1, while CBM is capable of optimizing all the metrics listed here.

3.10 Running Time

To facilitate wider comparison of different methods, we also report different algorithms' training time and prediction time measured in our experiments. We test two versions of CBM: sparse CBM, which takes advantage of sparsity during training to skip certain labels and training instances (as described in Section 3.5), and dense

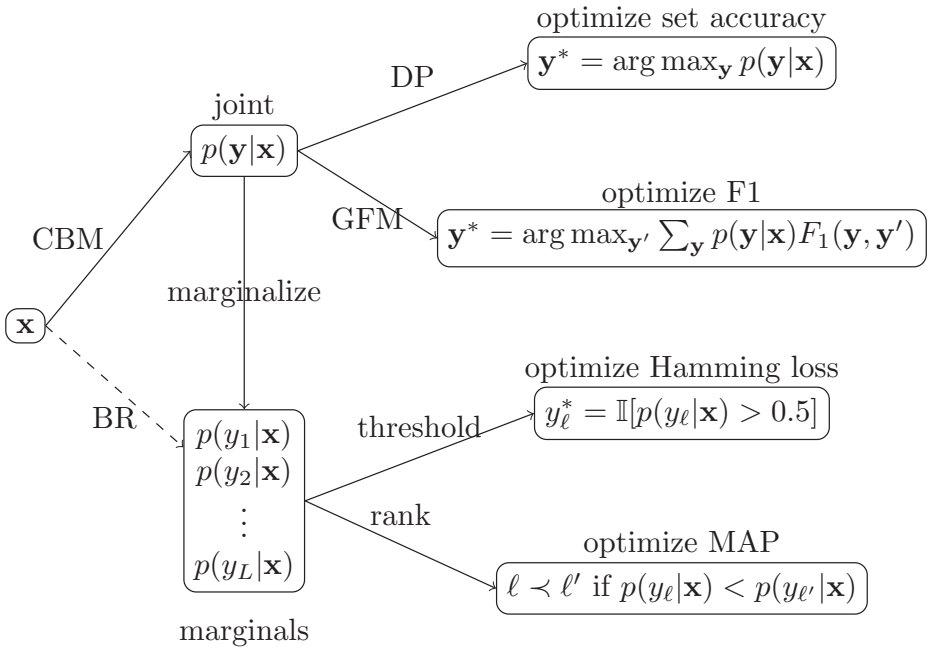


Figure 3.8: Different CBM prediction methods designed for different metrics.

CBM, which does not take advantage of sparsity during training. We use linear learners for all algorithms to make results comparable. Our Java implementations of BR, PowSet, CC, PCC, ECC, pairCRF and CBM are all multi-threaded. The CDN implementation is taken from the MEKA package [6] without further modification. Each experiment is conducted on a computer with Intel Xeon CPU E5-2690 v3 2.6GHz, 48 logical cores and 128GB of RAM. The timing results are in Table 3.7. All numbers are in seconds and the predict time measures the time required to make predictions on the entire test set. We can see in terms of speed, BR is the clear winner. However, its poor accuracy makes its advantage in speed less attractive. On datasets with a large number of possible subsets, CBM takes significantly less training time compared with PowSet, while still achieving better accuracy. By taking advantage of the sparsity, sparse CBM runs a few times faster than dense CBM. We also notice that on a separate large dataset RCV1-2K (with 20X more labels and 100X more instances than RCV1; see Appendix A for details), training a sparse CBM with 50 components is even 3 times faster than training a BR model, because with many components, each binary classifier training is done on a subset of data within the component, and thus can be much faster. This is very promising and shows that CBM not only is more powerful than BR, but also could be more efficient than BR on large datasets. Training CBM on extremely large dataset is beyond the scope of the this thesis and we leave it for future work.

Table 3.7: Comparison between CBM and other methods in terms of training time and prediction time. All numbers are in seconds.

	dataset	SCENE		RCV1		TMC2007		MEDIAMILL		NUSWIDE	
Method	Learner	Train	Predict	Train	Predict	Train	Predict	Train	Predict	Train	Predict
BR	LR	2	<1	19	<1	26	<1	136	<1	128	1
PowSet	LR	35	<1	3147	<1	38037	1	85794	1	521760	34
CC	LR	3	<1	509	<1	332	<1	1949	1	2520	2
PCC	LR	3	<1	509	3	332	1	1949	4	2520	27
ECC-label	LR	22	<1	4915	27	3404	15	19642	38	25791	246
ECC-subset	LR	22	<1	4915	26	3404	18	19642	39	25791	287
CDN	LR	4	45	18417	213433	54253	596228	3126	6572	17941	41789
pairCRF	linear	11	<1	2136	<1	215	<1	2990	<1	48404	7
(dense) CBM	LR	70	<1	4412	4	1495	1	17608	13	35363	48
(sparse) CBM	LR	24	<1	182	<1	393	<1	8862	5	15561	14

3.11 Conceptual Comparison with Related Methods

Modeling label dependencies has been long regarded as a central theme in multi-label classification. Estimating the high-dimensional conditional joint $p(\mathbf{y}|\mathbf{x})$ is very challenging and various approaches have been proposed to tackle this problem based on different approximations.

Classifier Chains decomposes the joint probability $p(\mathbf{y}|\mathbf{x})$ into a product of conditionals $p(y_1|\mathbf{x})p(y_2|\mathbf{x}, y_1)\cdots p(y_L|\mathbf{x}, y_1, .., y_{L-1})$, based on the chain rule. This reduces a multi-label learning problem to L binary learning problems, each of which learns a new label given all previous labels. During prediction, finding the exact joint mode is intractable. Classifier Chains (CC) [97] classify labels greedily in a sequence: label y_ℓ is decided by maximizing $p(y_\ell|\mathbf{x}, y_1, .., y_{\ell-1})$, and becomes a feature to be used in the prediction for label $y_{\ell+1}$. This greedy prediction procedure has three issues: 1) the predicted subset can be far away from the joint mode [33]; 2) errors in early label predictions propagate to subsequent label predictions; 3) the overall prediction depends on the chain order. To address the first two issues, Probabilistic Classifier Chains (PCC) replace the greedy search strategy with some more accurate search strategies, such as exhaustive search [24], ϵ-approximate search [34], Beam Search [64, 65], or A* search [77]. To address the third issue, Ensemble of Classifier Chains (ECC) [97] averages several predictions made by different chains, wherein the averaging can take place at either the individual label level (ECC-label) or the label subset level (ECC-subset). Using dynamic programming to find the optimal chain order [73] has been proposed recently.

A similar reduction method named **Conditional Dependency Networks** (CDN) estimates $p(\mathbf{y}|\mathbf{x})$ based on full conditionals and Gibbs sampling [50]. During learning, one binary classifier is trained for each full conditional $p(y_\ell|\mathbf{x}, y_1, .., y_{\ell-1}, y_{\ell+1}, ..., y_L)$. During prediction, Gibbs sampling is used to find the mode of the joint. The method's major limitation is that it cannot handle perfectly correlated or anti-correlated labels: consider binary classification as multi-label problem with only 2 exhaustive and exclusive labels. A perfect model for $p(y_1 = 1|\mathbf{x}, y_2)$ is $1 - y_2$; in other words, the feature information is completely ignored. The same applies to $p(y_2|\mathbf{x}, y_1)$. So the prediction will inevitably fail. In general, Gibbs sampling may fail in the presence of perfect correlations or anti-correlations since the resulting stochastic process is not ergodic; when relations are imperfect but very strong, Gibbs sampling may need a very long time to converge.

The method in [43] factorizes labels into independent factors based on some statistical conditional independence tests and a Markov boundary learning algorithm. Its limitation, as pointed out by the authors, is that the statistical tests are ineffective when the ratio between samples and labels is not big enough. Their method outperforms the Power-Set baseline on synthetic datasets, but not on any real dataset. [129] propose to use a Bayesian Network to encode the conditional dependencies of the labels as well as the feature set, with the features as the common parent of all labels.

Conditional Random Fields (CRF) [102] offer a general framework for structured prediction problems based on undirected graphical models. In multi-label classifications labels can form densely connected graphs, so restrictions are imposed in order to make training and inference tractable. For problems involving only hard label relations (exclusive or hierarchical), a special CRF model is proposed [35]; this works only when label dependencies are *strict* and *a priori* known.

pair-CRF limits the scope of potential functions to label pairs, and does not model higher order label interactions [44]. The exact inference and prediction in pair-wise CRF requires checking all possible label subsets, which is intractable for datasets with many labels. As shown in their experiment results, the most effective way of doing approximate inference and prediction is to consider only label subsets that occur in the training set. But this eliminates the possibility of predicting unseen subsets. Our CBM model does not have this limitation (see Section 3.6).

There are also algorithms which exploit label structures but do not estimate $p(\mathbf{y}|\mathbf{x})$ explicitly. Examples include Multi-label k-Nearest Neighbors [130], Compressed Sensing based method [53], Principle Label Space Transformation [104], Conditional Principal Label Space Transformation [23] and Compressed

Labeling [132].

Various **mixture models** exist for multi-label document classification or supervised topical modeling [75, 109, 74, 122, 96, 94, 60]. Our CBM model differs from these models in two aspects: 1) These models are generative in nature, i.e., they model $p(\mathbf{x})$, where $\mathbf{x}$ is typically a bag-of-words representation of a document. By contrast, our CBM is purely discriminative, since it targets $p(\mathbf{y}|\mathbf{x})$ directly. The principled advantage of discriminative approach over generative approach, as stated in [103], is that the former does not make overly simplistic independence assumptions among features, and is thus better suited to including rich, overlapping features. 2) Most of these models are designed specifically for text data, while our method can be applied to any multi-label classification problem. A Conditional Multinomial Mixture model has been proposed for Superset Label Learning [72].

The architecture of CBM also mimics that of Mixture of Experts (ME) [56, 58], in which a gate model divides the input space into disjoint regions probabilistically and an expert model generates the output in each region. ME has been mainly used in regression and multi-class problems [126]. CBM can be viewed as a multi-label extension of ME with a particular factorization of labels inside each expert.

3.12 Discussion and Future Work

3.12.1 CBM with Shared Parameters

While the CBM model as in Equation (3.4) is flexible, it may not be very efficient in parameterization because each component is forced to use its own set of parameters by setting $b(y_l|z = k, \mathbf{x}; \boldsymbol{\beta}_l) = b(y_l|\mathbf{x}; \boldsymbol{\beta}_l^k)$. Recall that during training, each binary classifier in component k is trained using only data within the (soft) component k. If the number of components is large and the total number of instances is small, each component may only get a small number of the training instances and thus we might not be able to estimate the binary classifier very well. Take object detection as an example. Each component in CBM may correspond to a different image scene/topic (see Figure 3.2 for an illustration). Without parameter sharing, we will need to build a human detector inside the kitchen scene, and another human detector inside the street scene, and another human detector inside the sports scene, and so on. It could be true that a human playing football may look somewhat differently than a human working in the kitchen, they still share many similarities. Training separate human detectors in each scene may lead to a data scarcity problem. To alleviate this problem, we can instead write the component indicator as a binary vector $\mathbf{z}$ with

k-th bit being 1, concatenate $\mathbf{z}$ with $\mathbf{x}$ and train a binary classifier on augmented features $(\mathbf{x}, \mathbf{z})$.

$$p(\mathbf{y}|\mathbf{x}) = \sum_{k=1}^{K} \pi(z^k = 1|\mathbf{x}; \boldsymbol{\alpha}) \prod_{\ell=1}^{L} b(y_\ell|\mathbf{x}, \mathbf{z}; \boldsymbol{\beta}_\ell), \tag{3.10}$$

The model is not forced to use different parameters for the same label classifier in different components. This new and more general formulation allows the model to share some parameters. The amount of parameter sharing depends on the concrete instantiation of the binary classifiers. Nonlinear classifiers such as neural networks and boosted trees can arbitrarily combine $\mathbf{x}$ and $\mathbf{z}$. They can potentially first find patterns that apply to all components and then make small adjustments inside each component.

We now consider how to train such a model. We denote with $\mathbf{e}_k$ the binary vector of length K with the k-th bit being one. The E step and the update for the π remain the same. The update for binary classifiers becomes: for $l = 1, 2, ..., L$

$$\boldsymbol{\beta}_l \leftarrow \arg\min_{\boldsymbol{\beta}_\ell} \sum_{k=1}^{K} \sum_{n}^{N} \gamma_n^k \log b(y_{nl}|\mathbf{x}_n, \mathbf{z} = \mathbf{e}_k; \boldsymbol{\beta}_l) \tag{3.11}$$

Conceptually, we have KN instance feature vectors of the form $(\mathbf{x}_n, \mathbf{z} = \mathbf{e}_k)$, and each instance has an associated weight γ_n^k and we need to solve a weighted maximum likelihood problem. Implementing this idea directly would require creating K different copies of $\mathbf{x}$, each concatenated with a different $\mathbf{z}$. This naive implementation is clearly quite inefficient in both training time and memory consumption. We will need to develop some tricks to avoid this. Interestingly, if one uses a linear model for $b(y_\ell|\mathbf{x}, \mathbf{z}; \boldsymbol{\beta}_\ell)$, the training can be greatly simplified – no explicit concatenation of $\mathbf{x}$ and $\mathbf{z}$ is required in the implementation.

For logistic regression learners:

$$b(Y_{nl} = 1|\mathbf{x}_n, \mathbf{z}) = \frac{1}{1 + \exp\{-(\sum_d w_l^d x_n^d + \sum_k c_l^k z^k + c_l^0)\}}$$

The gradient is easy to compute: Let $J_l = \sum_{k=1}^{K} \sum_n^N \gamma_n^k \log b(y_{nl}|\mathbf{x}_n, \mathbf{z}; \mathbf{w}_l, \mathbf{c}_l)$, then

$$\frac{\partial J_l}{\partial w_l^d} = \sum_n y_{nl} x_n^d - \sum_n x_n^d E_{nl}$$
$$\frac{\partial J_l}{\partial c_l^k} = \sum_n y_{nl} \gamma_n^k - \sum_n \gamma_n^k b(Y_{nl} = 1|\mathbf{x}_n, \mathbf{z} = \mathbf{e}_k)$$

where

$$E_{nl} = \sum_k \gamma_n^k p(Y_{nl} = 1 | \mathbf{x}_n, \mathbf{z} = \mathbf{e}_k)$$

The training complexity is (roughly) $O(NL(D + K))$, where N is the number of instances, L number of labels, D number of features, K number of components. As a comparison, the original (dense) CBM without parameter sharing has the complexity $O(NLDK)$. Typically $K << D$. It is clearly that in CBM with shared parameters, increasing the number of components no more leads to a linear increase in the training time. This is the main advantage of parameter sharing.

The disadvantage is that using linear classifiers leads to the most strict and least flexible parameter sharing: the overall scoring function $\sum_d w_l^d x^d + \sum_k c_l^k z^k + c_l^0$ shows that we essentially have a fixed label detector $\sum_d w_l^d x^d$ that provides a matching score between the instance $\mathbf{x}$ and the label l, one global bias c_l^0 that encodes the global prior probability of the label, and a component specific bias c_l^k that encodes the prior probability of that label appearing in the component. This model can represent shared (component-independent) patterns through the label detector function, but cannot make complicated adjustment inside each component. All it can do is to add some fixed score or subtract some fixed score.

Despite its limited modeling power, this does suggest to us some interesting ways to think about label dependencies. In object detection, many labels, such as `dinner table` or `car`, look pretty consistently across scenes/components. In this case, we can basically use one fixed detector $\sum_d w_l^d x^d$ for a label across all scenes. Dining tables appear frequently in the dinning room scene, and thus the label `dining table` will get a high positive bias c_l^k (a high prior) in the component that represents the dining room. The label `car`, however, will get a negative bias c_l^k (a low prior) in the dining room component. Now assuming that we have an image which seems to contain a blurry table or some parts of a table. CBM will first (probabilistically) map the image to a scene using the π classifier based on all the features. Assuming there is some clear evidence such as folks, foods, chairs, and fridge (which themselves could also be labels to be predicted – and they are indeed easily predictable in this case) and the image is correctly mapped to the dining room scene, then the binary classifier will add some extra positive score c_l^k to the original (possibly low) score $\sum_d w_l^d x^d$ for `dining table`, and thus boosts its chance of being predicted. And it can also deduct a lot of score for `car`, decreasing the chance of `car` being predicted.

This example shows that even if CBM uses a restricted parameterization and is only allowed to adjust the label prior probabilities in each component, it can still capture label dependencies. The dependency estimation is not done *directly* – by

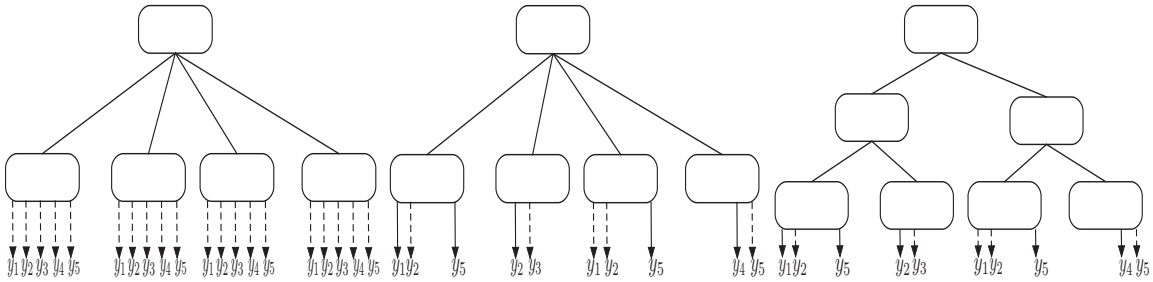


Figure 3.9: CBM with dense structure vs. CBM with sparse structure vs. CBM with hierarchical structure. From left to right: CBM, sparse CBM, and hierarchical sparse CBM. Mixture components are the leafs; dashed arrows represent binary label classifiers; solid arrow are labels always predicted in the component. Missing arrows indicate labels never predicted in the component.

estimating which set of labels tend to co-occur and which ones not; but rather, done *indirectly* – by clustering labels into scenes and memorizing which labels tend to occur in which scenes. We observe that on real datasets, most of the labels only occur in a few scenes. And this property can be explored to design a sparse CBM structure, as we have described in Section 3.5. Such clustering of labels must also be realizable by features – the π classifier should be able to send the instance to the right cluster(s) based on its features. Such clustering happens in the EM training procedure, where both label information and feature information are considered.

To fully utilize the power of parameter sharing, and allow the model to make more sophisticated adjustments in each component, in the future, we would like to develop an efficient training procedure for CBM with shared non-linear binary learners.

3.12.2 CBM with Hierachical Structures

Another training complexity reduction idea worth considering is to use a *hierarchical* mixture structure as opposed to the current *flat* mixture structure. If we view sparse CBM as a depth 1 probabilistic tree, then to make a prediction for each instance, we have to compute its probability of reaching each leaf node (i.e., the mixture coefficient), which requires evaluating a large multinomial classifier. If we use a hierarchical CBM, which can be viewed as a deep probabilistic tree, then each prediction requires evaluating some small multinomial (or binary if the tree is binary) classifiers. The hope is that many branches with low contribution can be pruned early in the evaluation process and only a small number of classifiers are

actually evaluated. See Figure 3.9 for a demonstration. This looks similar to the tree based methods for extreme multi-label classification. But our approach uses probabilistic tree splits as opposed to the hard tree splits, and provides probabilistic inference necessary for many downstream tasks (such as F1 optimization). If labels are almost independent, traditional tree based classifiers become very inefficient in representation, while our model with dedicated binary label classifiers will not have this issue. Note that using a hierarchical structure in CBM does not mean the labels have to form a hierarchy themselves.

Chapter 4

Calibrate and Rerank Multi-label Predictions

In the previous chapter, we have described CBM, a probabilistic model designed to capture label dependencies by directly estimating the joint distribution. In this chapter, we introduce a completely different approach to multi-label classification called BR-rerank. It consists of a BR model which only estimates marginal label probabilities, and a calibrator model which takes as input these marginals as well as other properties of the label set to produce a calibrated confidence for the label set. The calibrator captures label dependencies missed by the BR model and is able to rerank BR's predictions. Compared to standard BR, BR-rerank gives both better predictions and better confidence estimations. Similar to CBM, BR-rerank is a reduction methods: it transforms a multi-label classification problem to a series of binary classification problems and a regression/calibration problem.

4.1 Binary Relevance and its Drawbacks

The simplest approach to multi-label classification is to apply one binary classifier (e.g., binary logistic regression or support vector machine) to predict each label separately. This approach is called binary relevance (BR) [107] and is widely used due to its simplicity and speed. BR's training time grows linearly with the number of labels, which is considerably lower than many methods that seek to model label dependencies, and this makes BR run reasonably fast on commonly used datasets. (Admittedly, BR may still fail to scale to datasets with extremely large number of labels, in which case specially designed multi-label classifiers with sub-linear time

complexity should be employed instead. But in this thesis, we shall not consider such *extreme* multi-label classification problem.)

BR has two well-known drawbacks. First, BR neglects label dependencies and this often leads to prediction errors: some BR predictions are incomplete, such as tagging `cat` but not `animal` for an image, and some are conflicting, such as predicting both the code `Pain in left knee` and the code `Pain in unspecified knee` for a medical note. Second, the confidence score or probability (we shall use "confidence score" and "probability" interchangeably) BR associates to its overall set prediction $\mathbf{y}$ is often misleading, or uncalibrated. BR computes the overall set prediction confidence score as the product of the individual label confidence scores, i.e., $p(\mathbf{y}|\mathbf{x}) = \prod_{l=1}^{L} p(y_l|\mathbf{x})$. This overall confidence score often does not reflect reality: among all the set predictions on which BR claims to have roughly 80% confidence, maybe only 60% of them are actually correct (a predicted set is considered "correct" if it matches the ground truth set exactly). Having such uncalibrated prediction confidence makes it hard to integrate BR directly into a decision making pipeline where not only the predictions but also the confidence scores are used in downstream tasks.

In this chapter, we seek to address these two issues associated with BR. We first improve the BR set prediction confidence scores through a feature-based post calibration procedure to make confidence scores indicative of the true set accuracy. The features considered in calibration capture label dependencies that have otherwise been missing in standard BR. Next we improve BR's set prediction accuracy by reranking BR's prediction candidates using the new calibrated confidence scores. There exist multi-label methods that avoid the label independence assumption from the beginning and perform joint probability estimations [97, 24, 64, 44, 35, 68]; such methods often require more complex training and inference procedures. In this thesis we show that BR base model together with our proposed post calibration/reranking makes accurate set predictions on par with (or better than) these state-of-the-art multi-label methods —yet *calibrated*, *simpler*, and *faster*.

4.2 Calibrate BR Multi-label Predictions

We first address BR's confidence mis-calibration issue. There are two types of confidence scores in BR: the confidence of an individual label prediction $p(y_l|\mathbf{x})$, and the confidence of the entire predicted set $p(\mathbf{y}|\mathbf{x})$. In this work we take for granted that the individual label scores have already been calibrated, which can be easily done with established univariate calibration procedures such as isotonic

regression [98] or Platt scaling [127, 92]. We are concerned here with the set confidence calibration; note that calibrating all individual label confidence scores does not automatically calibrate set prediction confidence scores.

4.3 Evaluation Metrics for Confidence Calibration

To describe our calibration method, we need the following formal definitions:
• $c(\mathbf{y}) \in [0, 1]$ is the confidence score associated with the set prediction $\mathbf{y}$;

• $v(\mathbf{y}) \in \{0, 1\}$ is the 0/1 correctness of set prediction $\mathbf{y}$;

• $e(\mathrm{c}) = p[v(\mathbf{y}) = 1 | c(\mathbf{y}) = \mathrm{c}]$ is the average set accuracy among all predictions whose confidence is c. In practice, this is estimated by bucketing predictions based on confidence scores and computing the average accuracy for each bucket.

We use the following standard metrics for calibration [63]:

• Alignment error, defined as $\mathbb{E}_{\mathbf{y}}[e(c(\mathbf{y})) - c(\mathbf{y})]^2$, measures, on average over all predictions, the discrepancy between the claimed confidence and the actual accuracy. The smaller the better.

• Sharpness, defined as $\mathrm{Var}_{\mathbf{y}}[e(c(\mathbf{y}))]$, measures how widely spread the confidence scores are. The bigger the better.

• The mean squared error (MSE, also called Brier Score), defined as $\mathbb{E}_{\mathbf{y}}[(v(\mathbf{y}) - c(\mathbf{y}))^2]$, measures the difference between the confidence and the actual 0/1 correctness. It can be decomposed into alignment error, sharpness and an irreducible constant "uncertainty" due to only classification error (not calibration error) [63]:

$$\underbrace{\mathbb{E}_{\mathbf{y}}[(v(\mathbf{y}) - c(\mathbf{y}))^2]}_{MSE} = \underbrace{\mathrm{Var}_{\mathbf{y}}[v(\mathbf{y})]}_{uncertainty} - \underbrace{\mathrm{Var}_{\mathbf{y}}[e(c(\mathbf{y}))]}_{sharpness} + \underbrace{\mathbb{E}_{\mathbf{y}}[(e(c(\mathbf{y})) - c(\mathbf{y}))^2]}_{alignment\ error} \tag{4.1}$$

Alignment error and sharpness capture two orthogonal aspects of confidence calibration. A small alignment error implies that the confidence score is well aligned with the actual accuracy. However, small alignment error, alone, is not meaningful: the calibration can trivially achieve zero alignment error while being completely uninformative by assigning to all predictions the same confidence score, which is the average accuracy among all predictions on the dataset. A useful calibrator should also separate good predictions from bad ones as much as possible by assigning very different confidence scores to them. In other words, a good calibrator should simultaneously minimize alignment error and maximize sharpness. This can be achieved by minimizing MSE, thus MSE makes a natural objective for calibrator

training. Minimizing MSE leads to a standard regression task: one can simply train a regressor c that maps each prediction $\mathbf{y}$ to its binary correctness $v(\mathbf{y})$. Note that training a calibrator by optimizing MSE does not require estimation of $e(c(\mathbf{y}))$, but evaluating its sharpness and alignment error does. Estimating $e(c(\mathbf{y}))$ by bucketing predictions has some subtle issues, as we shall explain later when we present evaluation results in Section 4.6.

4.4 Features for Calibration

Besides the training objective, we also need to decide the parametric form of the calibrator and the features to be used. In order to explain the choices we make, we shall use the calibration on WISE dataset [7] as a running example.

Let us start by visualizing BR's predictions together with its original set prediction confidence calculated as $p(\mathbf{y}|\mathbf{x}) = \prod_{l=1}^{L} p(y_l|\mathbf{x})$, in which the individual label probabilities $p(y_l|\mathbf{x})$ have been well-calibrated by standard isotonic regression procedures. We group about every 100 predictions with similar confidence scores into a bucket, and plot those buckets as dots treating the average confidence in the group as the x-coordinate and treating the average prediction accuracy in the group as the y-coordinate[1]. Figure 4.1a shows that the set confidence scores computed this way are not calibrated even when the individual label confidence scores have been well-calibrated. Well calibrated set predictions should approximately lie on the diagonal line. In the figure, predictions above the diagonal are under-confident and those below the diagonal are over-confident.

The simplest way to improve the alignment is to fit another isotonic regression to these dots (see Figure 4.1c), and use the regression outputs as the new calibrated set prediction confidence scores (Figure 4.1d). This additional calibration makes the dots align with the diagonal much better. Quantitatively, the alignment error has been reduced to a small number. However, as mentioned earlier, having a small alignment error alone is not enough, as a trivial calibrator that outputs a constant would achieve zero alignment error (Figure 4.1b). One would also need to maximize the sharpness of the scores, by assigning very different scores to good predictions and bad predictions. Figure 4.2a and 4.2c show that there are features that can help the calibrator better separate good predictions from bad ones.

[1]This particular way of bucketing is only for visualization purpose; when we evaluate calibration quantitatively we follow the standard practice of using 10 equal-width buckets.

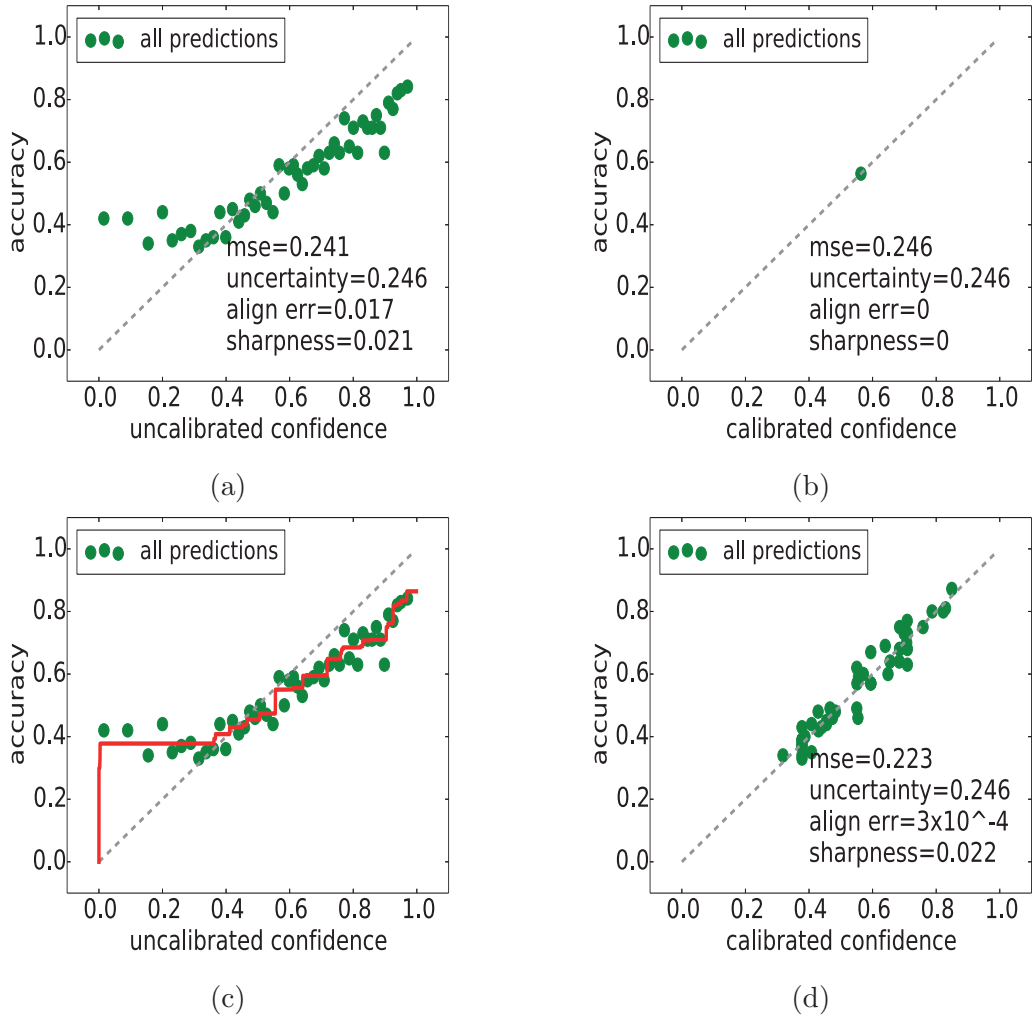


Figure 4.1: Trivial calibration and isotonic regression calibration on the WISE dataset. In all sub-figures, each dot represents a group of 100 set predictions with similar confidence scores. The average confidence score in the group is used as x-coordinate, and the average prediction accuracy is used as y-coordinate. (a) BR predictions with the original BR confidence scores. (b) Trivial calibration that gives all predictions the same confidence score which is the overall set accuracy on the dataset. (c) Isotonic regression (the solid line) trained on all predictions. (d) Predictions with isotonic regression calibrated confidence. To simplify the presentation, all calibrators are trained and evaluated on the same data.

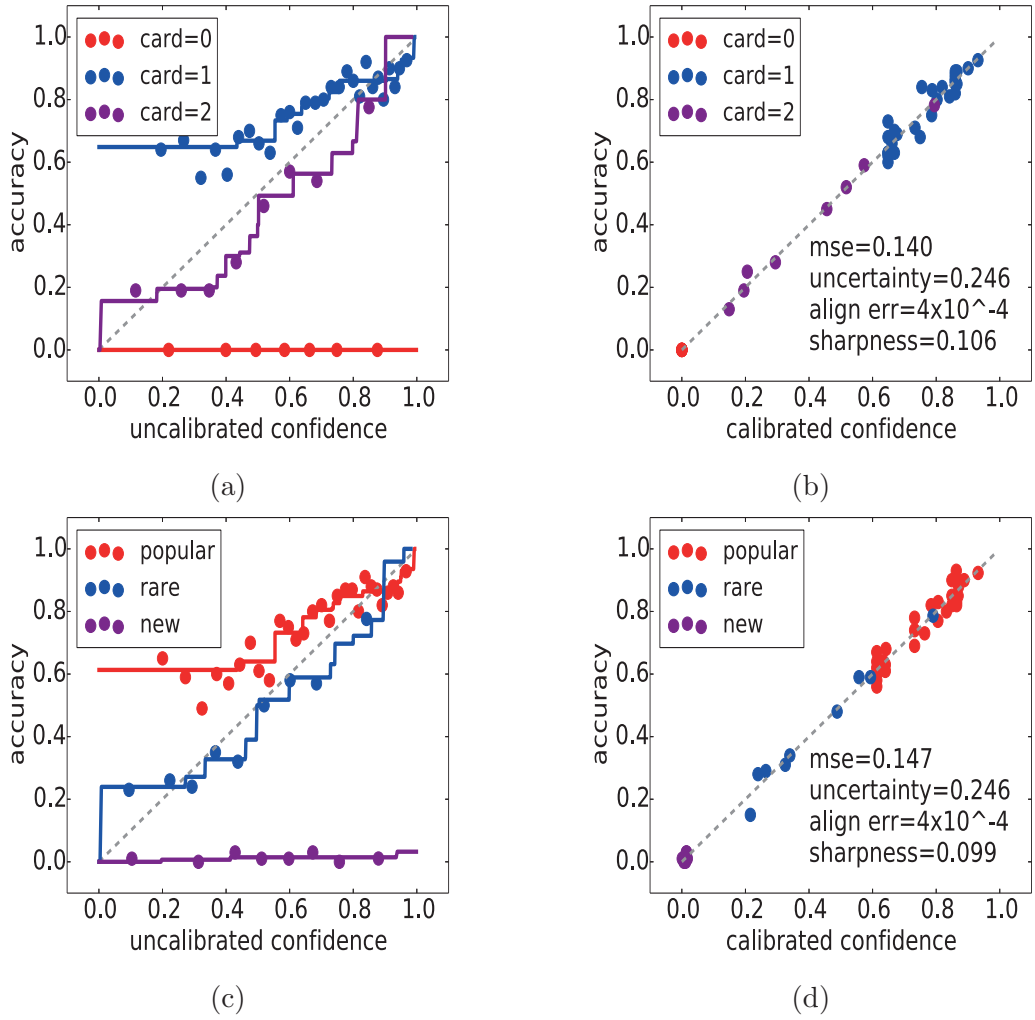


Figure 4.2: Cardinality and prior based isotonic regression calibrations on the WISE dataset. In all sub-figures, each dot represents a group of 100 set predictions with similar confidence scores. The average confidence score in the group is used as x-coordinate, and the average prediction accuracy is used as y-coordinate. (a) Break down all predictions by the set cardinality and train a separate isotonic regression (the solid line) for each cardinality. (b) Predictions with confidence calibrated by cardinality-based isotonic regressions. (c) Group all predictions into 3 categories by the popularity (prior probability) of predicted label combination in the training data ground truth (*popular*=the predicted label combination appears at least 100 times; *rare*=the predicted label combination appears less than 100 times; *new*=the predicted label combination does not appear at all in training data), and train a separate isotonic regression (the solid line) for each category. (d) Predictions with confidence calibrated by prior-based isotonic regressions. To simplify the presentation, all calibrators are trained and evaluated on the same data.

Figure 4.2a breaks down all predictions by the *cardinality* of the predicted set (i.e., the number of labels predicted). If we look at all predictions with uncalibrated confidence around 0.7, their average accuracy is around 0.58 (as shown in Figure 4.1c). However, Figure 4.2a shows that those singleton predictions have accuracy around 0.8; those predictions containing 2 labels only have accuracy about 0.54; and those empty set predictions have 0 accuracy (on this particular dataset, the ground truth set is always non-empty). Clearly, predictions with different cardinalities require different calibration mappings from the uncalibrated confidence to the actual accuracy. Fitting a separate isotonic regression for each cardinality results in Figure 4.2b, which is a clear improvement over the calibration without cardinality (Figure 4.1d); thus the cardinality feature greatly increases sharpness and reduces MSE. Visually, more points have moved towards left and right ends.

Another useful feature is the *popularity* of predicted label set in the training data (i.e., prior probability). Between two predictions with the same uncalibrated BR confidence, the one that is more popular often has a higher chance of being correct, as shown in Figure 4.2c. One can discretize the prior probabilities into intervals and train separate isotonic regressions for different intervals. Figure 4.2d shows that this also performs better than having only one isotonic regression.

Both set cardinality and prior probability are features defined on the whole label set, rather than individual labels. Such features capture constraints and dependencies among labels, which were not originally considered by BR. Therefore these features supplement BR's own prediction score and allow the calibrator to make better overall judgments on the predicted set. There can be other features that help the calibrator better judge set predictions. In order to incorporate arbitrary number of features and avoid manual partitioning of the data and training separate calibrators (which quickly becomes infeasible as the number of features grows), a general multi-variate regressor should be employed. The multi-variate extension of isotonic regression exists [99], but it is not well suited to our problem because some features such as cardinality do not have a monotonic relationship with the calibrated confidence (see Figure 4.2a). [63] proposes KNN and regression trees as calibrators for general structured prediction problem.

4.5 Gradient Boosting as a Calibrator

In this work, we choose Gradient Boosted Trees (GB) [40] as the calibrator model. Similar to regression trees, GB as a multi-variate regressor automatically partitions the prediction's feature space into regions and outputs (approximately) the average

prediction accuracy in each region as the calibrated confidence score. GB often produces smoother output than trees and generalizes better. GB is also very powerful in modeling complex feature interactions automatically by building many trees on top of the features. To leverage its power we also use the binary representation of the set prediction $\mathbf{y}$ itself as features for GB. This way GB can discover additional rules concerning certain label interactions that are not described by the manually designed features (for example, "if two conflicting labels A and B are both predicted, the prediction is never correct, therefore lower the confidence score"). It is also possible to use instance features $\mathbf{x}$ during calibration, but we do not find it helpful because BR was already built on $\mathbf{x}$.

There are two commonly used GB variants [40]. The first variant, GB-MSE (Algorithm 2.5), uses the tree ensemble score as the output, and MSE as the training objective. The second variant, GB-KL (Algorithm 2.6), adds a sigmoid transformation to the ensemble score and uses KL-divergence as the training objective. GB-MSE has the advantage of directly minimizing MSE, which matches the evaluation metric used for calibration (see section 4.3). But it has the disadvantage that its output is not bounded between 0 and 1 and one has to clip its output in order to treat that as a confidence score.

GB-KL has the advantage of providing bounded output, but its training objective does not directly match the evaluation metric used for calibration; note, however, that minimizing KL-divergence also encourages the model output to match the average prediction accuracy, hence achieves the calibration effect. It may appear that one could get the best of both worlds by having sigmoid transformation and MSE training objective at the same time. Unfortunately, adding sigmoid makes MSE a non-convex function of the ensemble scores, thus hard to optimize. In this chapter, we choose GB-MSE as our GB calibrator and shall simply call it GB from now on. In Appendix E, we show that GB-KL has very similar performance.

Each BR set prediction is transformed to a feature vector (containing original BR confidence score, set cardinality, set prior probability, and set binary representation) and the binary correctness of the prediction is used as the regression target. Since the goal of GB calibrator is to objectively evaluate BR's prediction accuracy, it is critical that the calibration data to be disjoint from the BR classifier training data. Otherwise, when BR over-fits its training data, the calibrator will see over-optimistic results on the same data and learn to generate over-confident scores. Similarly, it is also necessary to further separate the label calibration data and the set calibration data, since the product of the calibrated label probabilities is used as input to the set calibrator training. The whole calibrator training procedure is summarized in Algorithm 4.1.

Algorithm 4.1 Gradient Boosting Calibrator Training

Input: calibration dataset $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ together with BR's predictions $\hat{\mathbf{y}}_n, n = 1, 2, ..., N$.

1: **for** $n = 1, 2, ..., N$ **do**
2: create feature vector $\mathbf{u}_n$ from $\hat{\mathbf{y}}_n$ and binary label $v_n = \mathbb{I}[\hat{\mathbf{y}}_n = \mathbf{y}_n]$
3: **end for**
4: run Algorithm 2.5 on dataset $\{(\mathbf{u}_n, v_n)\}_{n=1}^N$ to produce GB model F

Output: final GB model F

Imposing partial monotonicity. Imposing monotonicity is a standard practice in univariate calibration methods such as isotonic regression [98] and Platt scaling [127, 92] as it avoids over-fitting and leads to better interpretability. Imposing (partial) monotonicity for a multi-variate calibrator is more challenging. Certain features considered in calibration are expected to be monotonically related to the confidence. For example, the confidence should always increase with the popularity (prior probability) of the predicted set, if all other features of the prediction are unchanged. The same is true for BR score. The rest of the features, including the cardinality of the set and the binary representation of the set, do not have monotonic relations with confidence. Therefore the calibration function is partially monotonic. We have done additional experiments on imposing partial monotonicity for the GB calibrator but did not observe significant improvement (details and experiment results are in Appendix E).

4.6 Experiment Results on Calibration

We test the proposed GB calibrator for BR set predictions on 6 commonly used multi-label datasets (see Appendix A for details). Each dataset is randomly split into training, calibration, validation and test subsets. BR model with logistic regression base learners is trained on training data; isotonic regression label calibrators and GB set calibrators are trained on (different parts of) calibration data. All hyper parameters in BR and calibrators are tuned on validation data. Calibration results are reported on test data.

For comparison, we consider the following calibrators:

- uncalib: use the uncalibrated BR probability as it is;
- isotonic: calibrate the BR probability with isotonic regression;
- card isotonic: for each label set cardinality, train one isotonic regression;

- tree: use the features considered by GB, train a single regression tree.

To make a fair comparison, for all methods, individual label probabilities have already been calibrated by isotonic regressions. We focus on their abilities to calibrate set predictions. BR prediction is made by thresholding each label's probability (calibrated by isotonic regression) at 0.5. This corresponds to the set with the highest BR score.

The evaluation metrics we use are MSE, sharpness and alignment error, as described in Section 4.3. Following the standard practice, we use 10 equal-width buckets to estimate sharpness and alignment error. One issue with evaluation by bucketing is that using different number of buckets leads to different estimations of alignment error and sharpness (but not MSE and uncertainty, whose computations do not depends on bucketing). In fact, increasing the number of buckets will increase both the estimated alignment error and sharpness by the same amount, due to Eq 4.1. Using 10 buckets often produces negligible alignment error (relative to MSE), and the comparison effectively focuses on sharpness. This amounts to maximizing sharpness subject to a very small alignment error [46], which is often a reasonable goal in practice. All calibrators are able to achieve small alignment error (on the order of 10^{-3} and contributing to less than 10% of the MSE), so we do not report that. The results are summarized in Table 4.1. All calibrators improve upon the BR uncalibrated probabilities. Our GB calibrator achieves the overall best MSE and sharpness calibration performance, due to use of additional features extracted from set predictions.

Table 4.1: BR prediction calibration performance in terms of MSE (the smaller the better) and sharpness (the bigger the better). Bolded numbers are the best.

Dataset	uncertainty	uncalib		isotonic		card isotonic		tree		GB	
		MSE	sharp	MSE	sharp	MSE	sharp	MSE	sharp	MSE	sharp
BIBTEX	0.139	0.193	0.007	0.140	0.002	0.109	0.038	0.086	0.065	**0.068**	**0.072**
OHSUMED	0.232	0.226	0.015	0.221	0.013	**0.182**	**0.051**	0.211	0.039	0.189	0.047
RCV1	0.247	0.175	0.077	0.175	0.075	0.159	0.093	0.134	**0.129**	**0.123**	0.126
TMC	0.212	0.192	0.019	0.192	0.020	0.192	0.022	0.194	0.029	**0.180**	**0.032**
WISE	0.249	0.252	0.017	0.234	0.017	0.151	0.098	0.166	0.093	**0.147**	**0.102**
MSCOCO	0.227	0.158	0.075	0.151	0.075	0.150	0.076	0.163	0.070	**0.143**	**0.083**

4.7 Reranking Multi-label Predictions with Calibrated Confidence

(a) (b)

Figure 4.3: Two example images from the MSCOCO datasets on which BR-rerank corrects BR's predictions. For (a), BR-rerank adds the correct label `baseball glove` to BR's predictions. For (b), BR-rerank removes the incorrect label `toothbrush` from BR's predictions.

Now we aim to improve BR's prediction accuracy, by fixing some of the prediction mistakes BR made due to ignoring label dependencies. Our solution is based on the calibrator we just developed. Traditionally, the only role of a calibrator is to map an uncalibrated confidence score to a calibrated confidence score. In that sense the calibrator usually does not affect the classification, only the confidence. In fact, popular univariate calibrators such as isotonic regression and Platt scaling implement monotonic functions, thus preserve the ranking/argmax of predictions. For our multi-variate GB calibrator, however, this is not the case. Even if we constrain the calibrated confidence to be monotonically increasing with the BR prediction scores, there are still other features that may affect the ranking; in particular the argmax predictions before and after calibration might be different $\mathbf{y}$ sets. If indeed different, the prediction based on calibrated confidence takes into account label dependencies and other constraints (which BR does not), and is more likely to be the correct set (even when the calibrated confidence is not very high in absolute terms). Therefore we can also use GB as a multi-label wrapper on top of BR to rerank its predictions. We name this method as ***BR-rerank***. Figure 4.3 shows two example images from

Table 4.2: The two-stage BR-rerank predictions on the two example images in Figure 4.3. For each example image, the left column of the table shows the top-5 set prediction candidates generated by BR. The middle column shows the uncalibrated BR confidence score. The right column shows the calibrated BR-rerank confidence score. For Figure 4.3a: BR predicts the incorrect set {`person,baseball bat`} with confidence 0.58. BR-rerank predicts the correct set {`person, baseball bat, baseball glove`} with confidence 0.17. For Figure 4.3b: BR predicts the incorrect set {`person, remote, toothbrush`} with confidence 0.70. BR-rerank predicts the correct set {`person, remote`} with confidence 0.18.

Prediction on Figure 4.3a		
y candidates	BR score	BR-rerank score
person, baseball bat	0.58*	0.16
person, baseball bat, baseball glove	0.35	0.17*
person, handbag, baseball bat	0.02	0.04
person, sports ball, baseball bat	0.02	0.08
person, handbag, baseball bat, baseball glove	0.01	0.03

Prediction on Figure 4.3b		
y candidates	BR score	BR-rerank score
person, remote, toothbrush	0.70*	0.16
person, remote	0.24	0.18*
person, toothbrush	0.03	0.05
person	0.01	0.02
person, tennis racket, remote, toothbrush	0.01	0.01

the MSCOCO datasets on which BR-rerank corrects BR's predictions. Table 4.2 compares the BR scores with the BR-rerank scores on these two examples.

BR-rerank uses a two stage prediction pipeline. For each test instance $\mathbf{x}$, we first list the top K label set candidates $\mathbf{y}$ by highest BR uncalibrated scores. This can be done efficiently using a dynamic programming procedure (Algorithm 4.2) which takes advantage of the label independence assumption made in BR. Although the label independence assumption does not hold in practice, we find empirically that when K is reasonably large (e.g., $K = 50$), the correct $\mathbf{y}$ is often included in the top-K list. The chance that the correct answer is included in the top-K list is commonly called "oracle accuracy", and it is an upper bound of the final prediction accuracy. Empirically, we observe the oracle accuracy to be much higher than the final prediction accuracy, indicating that the candidate generation stage is not a

bottleneck of final performance.

Prediction stage two: send the top set candidates with their scores and additional features to the GB calibrator, and select the one with the highest calibrated confidence as the final prediction. The calibrator has to be trained on more than top-1 BR candidates (on a separate calibration dataset) to evaluate correctly prediction candidates, so we train the GB calibrator on top-K candidates. Figure 4.4 illustrates the BR-rerank prediction on the test example in Figure 4.3b, showing the BR marginal probabilities and the features extracted from the candidate sets.

Algorithm 4.2 Generating the K-best prediction candidates from BR

Input: instance $\mathbf{x}$ and a BR classifier
Compute individual label probabilities based on BR: $p_l = p(y_l = 1|\mathbf{x}), l = 1, 2, ..., L$
Initialize an empty priority queue Q^k, and empty list C and an empty label set $\mathbf{y}_{best}$
for $\ell = 1, 2, ..., L$ **do**
  if $p_l > 0.5$ **then**
    add l to $\mathbf{y}_{best}$
  end if
end for
$Q^k.enqueue(\mathbf{y}_{best})$
while $|C| < K$ **do**
  $\mathbf{y} = Q^k.dequeue()$
  add $\mathbf{y}$ to C
  for $\ell = 1, 2, ..., L$ **do**
    Generate $\mathbf{y}'$ by flipping the ℓ-th bit of $\mathbf{y}$
    if $\mathbf{y}'$ has not been added to Q before **then**
      $Q^k.enqueue(\mathbf{y}')$
    end if
  end for
end while
Output: C

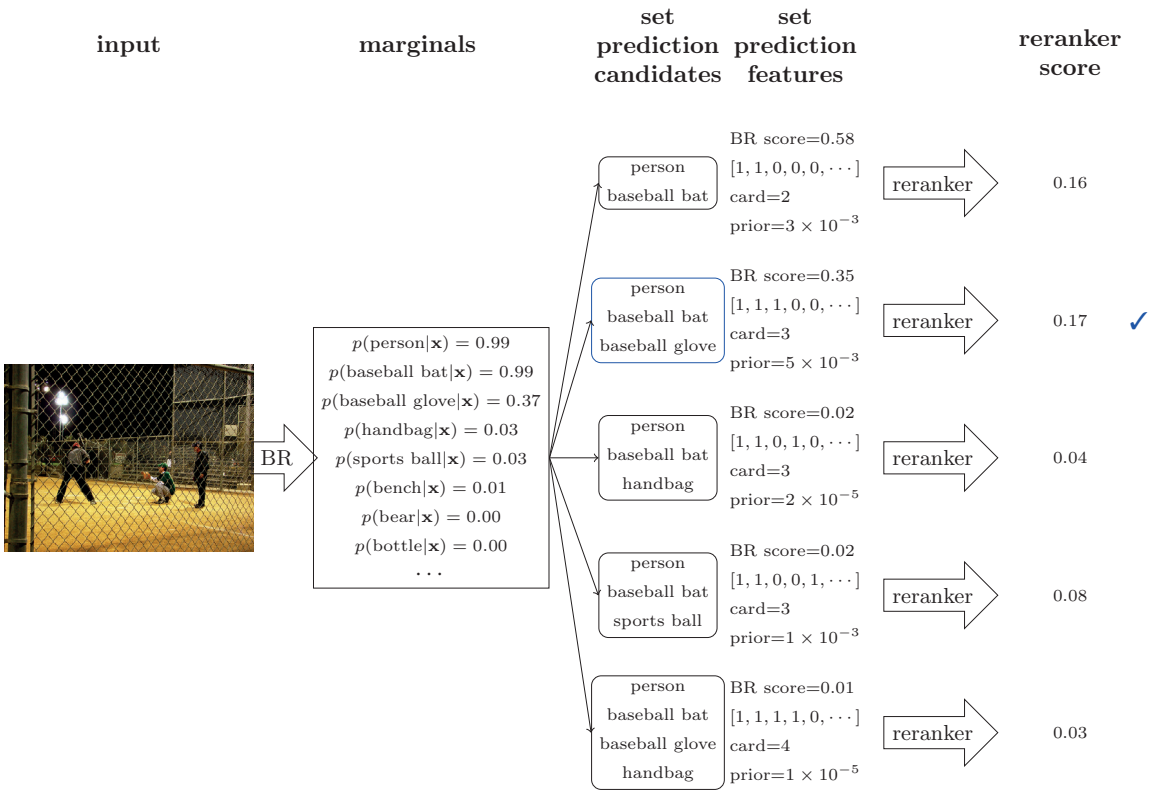


Figure 4.4: BR-rerank prediction details for the input test image. The "marginal" column shows the individual label probabilities estimated by BR. Note that the label `baseball glove` has a probability below the 0.5 threshold, and therefore will not be included in BR's predictions. The "set prediction candidates" column shows the top-5 set prediction candidates with the highest BR scores generated by dynamic programming based on BR marginals. The "set prediction features" column shows, for each set candidate, its BR score, its binary encoding, its cardinality and its prior probability. The "reranker score" column shows the calibrated BR-rerank confidence score for each set prediction candidate. For this image, BR predicts the incorrect set {`person,baseball bat`} with confidence 0.58. BR-rerank predicts the correct set {`person, baseball bat, baseball glove`} with confidence 0.17.

4.8 Conceptual Comparison with Related Multi-label Classifiers

Although the proposed BR-rerank classifier has a very simple design, it has some advantages over many existing multi-label classifiers. Here we make some conceptual comparisons between BR-rerank and related multi-label classifiers.

BR-rerank can be seen as a stacking method in which a stage-1 model provides initial estimations and a stage-2 model uses these estimations as input and makes the final decision. There are other stacking methods proposed in the literature, and the two most well-known ones are called 2BR [47, 106] and DBR [80]. The stage-1 models in 2BR and DBR are also BR models just as in BR-rerank. The stage-2 models in 2BR and DBR work differently. In 2BR, the stage-2 model predicts each label ℓ with a separate binary classifier which takes as input the original instance feature vector $\mathbf{x}$ as well as all label probabilities predicted by the stage-1 model. In DBR, the stage-2 model predicts each label ℓ with a separate binary classifier which takes as input the original instance feature vector $\mathbf{x}$ as well as the binary absence/presence information of all other labels. The absence/presence of label ℓ itself is not part of the input to avoid learning a trivial mapping. During training, the absence/presence information is obtained from the ground truth; during prediction, it is obtained from the stage-1 model's prediction. Clearly for DBR there is some inconsistency on how stage-2 inputs are obtained. BR-rerank and 2BR do not suffer from such inconsistency. All three stacking methods BR-rerank, 2BR, and DBR try to incorporate label dependencies into final classification. However, both 2BR and DBR have a critical flaw: when their stage-2 models make the final decision for a particular label, they do not really take into account the *final decisions* made for other labels by the stage-2 model; they instead only consider the *initial estimations* on other labels made by the stage-1 model, which can be quite different. As a result, the final set predictions made by 2BR and DBR may not respect the label dependencies/constraints these models have learned. By contrast, the stage-2 model in BR-rerank directly evaluates the final set prediction (based on its binary representation and other extracted features) to make sure that the final set prediction satisfies the desired label dependencies/constraints. For example, in the RCV1 dataset, each instance has at least one label. But DBR predicted the empty set on 6% of the test instances. By contrast, BR-rerank never predicted empty set on this dataset.

Many multi-label methods avoid the label independence assumption made in BR and model the joint distribution $p(\mathbf{y}|\mathbf{x})$ in more principled ways. Examples include Conditional Random Fields (CRF) [44], Conditional Bernoulli Mixtures (CBM) [68], and Probabilistic Classifier Chains (PCC) [97, 24, 64, 73]. Despite the joint estimation formulation, CRF, CBM, and PCC in practice often produce over-confident set prediction confidence scores, due to overfitting. Their prediction confidence must also be post-calibrated. Figure 4.5 compares BR, BR-rerank and CBM in terms of prediction confidence calibration on the MSCOCO dataset. As we can see from the plots, BR prediction scores are under-confident, BR-rerank prediction scores are well-calibrated, and CBM prediction scores are over-confident.

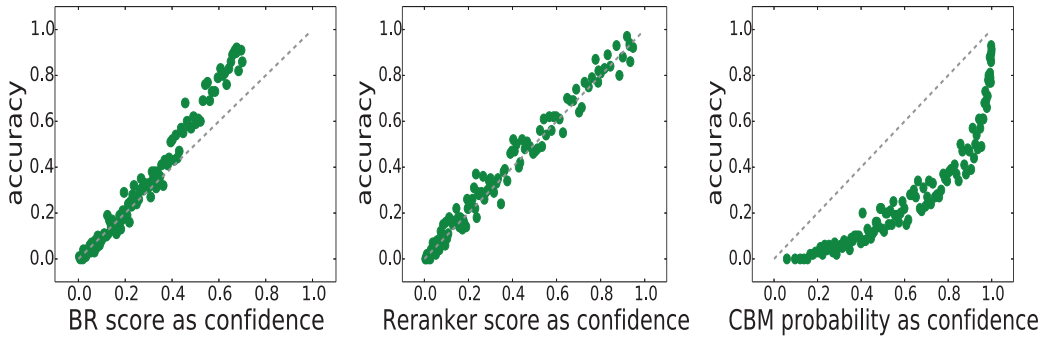


Figure 4.5: Comparison between BR, BR-rerank and CBM in terms of confidence calibration on MSCOCO test dataset. Each dot represents a group of 100 set predictions with similar confidence scores. The average confidence score in the group is used as x-coordinate, and the average prediction accuracy is used as y-coordinate.

The pair-wise CRF model [44] captures pairwise label interactions by estimating 4 parameters for each label pair. However, because the model needs to assign dedicated parameters to different label combinations, modeling higher order label dependencies becomes infeasible. The BR-rerank approach we propose relies on boosted trees to automatically build high order interactions as tree learns their splits on the binary representation of the label set – there is no need to allocate parameters in advance. There is another CRF model designed specifically to capture exclusive or hierarchical label relations [35]; this works only when the label dependency graph is *strict* and *a priori* known.

CBM is a latent variable model and represents the joint as a mixture of binary relevance models. However, it is hard to directly control the kinds of dependencies CBM learns, or to enforce constraints in the prediction. For example, CBM sometimes assigns high probability to the empty set even on dataset where empty prediction is not allowed. There is no natural way to enforce the cardinality constraint in CBM. In our CBM implementation, we have to add a dedicated rule in the prediction procedure to avoid the empty prediction.

PCC decomposes the joint $p(\mathbf{y}|\mathbf{x})$ into a product of conditional probabilities $p(y_1|\mathbf{x})p(y_2|\mathbf{x}, y_1) \cdots p(y_L|\mathbf{x}, y_1, .., y_{L-1})$, and reduces a multi-label problem to L binary problems, each of which learns a new label given all previous labels. However, different label chaining orders can lead to different results, and to find the best order is often a challenge. In BR-rerank, all labels are treated as features in the GB calibrator training and they are completely symmetric.

The Structured Prediction Energy Network (SPEN) [15] uses deep neural

networks to efficiently encode arbitrary relations between labels, which to large degree avoids parameterization issue associated with pair-wise CRF, but it cannot generate a confidence score for its MAP prediction as computing the normalization constant is intractable. The Predict-and-Constraint method (PC) [19] specifically handles cardinality constraint (but not other label constraints or relations) during learning and prediction. Deep value network (DVN) [52] trains a neural network to evaluate prediction candidates and then uses back-propagation to find the prediction that leads to the maximum score. The idea is similar to our BR-rerank idea. The difference is: DVN could only use the binary encoding of the label set, but not any higher level features extracted from the label set, such as cardinality and prior set probability. That is because its gradient based inference makes it very difficult to directly incorporate such features. There are methods that seek to rank labels [20, 41]. Our method differs from them in that we rank label sets as opposed to individual labels, and we take into account label dependencies in the label set.

4.9 Experiment Results on Classification

We test the proposed BR-rerank classifier on 6 popular multi-label datasets (see Appendix A for details). All datasets used in experiments contain at least a few thousands instances. We do not take datasets with only a few hundred instances as their testing performance tends to be quite unstable. We also do not consider datasets with extremely large number of labels as our method is not designed for extreme classification (our method aims to maximize set accuracy but on extreme data it is very unlikely to predict the entire label set correctly due to large label set cardinality and annotation noise). We compare BR-rerank with many other well-known multi-label methods: Binary Relevance (BR) [107], 2BR [47, 106], DBR [80], pair-wise Conditional Random Field (CRF) [44], Conditional Bernoulli Mixture (CBM) [68], Probabilistic Classifier Chain (PCC) [97], Structured Prediction Energy Network (SPEN) [15], PD-Sparse (PDS) [124], Predict-and-Constrain (PC) [19], Deep value network (DVN) [52], Multi-label K-nearest neighbors (KNN) [130], and Random k-label-sets (RAKEL) [108].

To make a fair comparison, we use logistic regressions as the underlying learners for BR as well as the stage-1 models in BR-rerank, 2BR and DBR. We use gradient boosting as the underlying learners in PCC as well as the stage-2 models in BR-rerank, 2BR and DBR. Each dataset is randomly split into training, validation and test subsets. All classifiers are trained on the training set, with hyper parameters tuned on validation set. Appendix B and Appendix D contain implementation and

Table 4.3: Prediction performance in terms of set accuracy (top) and instance F1 (bottom). Numbers are shown as percentages. Bold numbers are the best ones on each dataset. "-" means the method could not finish within 24 hours on a server with 56 cores and 256G RAM or 4 NVIDIA Tesla V100 GPUs. The ranking indicates for each method, on average over datasets, what position its performance is (lower is better). Our BR-rerank has the best average ranking on both measures. Note also that BR is not the worst as one might naively assume. Hyper parameters for all methods have been tuned on validation set.

Dataset	BR	*BR-rerank*	2BR	DBR	CBM	CRF	SPEN	PDS	DVN	PC	PCC	RAKEL	KNN
BIBT	16.6	21.5	16.1	20.2	22.9	**23.3**	14.8	16.1	16.2	20.3	21.4	18.3	8.4
OHSU	36.6	**42.0**	37.5	37.6	40.5	40.4	29.1	34.8	18.6	29.5	38.0	39.3	25.4
RCV1	44.5	53.2	42.3	45.8	**55.3**	53.8	27.5	40.8	13.7	39.7	48.7	46.0	46.2
TMC	30.4	**33.3**	32.1	31.7	30.8	28.2	26.7	23.4	20.3	23.0	31.3	27.6	18.9
WISE	52.9	60.5	51.8	55.8	**61.0**	46.4	-	52.4	28.3	-	55.9	3.5	2.4
MSCO	34.7	**35.9**	33.7	32.0	31.1	35.1	34.1	25.0	29.9	31.1	32.1	32.6	29.1
ranking	6.3	1.8	6.7	5.7	3.3	3.8	10.0	9.8	11.2	10.0	4.5	6.8	11.0

Dataset	BR	*BR-rerank*	2BR	DBR	CBM	CRF	SPEN	PDS	DVN	PC	PCC	RAKEL	KNN
BIBT	35.9	42.2	36.7	40.1	45.3	46.2	38.6	40.4	47.3	**47.5**	40.9	38.3	23.0
OHSU	62.9	**67.5**	62.9	61.5	67.2	65.6	58.8	66.4	60.0	60.5	61.7	62.3	48.6
RCV1	77.0	78.8	77.5	72.8	**80.3**	75.0	66.5	76.7	36.3	71.7	75.6	76.1	72.3
TMC	65.8	66.8	**67.9**	66.1	65.2	64.4	66.2	64.0	65.5	61.7	64.9	63.6	52.2
WISE	68.3	75.4	69.1	69.9	**76.0**	60.7	-	73.6	62.3	-	69.7	6.2	5.6
MSCO	73.0	73.2	72.6	69.6	70.0	**73.9**	73.2	64.8	72.7	72.7	69.6	71.7	68.2
ranking	6.3	2.5	5.5	7.3	4.0	5.7	8.3	6.8	7.5	8.8	7.5	8.7	12.0

hyper parameters tuning details. For BR-rerank and 2BR, since the stage-2 model uses stage-1 model's out-of-sample prediction as input, the stage-1 model and stage-2 model are trained on different parts of the training data. For DBR, since the stage-2 model training only takes the ground truth labels as input, both stage-1 model and stage-2 model are trained on the whole training set. For evaluation, we report set accuracy and instance F1, as defined in Section 2.1.

Test performance is reported in Table 4.3. As expected, by reranking BR-independent-prediction candidates, BR-rerank outperforms BR significantly. We also observe that generally BR-rerank only needs to rerank the top-10 candidates from BR in order to achieve the best performance. On each dataset, we rank all algorithms by performance, and report each algorithm's average ranking across all datasets. BR-rerank has the best average ranking with both metrics, followed by CBM and CRF. We emphasize that with slightly better performance, BR-rerank is noticeably simpler to use than CBM and CRF. CBM and CRF require implementing dedicated training and prediction procedures, while BR-rerank can be ran by simply combining existing machine learning libraries such as LIBLINEAR [37] for BR and Xgboost [21] for GB. BR-rerank is also much faster than CBM and CRF. Its running

time is determined mostly by its stage one, the BR classifier training. See Table 4.4 for a comparison.

Table 4.4: Comparison between BR-rerank and other methods in terms of training time. Time is measured in seconds. All algorithms run multi-threaded on a server with 56 cores.

Dataset	BIBTEX	OHSUMED	RCV1	TMC	WISE	MSCOCO
BR	4	3	7	8	80	1380
BR-rerank	9	6	10	11	88	1393
CBM	64	210	70	224	1320	8520
CRF	353	268	1223	771	16363	14760

Figure 4.6 shows BR-rerank classification set accuracy as a function of number of candidates K. When only the top-1 candidate is considered, BR-rerank is the same as BR. Significant improvement can be achieve by simply considering and reranking the top-10 candidates. Further increasing K does not give consistent improvement and sometimes causes the performance to drop slightly.

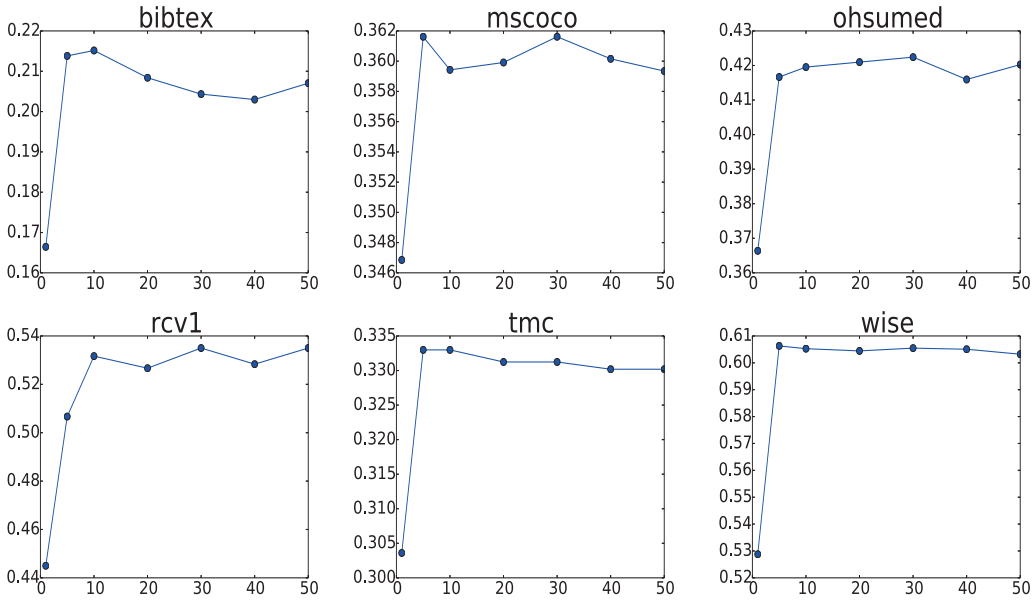


Figure 4.6: BR-rerank classification set accuracy as a function of the number of candidates K obtained from BR. Each subfigure is for a different dataset. X-axis is the number of candidates K, and Y-axis is the set accuracy on test data.

4.10 Calibration in the Presence of Noise

Standard calibration procedures (for binary/multi-class/multi-label classification) assume the availability of a calibration dateset with *clean* ground truth annotations. This is a natural requirement because the confidence (i.e., true accuracy of the predictions in each confidence bucket) is computed by comparing the predictions with the ground truth annotations. However, in many applications, such a clean calibration dataset may be hard to obtain: all we have is a *noisy* calibration set with random annotation errors.

All standard calibration procedures perform poorly on noisy data, since the confidence calibrated on noisy data only reflects the probability of the prediction matching the *noisy* annotation, as opposed to probability of getting the correct prediction, and the latter is what we really care about. Consider the extreme case where the classifier is 100% accurate but the annotation has 20% random noise. Then a calibrator trained on noisy data would output 0.8 as the confidence value for all predictions. In general, calibrators trained with standard procedures on noisy data tend to be under-confident. How to obtain well calibrated confidence using noisy data has great practical value but has not been studied before. Here we present an idea for calibration on noisy labels.

4.10.1 Seeing Through Noise with Unbiased Estimation

We tackle this problem by modifying the existing calibration methods to take into account the annotation noise rates. To do so, we assume that we have some estimations of the noise rates in the label annotation. Note that the availability of the noise rates is a much weaker assumption than the availability of the clean data. These noise rates can be obtained either based on prior experience, or from a small subset of data that went through the QA process. Let $c \in [0, 1]$ be the confidence that we want to output for a prediction, and let $v \in \{0, 1\}$ be the correctness of the prediction w.r.t. the clean ground truth, and let $\tilde{v} \in \{0, 1\}$ be the correctness of the prediction w.r.t. the noisy annotation. When we train the calibrator, we only observe $\tilde{v}$ directly, but not v. Let us introduce two parameters to describe the noise: $p(\tilde{v} = 0|v = 1) = \alpha$ and $p(\tilde{v} = 1|v = 0) = \beta$. α is the probability that an instance is mislabeled by the human annotator when the classifier's prediction is correct. When the classifier makes a correct prediction, the prediction matches the noisy annotation with probability $1 - \alpha$. The parameter β is the probability that the human annotator mislabels an instance in such a way that the incorrect annotation happens to match the classifier's wrong prediction. Both parameters are necessary

for theoretical analysis purpose, but in practice, we expect β to be close to 0 most of the time as it is rare for the classifier and the annotator to make exactly the same mistake at the same time. Also, we assume $\alpha + \beta < 1$ otherwise learning becomes impossible.

Ideally, one would like to match c with v by minimizing some loss function $\ell(c, v)$, such as square loss, or KL divergence. But since v is not directly observable, one has to work with a surrogate loss $\tilde{\ell}(c, \tilde{v})$. We want to define the surrogate loss $\tilde{\ell}$ in such a way that the expected loss $\tilde{\ell}$ w.r.t. the noisy annotation is the same as the loss ℓ w.r.t. the clean annotation, i.e., $\mathbb{E}_{\tilde{v}}\ \tilde{\ell}(c, \tilde{v}) = \ell(c, v)$. This gives an unbiased estimation of the true loss. The following lemma shows how to construct such a surrogate loss:

Lemma 1 ([12, 11, 84]) *Let $\ell(c, v)$ be any loss function. Suppose $p(\tilde{v} = 0|v = 1) = \alpha$ and $p(\tilde{v} = 1|v = 0) = \beta$. Then if we define*

$$\tilde{\ell}(c, \tilde{v}) := \begin{cases} \frac{(1-\alpha)\ell(c,0) - \beta\ell(c,1)}{1-\alpha-\beta} & \text{if } \tilde{v} = 0 \\ \frac{(1-\beta)\ell(c,1) - \alpha\ell(c,0)}{1-\alpha-\beta} & \text{if } \tilde{v} = 1 \end{cases} \tag{4.2}$$

we have, for any c, v, $\mathbb{E}_{\tilde{v}}\ \tilde{\ell}(c, \tilde{v}) = \ell(c, v)$.

4.10.2 Surrogate for Square Loss

When ℓ is the square loss $\ell(c, v) = (c - v)^2$, the surrogate loss $\tilde{\ell}(c, \tilde{v})$ has a simple form:

$$\tilde{\ell}(c, \tilde{v}) = (c - \frac{\tilde{v} - \beta}{1 - \alpha - \beta})^2 + M \tag{4.3}$$

where $M = \frac{\tilde{v}-\beta}{1-\alpha-\beta} - (\frac{\tilde{v}-\beta}{1-\alpha-\beta})^2$ is a constant that does not depend on c and can be safely ignored during optimization.

This surrogate has a natural interpretation: to train calibrators on noisy labels, one can still use the square loss. But instead of directly fitting the noisy target $\tilde{v}$, one should fit the modified target $\frac{\tilde{v}-\beta}{1-\alpha-\beta}$, which takes into account the noise rates. And on expectation, fitting this modified version of the noisy target is the same as fitting the noise-free target. Based on this observation, we can modify existing calibration methods that use square loss (e.g., isotonic regression and Gradient Boosting regressor) to make it noise-robust: we use the existing training procedures and simply change the training target from $\tilde{v}$ to $\frac{\tilde{v}-\beta}{1-\alpha-\beta}$.

However, there is a caveat: the modified training target $\frac{\tilde{v}-\beta}{1-\alpha-\beta}$ is either smaller than 0 (for $\tilde{v}=0$) or greater than 1 (for $\tilde{v}=1$), so they cannot be interpreted as confidence values themselves. One can view them as samples generated from a random variable whose mean is the confidence value that we want to estimate (which lies in the range $[0,1]$). The higher the noise rates α and β are, the higher the variance is, and the harder it is to estimate the mean accurately. Consider the high noise rate scenario with $\alpha=0.6$ and $\beta=0.2$. Then the training targets (samples) are either -1 or 4. Without sufficient training data, it is possible for the estimated mean (confidence) to lie outside of the desired range $[0,1]$ and lose its probabilistic interpretation. This is especially a problem for high capacity calibrators like GB which divide the feature space into regions and estimate the prediction confidence using samples in each region. Therefore we need to find a way to force the estimated confidence to lie in $[0,1]$. The simplest way is to clip the value. Another post-processing method is to "soft-clip" by passing all results through the function $1/[1+e^{-(x-0.5)}]$. While this function has the correct slope near 1/2 and properly preserves input probabilities near 1/2 to have associated output probabilities that are about the same, it does too much "soft clipping" near 1 or 0. The third idea is to first add a sigmoid transformation on top of the calibrator function and then train the calibrator with sigmoid by minimizing squared error. However, as noted in Section 4.5, adding the sigmoid this way makes the loss function non-convex w.r.t. the calibration score and therefore is hard to optimize. In short, although the surrogate loss for square loss has a simple form, we still lack a principled way to bound the calibrator output in a y has not been well solved.

4.10.3 Surrogate for KL Divergence

When ℓ is the KL divergence $\ell(c,v)=-\log(1-c)\mathbb{I}[v=0]-\log(c)\mathbb{I}[v=1]$, the surrogate loss is in the form

$$\tilde{\ell}(c,\tilde{v})=-m[(1-\tilde{v}-\alpha)\log(1-c)+(\tilde{v}-\beta)\log(c)] \tag{4.4}$$

where $m=\frac{1}{1-\alpha-\beta}$.

Unlike the surrogate loss for square loss, the surrogate loss for KL-divergence does not have a simple interpretation. In particular, it cannot be re-formulated as a KL-divergence between the confidence and some modified target. However, this surrogate loss has the advantage that it can be used together with the sigmoid transformation to bound the confidence score while preserving the convexity.

Consider a linear calibrator (e.g., Platt scaling) or non-linear calibrator (e.g., GB) which internally computes some model score s and then applies sigmoid

transformation to produce a confidence output $c = \frac{1}{1+e^{-s}}$. The surrogate loss (4.4) can be rewritten with s as

$$\tilde{\ell}(s, \tilde{v}) = m[(1 - \alpha - \beta) \log(1 + e^{-s}) + (1 - \tilde{v} - \alpha)s] \tag{4.5}$$

Its first and second order derivatives are

$$\frac{\partial \tilde{\ell}(s, \tilde{v})}{\partial s} = m[c - \tilde{v} + (1 - c)\beta - c\alpha] \tag{4.6}$$

$$\frac{\partial^2 \tilde{\ell}(s, \tilde{v})}{\partial s^2} = m[(1 - \alpha - \beta)\frac{e^{-s}}{(1 + e^{-s})^2}] \tag{4.7}$$

Clearly the second order derivative $\frac{\partial^2 \tilde{\ell}(s,\tilde{v})}{\partial s^2} > 0$ and therefore $\tilde{\ell}(s, \tilde{v})$ is a convex function of s.

4.10.4 Noise Tolerant GB Calibrator

In Section 4.5, we have described two GB calibrator variants, one using ensemble score as confidence and trained with square loss, and one applying sigmoid to the ensemble score as confidence and trained with KL-divergence. When we have clean calibration data, they perform similarly. When the calibration data is noisy, we shall add the sigmoid transformation to the GB ensemble score to get a bounded confidence and train the model with the surrogate loss for KL-divergence as we just derived.

Given a calibration dataset $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ with noisy annotations, and a BR classifier's predictions $\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_N$ on the calibration data, we train the noise tolerant GB calibrator as follows. We transform each prediction $\hat{\mathbf{y}}_n$ into a vector $\mathbf{u}_n$ by extracting various features from it, such as its raw BR probability, set cardinality, set prior probability, and we create binary labels $\tilde{v}_n = \mathbb{I}[\hat{\mathbf{y}}_n = \mathbf{y}_n]$. Then we train a GB on the dataset $\{(\mathbf{u}_n, \tilde{v}_n)\}_{n=1}^N$ with loss (4.5). The whole procedure is summarized in Algorithm 4.3. Once GB training is done, it can be used to output calibrated confidence: for a prediction $\mathbf{y}$, we first transform it to a vector $\mathbf{u}$, and then compute confidence score as $c = \frac{1}{1+e^{-F(\mathbf{u})}}$, where $F = h_1 + h_2 + \cdots + h_T$ is the GB ensemble.

4.10.5 Non-uniform Noise

The method described so far assumes that there are some uniform noise rates that apply to all annotations. In practice, it is often the case that different annotations

Algorithm 4.3 Robust Gradient Boosting Calibrator Training for Uniform Noise

Input: calibration dataset $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ together with BR's predictions $\hat{\mathbf{y}}_n, n = 1, 2, ..., N$.

1: **for** $n = 1, 2, ..., N$ **do**
2: create feature vector $\mathbf{u}_n$ from $\hat{\mathbf{y}}_n$ and binary label $\tilde{v}_n = \mathbb{I}[\hat{\mathbf{y}}_n = \mathbf{y}_n]$
3: **end for**
4: **for** iteration $t = 1, 2, ..., T$ **do**
5: **for** $n = 1, 2, ..., N$ **do**
6: compute the gradient of the loss (4.5) w.r.t. the current ensemble score $F(\mathbf{u}_n)$: $g_n = \frac{\partial \hat{\ell}(F(\mathbf{u}_n), \tilde{v}_n)}{\partial F(\mathbf{u}_n)} = m[c_n - \tilde{v}_n + (1 - c_n)\beta - c_n\alpha]$, where $c_n = \frac{1}{1+e^{-F(\mathbf{u}_n)}}$ and $m = \frac{1}{1-\alpha-\beta}$
7: **end for**
8: fit a regression tree h_t to the regression dataset $\{(\mathbf{u}_n, -g_n)\}_{n=1}^N$
9: shrink the regression tree leaf output values by ρ: $h_t \leftarrow \rho h_t$
10: add the new regression tree h_t to the ensemble: $F = h_1 + h_2 + \cdots + h_t$
11: **end for**

Output: final ensemble $F = h_1 + h_2 + \cdots + h_T$

have different levels of noise. For example, in medical billing, billing codes are annotated by a combination of expert coders, regular coders and novice coders. Clearly coders with different levels of skills have different annotation noise rates. It is also often known who made which annotation. This information allows us to assign different levels of noise rates to different groups of annotations.

To keep the mathematical derivation as general as possible, we assume each annotation $\mathbf{y}_n$ has its own noise rates α_n and β_n. How to set α_n and β_n properly in practice is problem-dependent. For example, in the medical billing example mentioned above, we may estimate the noise rates for each group of coders and assign the same rates to annotations done by coders from the same group. Our calibrator training procedure can be modified to handle non-uniform noise rates, as shown in Algorithm 4.4. Notice that here we treat $m_n = \frac{1}{1-\alpha_n-\beta_n}$ as a weight defined for each instance, ignoring it when calculating gradients but considering it when fitting regression trees.

4.10.6 Experiment Results

We test the proposed robust GB calibrator on MSCOCO dataset. We assume the given annotations are clean and inject artificial noise to the annotations. Our first

Algorithm 4.4 Robust Gradient Boosting Calibrator Training for Non-uniform Noise

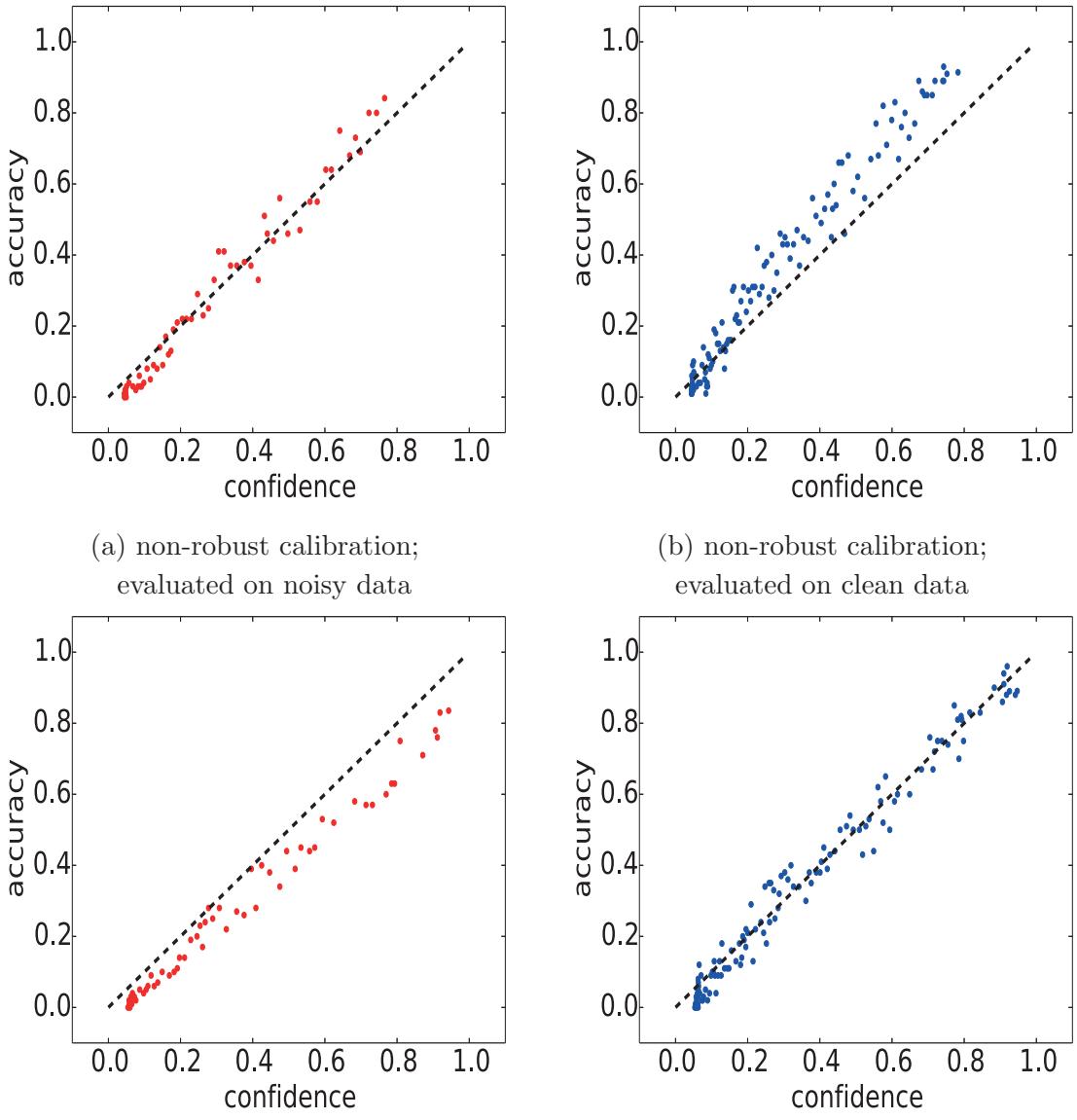
Input: calibration dataset $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ together with BR's predictions $\hat{\mathbf{y}}_n, n = 1, 2, ..., N$.

1: **for** $n = 1, 2, ..., N$ **do**
2: create feature vector $\mathbf{u}_n$ from $\hat{\mathbf{y}}_n$ and binary label $\tilde{v}_n = \mathbb{I}[\hat{\mathbf{y}}_n = \mathbf{y}_n]$
3: **end for**
4: **for** iteration $t = 1, 2, ..., T$ **do**
5: **for** $n = 1, 2, ..., N$ **do**
6: compute the gradient of the loss (4.5) w.r.t. the current ensemble score $F(\mathbf{u}_n)$: $g_n = \frac{\partial \hat{\ell}(F(\mathbf{u}_n), \tilde{v}_n)}{\partial F(\mathbf{u}_n)} = c_n - \tilde{v}_n + (1 - c_n)\beta_n - c_n\alpha_n$, where $c_n = \frac{1}{1+e^{-F(\mathbf{u}_n)}}$
7: **end for**
8: fit a regression tree h_t to the weighted regression dataset $\{(\mathbf{u}_n, -g_n, m_n)\}_{n=1}^N$, where $m_n = \frac{1}{1-\alpha_n-\beta_n}$
9: shrink the regression tree leaf output values by ρ: $h_t \leftarrow \rho h_t$
10: add the new regression tree h_t to the ensemble: $F = h_1 + h_2 + \cdots + h_t$
11: **end for**

Output: final ensemble $F = h_1 + h_2 + \cdots + h_T$

experiment tests uniform noise. We set $\beta = 0$ and only keep α. For each instance in the calibration data, with probability $\alpha = 0.2$ we corrupt its annotation by randomly adding or moving a label. To assess the calibrator's true performance w.r.t. *clean* annotations, we use the *clean* test set without noise for evaluation. For reference, we also show the calibrator's performance on the *noisy* calibration data. Note that our ultimate goal is to produce calibrated confidence w.r.t. the *clean* data. Figure 4.7 shows that the non-robust calibrator, while producing calibrated scores w.r.t. noisy labels, produces under-confident scores w.r.t. the clean labels. The robust calibrator, while producing over-confident scores w.r.t. the noisy labels, produces calibrated scores w.r.t. the clean labels. This shows that our robust calibration procedure is able to see through the noise and estimate the true confidence measured w.r.t. the unknown clean labels. Figure 4.8 shows similar trends when the noise rate is increased from 0.2 to 0.5.

Our second experiment tests non-uniform noise. We simulate three coders with different levels of skills. The expert coders has noise rate 0.1, the regular coder has noise rate 0.3 and the novice coder has noise rate 0.5. Each coder annotates a randomly chosen 1/3 of the data. For each instance in the calibration data, we corrupt its labels according to its coder's noise rate. We then train a robust calibrator using these noise rates. Figure 4.9 shows that our robust calibration procedure is also effective with non-uniform noise.



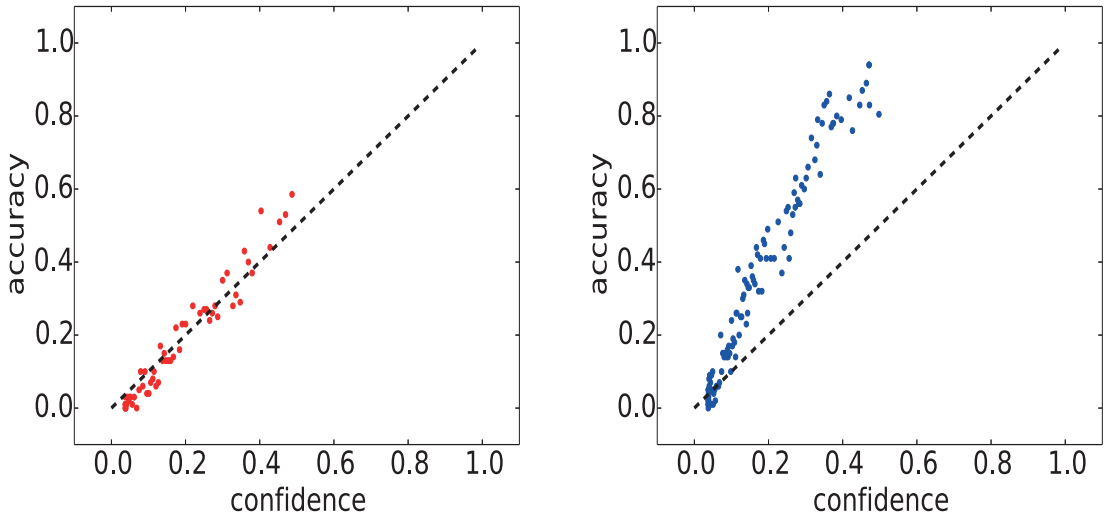
(a) non-robust calibration; evaluated on noisy data

(b) non-robust calibration; evaluated on clean data

(c) robust calibration; evaluated on noisy data

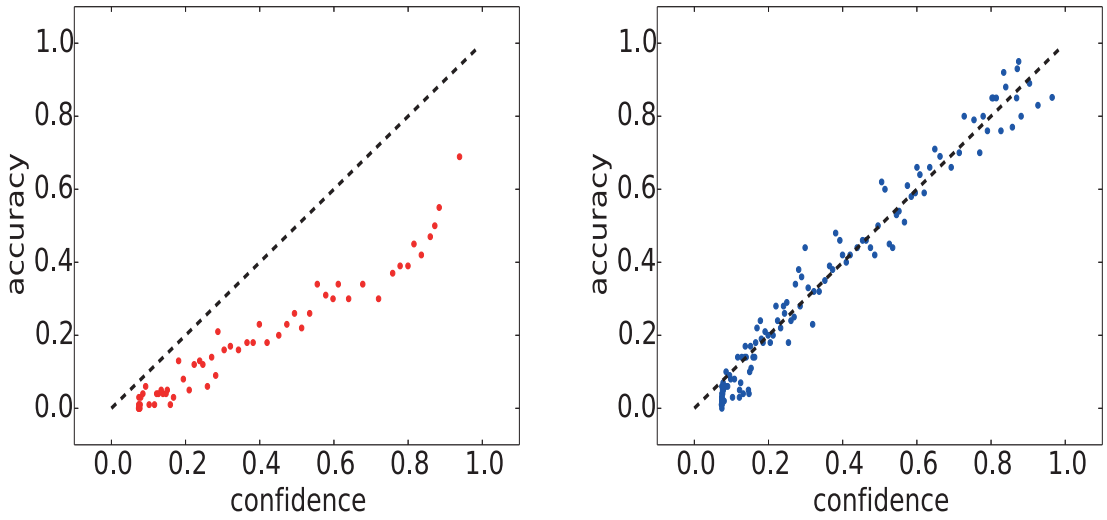
(d) robust calibration; evaluated on clean data

Figure 4.7: Non-robust calibrator vs. robust calibrator trained on noisy MSCOCO data with corrupted labels. Noise rate $\alpha = 0.2$. Each calibrator is evaluated against both clean data and noisy data.



(a) non-robust calibration; evaluated on noisy data

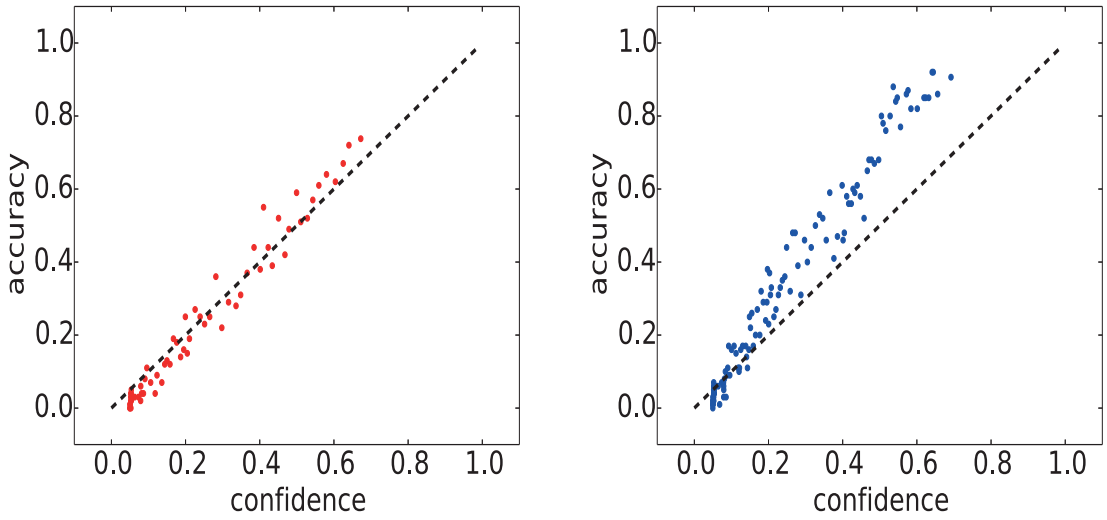
(b) non-robust calibration; evaluated on clean data



(c) robust calibration; evaluated on noisy data

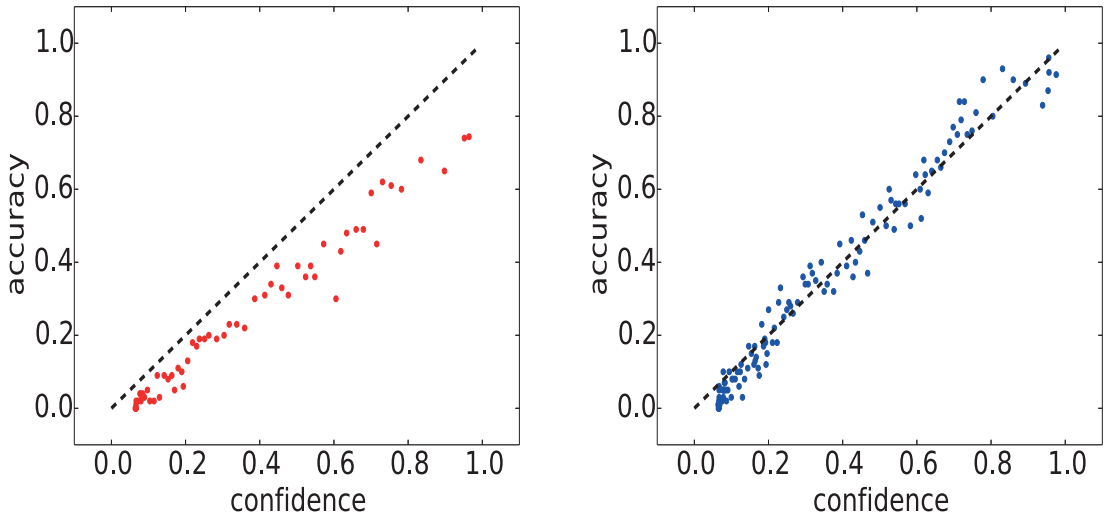
(d) robust calibration; evaluated on clean data

Figure 4.8: Non-robust calibrator vs. robust calibrator trained on noisy MSCOCO data with corrupted labels. Noise rate $\alpha = 0.5$. Each calibrator is evaluated against both clean data and noisy data.



(a) non-robust calibration; evaluated on noisy data

(b) non-robust calibration; evaluated on clean data



(c) robust calibration; evaluated on noisy data

(d) robust calibration; evaluated on clean data

Figure 4.9: Non-robust calibrator vs. robust calibrator trained on noisy MSCOCO data with corrupted labels. Noise is non-uniform with three different rates 0.1, 0.3 and 0.5. Each calibrator is evaluated against both clean data and noisy data.

4.11 Discussion and Future Work

The idea of first generating prediction candidates and then reranking them using richer features has been considered in several natural language processing tasks, including parsing [28] and machine translation [101]. Here we show that the reranking idea, with properly designed models and features, is well suited for multi-label classification as well. Generative Adversarial Nets (GANs) [48] also employ two models, one for generating samples and one for judging these samples. GANs are usually trained in an unsupervised fashion, and are mainly used for generating new samples. By contrast, our BR-rerank is trained in supervised fashion, and its main goal is to do classification. Also the two models in GANs are trained simultaneously, while the two models in BR-rerank are trained in separate stages.

Besides isotonic regression and Platt scaling, there are also some recent developments on binary, multi-class, and structured prediction calibration methods [62, 49, 63, 22]. Our work instead focuses on how to design the calibrator model and features for the BR multi-label classifier and how to take advantage of the calibrated confidence to get better multi-label predictions.

The calibration and reranking method described in this chapter has been focusing on the BR model and the set accuracy metric. There are several possible extensions to consider:

- **Applying the post-calibration and reranking procedure to other multi-label models.** Probabilistic multi-label models such as PCC, CRF and CBM intend to capture label dependencies and estimate the joint distribution in the first place. However, such models often produce uncalibrated confidence for the predicted set on the test data, during to overfitting. Additional calibration can be beneficial for these models as well. It would also be interesting to test whether reranking can further improve these model's classification performance. We speculate that PCC and CBM may not easily capture cardinality constraints, and the cardinality feature in the ranker may be helpful.
- **Calibrating and reranking w.r.t. other target metrics.** In the current form, the calibrator evaluates the chance of the predicted set matching the ground truth exactly, and the reranking step finds the set with the highest chance of being the perfect match. In some cases, set accuracy is not the proper metric to optimize. For example, when the label vocabulary is very large or when the annotation is very noisy, it is unlikely to have exact matches. Metrics assigning partial scores, such as F1 metric, are more appropriate.

How to calibrate and rerank w.r.t. F1 has great values both in theory and in practice. We can generalize the notion of confidence and define it w.r.t. F1 as: the confidence of the prediction w.r.t. F1 is the expected F1 of the prediction. We speculate that the features used to estimate expected F1 will be different than the features used to estimate expected set accuracy. For example, a predicted set with unseen label combinations is unlikely to be the correct set, but it may still have a high F1 score. So the prior probability feature is informative only in estimating set accuracy, but not F1.

- **Normalizing calibrated confidence scores.** The calibrated confidence scores among all possible sets generally do not sum to 1, even though the uncalibrated confidence scores do. This is because the calibrator maps each uncalibrated score to a calibrated one separately, and there is no post-normalization to ensure all calibrated scores sum to 1. Explicit normalization by summing over all possible prediction candidates is clearly intractable. This lack of normalization may not be an issue in practice if one is only interested in the confidence associated with the top-1 or top-K predictions. But from a theoretical perspective, this is undesirable as one can no longer interpret the calibrated confidence as a well defined probability distribution. To the best of our knowledge, how to compute normalized calibrated confidence scores in an efficient way has not been studied and it could be an interesting topic in itself.

- **Calibration with unknown noise rates.** The robust calibrator training algorithm developed in this chapter assumes that the noise rates are known to the training algorithm. Getting these noise rates in practice requires dedicated QA process in which the given noisy annotations are re-judged. This demands additional human effort. How to calibrate on noisy annotations when the noise rates are unknown has great practical value. We speculate that it is possible to estimate the noise rates from the noisy annotation alone without parallel clean annotations. One potential idea is to do a grid search of the noise rates using *robust classifiers*. There are robust classifier training procedures that take into account noise rates [84]. One can train a series of robust classifiers with different noise rates. Then the best noise rates can be chosen by evaluating and comparing all robust classifiers on a *noisy* validation set, because the the classification accuracy measured w.r.t. noisy labels is monotonically increasing with the classification accuracy measured w.r.t. clean labels.

Chapter 5

Conclusion

In this thesis, we considered the problem of multi-label classification, which assigns a set of labels to each instance. For example, an article can belong to multiple categories; an image can be associated with several tags; in medical billing, a patient report is annotated with multiple diagnosis codes. The most popular approach, named binary relevance (BR), trains one binary classifier to predict each label separately. BR ignores label dependencies and often makes conflicting predictions, such as tagging `cat` but not `animal` for an image. How to learn label dependencies from data and train classifiers to account for such dependencies is the central question in multi-label classification. This thesis described two new approaches to multi-label classification which leverage label dependencies and achieve better classification accuracy than BR and many other sophisticated methods.

The first approach, called conditional Bernoulli mixture (CBM), directly estimates the joint probability distribution among all labels with a mixture model. It consists of a multi-class classifier that assigns each instance to a mixture component probabilistically and one BR model per component that estimates marginal label probabilities. We showed that the joint distribution constructed this way is capable of capturing label dependencies. This special model structure also allows for efficient training, joint inference and marginal inference procedures designed to optimize different metrics. Specifically we developed a EM based training procedure that updates the multi-class classifier and all the binary classifiers in the model. By leveraging the sparsity of the model structure we were able to further speed up training. We developed a dynamic programming based joint label prediction method to optimize set accuracy. We sent the CBM joint estimation to the GFM algorithm to optimize instance F1 metric. We also showed that CBM can perform marginal inference efficiently to optimize Hamming loss.

The second approach, named BR-rerank, seeks to to improve both BR's confidence estimation and prediction through post calibration and reranking procedures. BR-rerank takes the BR predicted set of labels and its product score as features, extracts more features from the prediction itself to capture label constraints, and applies Gradient Boosted Trees (GB) as a calibrator to map these features into a calibrated confidence score. The GB calibrator not only produces well-calibrated scores (*aligned with accuracy* and *sharp*), but also models label interactions, correcting a critical flaw in BR. We further showed that reranking label sets by the new calibrated confidence makes accurate set predictions on par with state-of-the-art multi-label classifiers—yet calibrated, simpler, and faster. We further considered the problem of calibration in the presence of annotation noise, and developed a noise-robust calibration procedure by modifying the calibrator training loss function.

Both CBM and BR-rerank are reduction methods: they transform a complex multi-label classification problem to a series of standard binary classification, multi-class classification and regression problems, which are easier to solve.

We implemented the proposed methods CBM and BR-rerank together with many baseline methods and made the implementations publicly available at `https://github.com/cheng-li/pyramid`.

Appendix A

Datasets Used in Experiments

Table A.1: Datasets used in experiments

	type	source	labels	label sets	features	instances	cardinality
BIBTEX	bookmark	[8]	159	2856	1836	7395	2.4
IMDB	movie genre	crawled	28	2564	27228	34157	2.4
MEDIAMILL	video	[8]	101	6555	120	43907	4.4
MSCOCO	image	[5]	80	25500	4096	123287	2.9
NUSWIDE	image	[8]	81	18430	128	269648	1.9
OHSUMED	medical note	[6]	23	1147	12639	13929	1.7
RCV1	news	[8]	103	799	47236	6000	3.2
RCV1-2K	news	[4]	2456	13914	47236	779809	4.8
SCENE	image	[8]	6	15	294	2407	1.1
TMC2007	report	[8]	22	1341	49060	28596	2.2
WISE	article	[7]	203	4253	301561	64857	1.4
WIPO	patent	[9]	188	151605	74435	1710	4.0

Appendix B

Implementations Used in Experiments

Table B.1 shows the code we ran in our experiments.

Method	Implementation
BR, BR-rerank, 2BR, DBR, PCC, CRF, CBM	ours: `https://github.com/cheng-li/pyramid`
RAKEL, KNN	`https://github.com/scikit-multilearn/scikit-multilearn`
DVN	`https://github.com/gyglim/dvn`
PC	`https://github.com/Natalybr/predict_and_constrain`
PDS	`http://www.cs.utexas.edu/~xrhuang/PDSparse/`
SPEN	`https://github.com/davidBelanger/SPEN`

Table B.1: Implementations used in our experiments.

Appendix C

Hyper Parameter Tuning in CBM Experiments

For the experiments conducted in Section 3.6.3, we did cross-validation on the training set to tune the following hyper parameters:

- LR Gaussian prior variance V_{LR}, on grids $\{10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4}, 10^{5}, 10^{6}\}$;
- CRF Gaussian prior variance V_{CRF}, on grids $\{10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4}, 10^{5}, 10^{6}\}$;
- CBM+LR number of components K, on grids $\{5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60\}$;
- CBM+GB number of components K, on grids $\{5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60\}$.

Tuning results are shown in Table C.1.

Table C.1: Tuned CBM Hyper Parameters

dataset	V_{LR}	V_{CRF}	K (CBM+LR)	K (CBM+GB)
SCENE	1.0	1.0	20	25
RCV1	10^{6}	1.0	45	45
TMC2007	10^{-1}	10^{-1}	40	20
MEDIAMILL	10^{3}	1	50	5
NUSWIDE	1.0	1.0	50	10

Our gradient boosting implementation uses regression trees of 5 leaves and shrinkage rate 0.1 as default values. For PCC, the beam search width is set to be 15, as suggested in [65].

Appendix D

Hyper Parameters Tuning in BR-rerank Experiments

- BR: elastic-net regularization penalty strength $\lambda \in \{0.0001, 0.000001\}$; L1 ratio $\in \{0.1, 0.5\}$; training iteration is decided by monitoring the validation performance after each iteration.

- GB calibrator: shrinkage = 0.1; regression tree weak learner has 10 leaves; the minimum number of instances per leaf is 5; training iteration is tuned on validation set.

- BR-rerank: apart from setting the hyper parameters mentioned in GB calibrator, we also set the number of candidates $K = 10$.

- 2BR and DBR: we use the validation data to decide whether the stage-2 model should take instance features $\mathbf{x}$ as par of the input; for the GB base learner, we tune the number of training iterations using validation data; we set shrinkage = 0.1; regression tree weak learner has 10 leaves; the minimum number of instances per leaf is 5.

- Probabilistic Classifier Chains (PCC): following the common practice, labels are arranged by decreasing frequency; for the GB base learner, we tune the number of training iterations using validation data; we set shrinkage = 0.1; regression tree weak learner has 10 leaves; the minimum number of instances per leaf is 5; for prediction, we use beam search with width 5.

- Random k-label-sets (RAKEL): The number of labels used in each labelset $K \in \{2, 4, 6, \ldots, 30\}$.

APPENDIX D. HYPER PARAMETERS TUNING IN BR-RERANK EXPERIMENTS

- Multi-label k Nearest Neighbors (KNN): number of neighbors $k \in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21\}$; the smoothing parameter $s \in \{0.5, 0.7, 1.0\}$.
- Deep Value Network (DVN): whether or not to includes a linear layer after the two-layer perceptron.
- Predict and Constrain (PC): number of hidden units for the linear part $h \in \{100, 150, 200\}$.
- PD-Sparse (PDS): L1 regularization weight, $\lambda \in \{1, 0.1, 0.01, 0.001, 0.0001\}$; in order to predict a subset of labels as opposed to simply ranking labels, we tune the score threshold.
- SPEN: we use the hidden layer sizes mentioned in the original paper.

Appendix E

Experiments with Monotonicity and Another GB Variant

We conducted additional experiments on imposing partial monotonicity in GB. Since the GB score is the sum of regression tree scores, it suffices to impose monotonicity constraints in each tree. We follow the method implemented in the xgboost package [21] and it works as follows: We associate a lower bound and an upper bound with each tree node (including intermediate nodes). As the tree grows, each node first inherits its parent's bounds and then tightens the bounds as new constraints are introduced. A leaf always outputs a value within its bounds. When a node is split into two children by a monotonic feature, assuming data with smaller feature value goes left, we compute the middle point between the output of the left child and the output of the right child (they are treated as leaves), and set the upper bound of the left child and the lower bound of the right child to be this middle point. When a node is split with a non-monotonic feature, its children just inherit its bounds without further tightening.

We also tested anther variant of GB, named GB-KL, which has a sigmoid transformation on top of the ensemble score and is trained by minimizing KL divergence. We compare it with GB-MSE, which does not employ the sigmoid transformation and is trained by minimizing square error.

We tested all 4 configurations of calibrator and monotonicity and the results are shown in Tables E.1,E.2,E.3,E.4. All 4 configurations have similar calibration and prediction performance.

Table E.1: BR prediction calibration performance in terms of MSE (the smaller the better).

calibrator	monotonicity	BIBTEX	OHSUMED	RCV1	TMC	WISE	MSCOCO
GB-MSE	no	0.0682	0.1889	0.1229	0.1800	0.1472	0.1434
GB-MSE	yes	0.0653	0.1854	0.1227	0.1783	0.1468	0.1438
GB-KL	no	0.0696	0.1904	0.1260	0.1794	0.1476	0.1437
GB-KL	yes	0.0666	0.1850	0.1253	0.1772	0.1467	0.1436

Table E.2: BR prediction calibration performance in terms of sharpness (the bigger the better).

calibrator	monotonicity	BIBTEX	OHSUMED	RCV1	TMC	WISE	MSCOCO
GB-MSE	no	0.0719	0.0467	0.1262	0.0319	0.1022	0.0825
GB-MSE	yes	0.0783	0.0475	0.1366	0.0347	0.1029	0.0823
GB-KL	no	0.0753	0.0475	0.1236	0.0330	0.1014	0.0824
GB-KL	yes	0.0782	0.0496	0.1299	0.0351	0.1028	0.0824

Table E.3: Prediction performance of BR-rerank in terms of set accuracy.

calibrator	monotonicity	BIBTEX	OHSUMED	RCV1	TMC	WISE	MSCOCO
GB-MSE	no	21.5	42.0	53.2	33.3	60.5	35.9
GB-MSE	yes	22.1	41.9	52.7	32.9	60.4	36.0
GB-KL	no	21.1	41.1	51.8	33.0	60.3	36.2
GB-KL	yes	22.1	41.4	52.2	33.1	60.5	36.1

Table E.4: Prediction performance of BR-rerank in terms of F1 score.

calibrator	monotonicity	BIBTEX	OHSUMED	RCV1	TMC	WISE	MSCOCO
GB-MSE	no	42.2	67.5	78.8	66.8	75.4	73.2
GB-MSE	yes	43.1	67.2	78.6	66.6	75.5	73.4
GB-KL	no	41.8	66.8	78.2	66.4	75.1	73.3
GB-KL	yes	42.4	67.1	78.2	66.5	75.3	73.3

Bibliography

[1] http://feeds.soundcloud.com/stream/316591586-twiml-twiml-talk-018-part-4-behold-ai-increasing-efficiency-healthcare-insurance-billing.mp3.

[2] https://storage.googleapis.com/openimages/web/download.html.

[3] https://research.google.com/youtube8m/index.html.

[4] http://manikvarma.org/downloads/XC/XMLRepository.html.

[5] http://cocodataset.org/.

[6] http://meka.sourceforge.net/.

[7] https://www.kaggle.com/c/wise-2014/data/.

[8] http://mulan.sourceforge.net/.

[9] http://kt.ijs.si/DragiKocev/PhD/resources/doku.php?id=hmc_classification.

[10] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*, 2016.

[11] Javed A Aslam. *Noise tolerant algorithms for learning and searching*. PhD thesis, Massachusetts Institute of Technology, 1995.

[12] Javed A Aslam and Scott E Decatur. Improved noise-tolerant learning and generalized statistical queries. 1994.

[13] Miriam Ayer, H Daniel Brunk, George M Ewing, William T Reid, and Edward Silverman. An empirical distribution function for sampling with incomplete information. *The annals of mathematical statistics*, pages 641–647, 1955.

[14] Rohit Babbar and Bernhard Shoelkopf. Dismec-distributed sparse machines for extreme multi-label classification. *arXiv preprint arXiv:1609.02521*, 2016.

[15] David Belanger and Andrew McCallum. Structured prediction energy networks. In *Proceedings of the International Conference on Machine Learning*, 2016.

[16] Alexander C Berg, Tamara L Berg, Hal Daume, Jesse Dodge, Amit Goyal, Xufeng Han, Alyssa Mensch, Margaret Mitchell, Aneesh Sood, Karl Stratos, et al. Understanding and predicting importance in images. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3562–3569. IEEE, 2012.

[17] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems*, pages 730–738, 2015.

[18] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[19] Nataly Brukhim and Amir Globerson. Predict and constrain: Modeling cardinality in deep structured prediction. *arXiv preprint arXiv:1802.04721*, 2018.

[20] Serhat S Bucak, Pavan Kumar Mallapragada, Rong Jin, and Anil K Jain. Efficient multi-label ranking for multi-class learning: application to object recognition. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2098–2105. IEEE, 2009.

[21] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[22] Tongfei Chen, Jiří Navrátil, Vijay Iyengar, and Karthikeyan Shanmugam. Confidence scoring using whitebox meta-models with linear classifier probes. *arXiv preprint arXiv:1805.05396*, 2018.

[23] Yao-Nan Chen and Hsuan-Tien Lin. Feature-aware label space dimension reduction for multi-label classification. In *Advances in Neural Information Processing Systems*, pages 1529–1537, 2012.

[24] Weiwei Cheng, Eyke Hüllermeier, and Krzysztof J Dembczynski. Bayes optimal multilabel classification via probabilistic classifier chains. In *Proceedings of the*

27th international conference on machine learning (ICML-10), pages 279–286, 2010.

[25] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In *Proceedings of the ACM international conference on image and video retrieval*, page 48. ACM, 2009.

[26] Moustapha Cissé, COM Maruan Al-Shedivat, and Samy Bengio. Adios: Architectures deep in output space. In *Proceedings of the 33rd International Conference on Machine Learning, ICML*, 2016.

[27] Moustapha M Cisse, Nicolas Usunier, Thierry Artieres, and Patrick Gallinari. Robust bloom filters for large multilabel classification tasks. In *Advances in Neural Information Processing Systems*, pages 1851–1859, 2013.

[28] Michael Collins and Terry Koo. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70, 2005.

[29] Stijn Decubber, Thomas Mortier, Krzysztof Dembczyński, and Willem Waegeman. Deep f-measure maximization in multi-label classification: A comparative study. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 290–305. Springer, 2018.

[30] Krzysztof Dembczynski, Arkadiusz Jachnik, Wojciech Kotlowski, Willem Waegeman, and Eyke Hüllermeier. Optimizing the f-measure in multi-label classification: Plug-in rule approach versus structured loss minimization. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1130–1138, 2013.

[31] Krzysztof Dembczyński, Wojciech Kotłowski, Willem Waegeman, Róbert Busa-Fekete, and Eyke Hüllermeier. Consistency of probabilistic classifier trees. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 511–526. Springer, 2016.

[32] Krzysztof Dembczyński, Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier. On label dependence and loss minimization in multi-label classification. *Machine Learning*, 88(1-2):5–45, 2012.

[33] Krzysztof Dembczyński, Willem Waegeman, and Eyke Hüllermeier. Joint mode estimation in multi-label classification by chaining. In *CoLISD 2011 (Collective Learning and Inference on Structured Data 2011)*, 2011.

[34] Krzysztof Dembczynski, Willem Waegeman, and Eyke Hüllermeier. An analysis of chaining in multi-label classification. In *ECAI*, pages 294–299, 2012.

[35] Jia Deng, Nan Ding, Yangqing Jia, Andrea Frome, Kevin Murphy, Samy Bengio, Yuan Li, Hartmut Neven, and Hartwig Adam. Large-scale object classification using label relation graphs. In *Computer Vision–ECCV 2014*, pages 48–64. Springer, 2014.

[36] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences*, 66(4):614–656, 2003.

[37] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[38] Zheyun Feng, Songhe Feng, Rong Jin, and Anil K Jain. Image tag completion by noisy matrix recovery. In *European Conference on Computer Vision*, pages 424–438. Springer, 2014.

[39] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[40] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[41] Johannes Fürnkranz, Eyke Hüllermeier, Eneldo Loza Mencía, and Klaus Brinker. Multilabel classification via calibrated label ranking. *Machine learning*, 73(2):133–153, 2008.

[42] Maxime Gasse and Alex Aussem. F-measure maximization in multi-label classification with conditionally independent label subsets. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 619–631. Springer, 2016.

[43] Maxime Gasse, Alexandre Aussem, and Haytham Elghazel. On the optimality of multi-label classification under subset zero-one loss for distributions satisfying the composition property. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2531–2539, 2015.

[44] Nadia Ghamrawi and Andrew McCallum. Collective multi-label classification. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 195–200. ACM, 2005.

[45] Eva Gibaja and Sebastián Ventura. A tutorial on multilabel learning. *ACM Computing Surveys (CSUR)*, 47(3):52, 2015.

[46] Tilmann Gneiting, Fadoua Balabdaoui, and Adrian E Raftery. Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(2):243–268, 2007.

[47] Shantanu Godbole and Sunita Sarawagi. Discriminative methods for multi-labeled classification. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 22–30. Springer, 2004.

[48] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[49] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. *arXiv preprint arXiv:1706.04599*, 2017.

[50] Yuhong Guo and Suicheng Gu. Multi-label classification using conditional dependency networks. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[51] Maya R Gupta and Yihua Chen. *Theory and use of the EM algorithm*. Now Publishers Inc, 2011.

[52] Michael Gygli, Mohammad Norouzi, and Anelia Angelova. Deep value networks learn to evaluate and iteratively refine structured outputs. *arXiv preprint arXiv:1703.04363*, 2017.

[53] Daniel Hsu, Sham Kakade, John Langford, and Tong Zhang. Multi-label prediction via compressed sensing. In *NIPS*, volume 22, pages 772–780, 2009.

[54] Kuan-Hao Huang and Hsuan-Tien Lin. Cost-sensitive label embedding for multi-label classification. *Machine Learning*, 106(9-10):1725–1746, 2017.

[55] Hamid Izadinia, Bryan C Russell, Ali Farhadi, Matthew D Hoffman, and Aaron Hertzmann. Deep classifiers from image tags in the wild. In *Proceedings of the 2015 Workshop on Community-Organized Multimodal Mining: Opportunities for Novel Solutions*, pages 13–18. ACM, 2015.

[56] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[57] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–944. ACM, 2016.

[58] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

[59] Atsushi Kanehira and Tatsuya Harada. Multi-label ranking from positive and unlabeled data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5138–5146, 2016.

[60] Dongwoo Kim, Suin Kim, and Alice Oh. Dirichlet process with mixed random measures: a nonparametric topic model for labeled data. *arXiv preprint arXiv:1206.4658*, 2012.

[61] Oluwasanmi O Koyejo, Nagarajan Natarajan, Pradeep K Ravikumar, and Inderjit S Dhillon. Consistent multilabel classification. In *Advances in Neural Information Processing Systems*, pages 3303–3311, 2015.

[62] Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate uncertainties for deep learning using calibrated regression. *arXiv preprint arXiv:1807.00263*, 2018.

[63] Volodymyr Kuleshov and Percy S Liang. Calibrated structured prediction. In *Advances in Neural Information Processing Systems*, pages 3474–3482, 2015.

[64] Abhishek Kumar, Shankar Vembu, Aditya Krishna Menon, and Charles Elkan. Learning and inference in probabilistic classifier chains with beam search. In *Machine Learning and Knowledge Discovery in Databases*, pages 665–680. Springer, 2012.

[65] Abhishek Kumar, Shankar Vembu, Aditya Krishna Menon, and Charles Elkan. Beam search algorithms for multilabel learning. *Machine learning*, 92(1):65–89, 2013.

[66] Paul Felix Lazarsfeld, Neil W Henry, and Theodore Wilbur Anderson. *Latent structure analysis*. Houghton Mifflin Boston, 1968.

[67] Cheng Li, Virgil Pavlu, Javed A. Aslam, Bingyu Wang, and Kechen Qin. Learning to calibrate and rerank multi-label predictions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2019. To appear.

[68] Cheng Li, Bingyu Wang, Virgil Pavlu, and Javed A. Aslam. Conditional bernoulli mixtures for multi-label classification. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 2482–2491, 2016.

[69] Chun-Liang Li and Hsuan-Tien Lin. Condensed filter tree for cost-sensitive multi-label classification. In *ICML*, pages 423–431, 2014.

[70] Xirong Li, Tiberio Uricchio, Lamberto Ballan, Marco Bertini, Cees GM Snoek, and Alberto Del Bimbo. Image tag assignment, refinement and retrieval. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 1325–1326. ACM, 2015.

[71] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124. ACM, 2017.

[72] Liping Liu and Thomas G Dietterich. A conditional multinomial mixture model for superset label learning. In *Advances in neural information processing systems*, pages 557–565, 2012.

[73] Weiwei Liu and Ivor Tsang. On the optimality of classifier chain for multi-label classification. In *Advances in Neural Information Processing Systems*, pages 712–720, 2015.

[74] Jon D McAuliffe and David M Blei. Supervised topic models. In *Advances in neural information processing systems*, pages 121–128, 2008.

[75] Andrew McCallum. Multi-label text classification with a mixture model trained by EM. In *AAAI'99 workshop on text learning*, pages 1–7, 1999.

[76] Geoffrey McLachlan and David Peel. *Finite mixture models*. John Wiley & Sons, 2004.

[77] Deiner Mena, Elena Montañés, José R Quevedo, and Juan José Del Coz. Using A* for inference in probabilistic classifier chains. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 3707–3713. AAAI Press, 2015.

[78] Ishan Misra, C Lawrence Zitnick, Margaret Mitchell, and Ross Girshick. Seeing through the human reporting bias: Visual classifiers from noisy human-centric labels.

[79] Volodymyr Mnih, Hugo Larochelle, and Geoffrey E Hinton. Conditional restricted boltzmann machines for structured output prediction. *arXiv preprint arXiv:1202.3748*, 2012.

[80] Elena Montañes, Robin Senge, Jose Barranquero, José Ramón Quevedo, Juan José del Coz, and Eyke Hüllermeier. Dependent binary relevance models for multi-label classification. *Pattern Recognition*, 47(3):1494–1508, 2014.

[81] Jinseok Nam, Jungi Kim, Eneldo Loza Mencía, Iryna Gurevych, and Johannes Fürnkranz. Large-scale multi-label text classification—revisiting neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 437–452. Springer, 2014.

[82] Jinseok Nam, Eneldo Loza Mencía, Hyunwoo J Kim, and Johannes Fürnkranz. Maximizing subset accuracy with recurrent neural networks in multi-label classification. In *Advances in Neural Information Processing Systems*, pages 5413–5423, 2017.

[83] Ye Nan, Kian Ming Chai, Wee Sun Lee, and Hai Leong Chieu. Optimizing f-measure: A tale of two approaches. *arXiv preprint arXiv:1206.4625*, 2012.

[84] Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. Learning with noisy labels. In *Advances in neural information processing systems*, pages 1196–1204, 2013.

[85] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[86] Stefanie Nowak and Stefan Rüger. How reliable are annotations via crowdsourcing: a study about inter-annotator agreement for multi-label image annotation. In *Proceedings of the international conference on Multimedia information retrieval*, pages 557–566. ACM, 2010.

[87] Shameem Puthiya Parambath, Nicolas Usunier, and Yves Grandvalet. Optimizing f-measures by cost-sensitive classification. In *Advances in Neural Information Processing Systems*, pages 2123–2131, 2014.

[88] Sang-Hyeun Park and Johannes Fürnkranz. Multi-label classification with label constraints. In *ECML PKDD 2008 Workshop on Preference Learning*, pages 157–171, 2008.

[89] James Petterson and Tibério S Caetano. Reverse multi-label learning. In *Advances in Neural Information Processing Systems*, pages 1912–1920, 2010.

[90] James Petterson and Tibério S Caetano. Submodular multi-label learning. In *Advances in Neural Information Processing Systems*, pages 1512–1520, 2011.

[91] Ignazio Pillai, Giorgio Fumera, and Fabio Roli. Designing multi-label classifiers that maximize f measures: State of the art. *Pattern Recognition*, 61:394–404, 2017.

[92] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[93] Yashoteja Prabhu and Manik Varma. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 263–272. ACM, 2014.

[94] Antti Puurula. Mixture models for multi-label text classification. In *10th New Zealand Computer Science Research Student Conference*, 2011.

[95] Kechen Qin, Cheng Li, Virgil Pavlu, and Javed Aslam. Adapting rnn sequence prediction model to multi-label set prediction. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3181–3190, 2019.

[96] Daniel Ramage, David Hall, Ramesh Nallapati, and Christopher D Manning. Labeled lda: A supervised topic model for credit attribution in multi-labeled corpora. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 248–256. Association for Computational Linguistics, 2009.

[97] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333–359, 2011.

[98] Tim Robertson. Order restricted statistical inference. Technical report, 1988.

[99] S Sasabuchi, M Inutsuka, and DDS Kulatunga. A multivariate version of isotonic regression. *Biometrika*, 70(2):465–472, 1983.

[100] Robert E Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine learning*, 39(2):135–168, 2000.

[101] Libin Shen, Anoop Sarkar, and Franz Josef Och. Discriminative reranking for machine translation. In *HLT-NAACL 2004*, 2004.

[102] Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. *Introduction to statistical relational learning*, pages 93–128, 2006.

[103] Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *Machine Learning*, 4(4):267–373, 2011.

[104] Farbound Tai and Hsuan-Tien Lin. Multilabel classification with principal label space transformation. *Neural Computation*, 24(9):2508–2542, 2012.

[105] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(Sep):1453–1484, 2005.

[106] Grigorios Tsoumakas, Anastasios Dimou, Eleftherios Spyromitros, Vasileios Mezaris, Ioannis Kompatsiaris, and Ioannis Vlahavas. Correlation-based pruning of stacked binary relevance models for multi-label learning. In *Proceedings of the 1st international workshop on learning from multi-label data*, pages 101–116, 2009.

[107] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *Int J Data Warehousing and Mining*, 2007:1–13, 2007.

[108] Grigorios Tsoumakas and Ioannis Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *Machine learning: ECML 2007*, pages 406–417. Springer, 2007.

[109] Naonori Ueda and Kazumi Saito. Parametric mixture models for multi-labeled text. In *Advances in neural information processing systems*, pages 721–728, 2002.

[110] Arash Vahdat. Toward robustness against label noise in training deep discriminative neural networks. *arXiv preprint arXiv:1706.00038*, 2017.

[111] Andreas Veit, Neil Alldrin, Gal Chechik, Ivan Krasin, Abhinav Gupta, and Serge Belongie. Learning from noisy large-scale datasets with minimal supervision. *arXiv preprint arXiv:1701.01619*, 2017.

[112] Willem Waegeman, Krzysztof Dembczyński, Arkadiusz Jachnik, Weiwei Cheng, and Eyke Hüllermeier. On the bayes-optimality of f-measure maximizers. *The Journal of Machine Learning Research*, 15(1):3333–3388, 2014.

[113] Bingyu Wang, Li Chen, Wei Sun, Kechen Qin, Kefeng Li, and Hui Zhou. Ranking-based autoencoder for extreme multi-label classification. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2820–2830, 2019.

[114] Bingyu Wang, Cheng Li, Virgil Pavlu, and Jay Aslam. A pipeline for optimizing f1-measure in multi-label text classification. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 913–918. IEEE, 2018.

[115] Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. Cnn-rnn: A unified framework for multi-label image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2285–2294, 2016.

[116] WISE. Greek media monitoring multilabel classification (wise 2014), July 2014. `www.kaggle.com/c/wise-2014`.

[117] Baoyuan Wu, Siwei Lyu, and Bernard Ghanem. Ml-mg: multi-label learning with missing labels using a mixed graph. In *Proceedings of the IEEE international conference on computer vision*, pages 4157–4165, 2015.

[118] Baoyuan Wu, Siwei Lyu, and Bernard Ghanem. Constrained submodular minimization for missing labels and class imbalance in multi-label learning. In *AAAI*, pages 2229–2236, 2016.

[119] Pengtao Xie, Ruslan Salakhutdinov, Luntian Mou, and Eric P Xing. Deep determinantal point process for large-scale multi-label classification. In *ICCV*, pages 473–482, 2017.

[120] Chang Xu, Dacheng Tao, and Chao Xu. Robust extreme multi-label learning. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA August*, pages 13–17, 2016.

[121] Yan Yan, Glenn Fung, Jennifer G Dy, and Romer Rosales. Medical coding classification by leveraging inter-code relationships. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 193–202. ACM, 2010.

[122] Shuang-Hong Yang, Hongyuan Zha, and Bao-Gang Hu. Dirichlet-bernoulli alignment: A generative model for multi-class multi-label multi-instance

corpora. In *Advances in neural information processing systems*, pages 2143–2150, 2009.

[123] Yelp. Automatically categorizing yelp businesses, Sep 2015. https://engineeringblog.yelp.com/amp/2015/09/automatically-categorizing-yelp-businesses.html.

[124] Ian EH Yen, Xiangru Huang, Kai Zhong, Pradeep Ravikumar, and Inderjit S Dhillon. Pd-sparse: A primal and dual sparse approach to extreme multiclass and multilabel classification. In *Proceedings of the 33nd International Conference on Machine Learning*, 2016.

[125] Hsiang-Fu Yu, Prateek Jain, Purushottam Kar, and Inderjit S Dhillon. Large-scale multi-label learning with missing labels. In *ICML*, pages 593–601, 2014.

[126] Seniha Esen Yuksel, Joseph N Wilson, and Paul D Gader. Twenty years of mixture of experts. *Neural Networks and Learning Systems, IEEE Transactions on*, 23(8):1177–1193, 2012.

[127] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *KDD*, pages 694–699. ACM, 2002.

[128] Min-Ling Zhang and Lei Wu. Lift: Multi-label learning with label-specific features. *IEEE transactions on pattern analysis and machine intelligence*, 37(1):107–120, 2015. Code: `http://cse.seu.edu.cn/PersonalPage/zhangml`.

[129] Min-Ling Zhang and Kun Zhang. Multi-label learning by exploiting label dependency. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 999–1008. ACM, 2010.

[130] Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition*, 40(7):2038–2048, 2007.

[131] Feipeng Zhao and Yuhong Guo. Semi-supervised multi-label learning with incomplete labels. In *IJCAI*, pages 4062–4068, 2015.

[132] Tianyi Zhou, Dacheng Tao, and Xindong Wu. Compressed labeling on distilled labelsets for multi-label learning. *Machine Learning*, 88(1-2):69–126, 2012.