2015-07-01

An Improved Classifier Chain Ensemble for Multi-DimensionalClassification with Conditional Dependence

Joseph Ethan Heydorn
Brigham Young University - Provo

An Improved Classifier Chain Ensemble for Multi-Dimensional Classification with Conditional Dependence

Joseph Ethan Heydorn

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Tony Martinez, Chair
Eric Ringger
Seth Holladay

Department of Computer Science

Brigham Young University

July 2015

ABSTRACT

An Improved Classifier Chain Ensemble for Multi-Dimensional Classification with Conditional Dependence

Joseph Ethan Heydorn
Department of Computer Science, BYU
Master of Science

We focus on multi-dimensional classification (MDC) problems with conditional dependence, which we call multiple output dependence (MOD) problems. MDC is the task of predicting a vector of categorical outputs for each input. Conditional dependence in MDC means that the choice for one output value affects the choice for others, so it is not desirable to predict outputs independently. We show that conditional dependence in MDC implies that a single input can map to multiple correct output vectors. This means it is desirable to find multiple correct output vectors per input. Current solutions for MOD problems are not sufficient because they predict only one of the correct output vectors per input, ignoring all others. We modify four existing MDC solutions, including chain classifiers, to predict multiple output vectors. We further create a novel ensemble technique named weighted output vector ensemble (WOVE) which combines these multiple predictions from multiple chain classifiers in a way that preserves the integrity of output vectors and thus preserves conditional dependence among outputs. We verify the effectiveness of WOVE by comparing it against 7 other solutions on a variety of data sets and find that it shows significant gains over existing methods.

Keywords: multi-dimensional classification, multi-target, conditional dependence, ensemble, chain classifier

Contents

1 Introduction **1**

2 Related Work **5**

3 Chain Classifiers **7**

3.1 Predicting 7

3.2 Training 8

4 Weighted Output Vector Ensemble **9**

5 Model Adaptations For Predicting Multiple Output Vectors **10**

5.1 Chain Classifiers 11

5.2 Monolithic Tag Technique 12

5.3 Independent Classifiers 12

5.4 Hierarchical Multi-Output Nearest Neighbor 13

6 Experiments **13**

6.1 Data Sets 14

6.2 Evaluation Measure 19

6.3 Solutions to Compare 20

6.4 Experiment Setup 20

6.4.1 Base classifiers 21

6.5 Results 22

7 Other Related Work **24**

7.1 Multi-Dimensional Classification 29

7.2 Multi-label Classification 30

7.3 Multivariate Regression 30

7.4 Multi-task Learning 30

7.5 Structured Prediction 31

8 Conclusions and Future Work **31**

A Proof of Theorem 1 **32**

1 Introduction

A typical goal in machine learning is classification, in which a single class value is predicted for each input. A more complex task is multi-dimensional classification (MDC), in which multiple class values are predicted for each input, forming a vector of output values. If these outputs depend on each other, than the prediction of one output depends on the predictions for others. There may also be multiple correct vectors of outputs values for each input. In this case, solutions which predict each output independently will not perform well. When outputs are inter-dependent given an input, we call this type of problem multiple output dependence (MOD).

Following is an example of a MOD problem. Suppose that, given information about a person and the current outdoor temperature, we want to give a list of outfits they might want to wear today. Each outfit consists of pants, a shirt, and a coat. If a model predicts each output independently, it may choose a pair of pants or a coat that do not go with the predicted shirt. Some combinations just do not go together. If it instead chooses a pair of pants first, then a shirt, the choice of shirt may change, depending on the chosen pair of pants. Additionally, if the temperature is cold, the person might prefer warmer combinations of pants, shirts, and coats. We want a solution which predicts a list of output vectors because there should be multiple good outfits for a particular person and weather combination.

MOD is a special case of multi-dimensional classification (MDC). The goal of MDC is to approximate a function $f : \mathcal{X}_1 \times \mathcal{X}_2 \times ... \times \mathcal{X}_m \rightarrow \mathcal{Y}_1 \times \mathcal{Y}_2 \times ... \times \mathcal{Y}_d$ with m inputs and d outputs. Inputs can be categorical or real-valued, and outputs are categorical. An output vector[1] $\mathbf{y}$ is a vector of values $\langle y_1, y_2, ..., y_d \rangle$, where the domain of each y_i is $\mathcal{Y}_i$ (where $1 \leq i \leq d$). To be clear, we will refer to each output dimension $\mathcal{Y}_1$ through $\mathcal{Y}_d$ as *output dimensions* or *outputs*. We will refer to each value y_1 through y_d as *output values*, and we will refer to $\mathbf{y}$ as an *output vector*. Multi-label classification (MLC) is also a special case of MDC in which each input can be mapped to multiple labels (i.e., each maps to a set of

[1]Vectors are in bold.

labels). Zaragoza et al. [1] show that MLC problems can be transformed into MDC problems by creating a single binary output for each label, indicating whether it is present for a given input.

For a problem of interest, let there exist a training set T used to train a model, where instance i ($1 \leq i \leq |T|$) shows input vector $\mathbf{x}_i$ labeled with an output vector $\mathbf{y}_i$. Let there also exist a test set D, ideally from the same distribution, used to test the model's ability to generalize to unseen instances.

There are two types of statistical dependence that can exist among outputs: unconditional dependence and conditional dependence. Unconditional dependence exists between two outputs Y_i and Y_j when[2] $P(Y_i) \neq P(Y_i|Y_j)$. This means that knowing either output Y_i or Y_j tells us something about the other. Two outputs are conditionally dependent when, for some observed input $\mathbf{x}$, $P(Y_i|\mathbf{x}) \neq P(Y_i|Y_j, \mathbf{x})$. This means that when $\mathbf{x}$ is observed, then knowing either Y_i or Y_j tell us something about the other. Dembszynski et al. [2] discuss these two types of dependence in detail. They show that outputs can be conditionally independent yet unconditionally dependent, and that they can be conditionally dependent and yet unconditionally independent. Thus conditional dependence does not imply unconditional dependence, nor vice versa. Read et al. [3] explain that although unconditional dependence may be useful for building a classifier, only conditional dependence considers the input (in a data instance), so it is of greater interest. For this reason, we focus on conditional dependence in this work.

Morris and Martinez [4] show that the conditional dependence in MOD implies the existence of inputs that map to multiple different output vectors. If an input maps to multiple output vectors, it is represented by multiple instances in the data set, labeled with different output values. Theorem 1 is a revised version of their theorem that avoids defining probabilities using frequencies of categorical values in the data set and that is applicable

[2]When referring to random variables, we use upper case to refer to random variables, and we use lower case to refer to observed values of random variables, so $\mathbf{x}$ is an observed value from X. Probability statements involving random variables are to be interpreted distributionally, i.e., for all values of the random variable.

when the input random variable X is continuous or categorical. Theorem 1 states, given some input $\mathbf{x}$, if two outputs are conditionally dependent, then there is a non-zero probability of multiple distinct output vectors. This means that if we held $\mathbf{x}$ fixed and sampled multiple times from $P(Y_1, Y_2|\mathbf{x})$, we would expect to draw multiple different output vectors. This indicates that training set T may include the same or similar inputs labeled with multiple output vectors. Thus, the goal of MOD is to approximate a relation, not a function. The proof of the theorem is found in Appendix A.

Theorem 1. *Given continuous (or categorical) input random variable X and categorical output random variables Y_1 and Y_2, if Y_1 is conditionally dependent on Y_2 given an observed $\boldsymbol{x}$, then the probability distribution $P(Y_1, Y_2|\boldsymbol{x})$ assigns non-zero probability to at least 2 distinct output vectors $\boldsymbol{y}$ and $\boldsymbol{y'}$.*

The converse of Theorem 1 is not true. It would state that if there is non-zero probability that $\mathbf{x}$ maps to multiple output vectors, then $Y1$ and $Y2$ must be conditionally dependent. It may be the case that given $\mathbf{x}$ there is non-zero probability of observing multiple output vectors in T simply because of random variation in the inputs, random variation in the output values added independently of the input or other outputs, or it may even be the case that any value of $\mathbf{y}$ is correct. In either case, there may not necessarily be conditional dependence among the outputs.

Often in machine learning, when a single input is seen mapped to multiple different outputs (output vectors), one of the outputs is selected as the "correct" one, and all others are considered to be noise. In MOD, however, there may be multiple correct output vectors for an input, and we wish to find them all. We wish to predict a ranked list of correct output vectors for any given input, with those that are more compatible with the input being ranked higher on the list. Output vectors which occur often with an input should be ranked higher than those that occur rarely.

For a general example of a MOD problem, suppose T is a data set in which an input $\mathbf{x}$ is often seen mapped to 2 different output vectors, $\mathbf{x} \rightarrow \langle a, b, c \rangle$, and $\mathbf{x} \rightarrow \langle x, y, z \rangle$. When

the model encounters $\mathbf{x}$ in the test set, it should output either $\langle a, b, c \rangle$ or $\langle x, y, z \rangle$, or both. If it predicts each output independently, however, it might predict $\langle a, y, z \rangle$, $\langle x, b, c \rangle$, or any other combination of the two output vectors. This is not desirable. The model should use the training set to learn which combinations of output values go together and which do not. If the input contains real values, it is unlikely that the same $\mathbf{x}$ will occur more than once in T. In this case we would expect similar values of $\mathbf{x}$ to map to the same set of output vectors.

A real-world example of such a MOD task can be found at the company InsideSales.com. Given information about a sales contact, the task is to predict who should make the contact, how to contact them, and how long to wait before doing so. All of these decisions depend on each other, and there may be multiple good responses.

To solve MOD problems we modify chain classifiers, a popular model for predicting multiple outputs, to predict scored lists of output vectors. Furthermore, we combine scored lists from multiple chain classifiers using a novel ensemble technique named weighted output vector ensemble (WOVE), which preserves the integrity of predicted output vectors, thus preserving conditional dependence and improving predictive performance over weighted voting of each output independently. WOVE also produces a ranked list of predictions. We also show how to adapt three other MDC solutions (four, including chain classifiers) to solve MOD problems by predicting multiple output vectors.

Section 2 discusses related work which we compare WOVE against. Section 3 describes chain classifiers for those not familiar with them. In Section 4 we discuss WOVE. Section 5 describes our model adaptations for predicting multiple output vectors. Section 6 gives experimental results which show significant advantages of WOVE compared to state of the art methods, as well as improvements in other models from the adaptations discussed in Section 5. Section 7 discusses other related work.

2 Related Work

This section discusses several solutions to MDC which we also compare against in Section 6, although these solutions predict only 1 output vector. In Section 7 we discuss other related work which we do not compare against, because they are either not directly applicable or because time and resources do not allow it. We choose to compare our methods against solutions which are either a good baseline or which show promising results on multiple data sets.

Read et al. [5] introduce chain classifiers, which are used to capture conditional dependence in MLC problems [5] and MDC problems [3, 6]. Chain classifiers require that the outputs have an order. They predict each output in sequence, using the predicted values from previous outputs to predict the next output in the chain. More details are given in Section 3. Read et al. [5] handle the issue of determining output order by creating an ensemble of classifier chains (ECC), each with a random order. ECC combines output vectors from each classifier by selecting the output value for each output that maximizes the scores given by each model to each possible value for that output. It maximizes each output separately, so the resulting output vector may not have been predicted by any sub-model. ECC also uses bagging with every sub-model in the ensemble. We compare WOVE against ECC.

In a later work, Read et al. [6] use a Monte Carlo search technique both to determine chain order while training and to determine output values when predicting. The technique they use to make predictions in an ensemble does preserve the conditional dependence among outputs learned by each chain classifier, but it does not combine multiple output vectors as does WOVE. Instead it generates many output vectors from each chain classifier and keeps the one with the highest score (highest probability). They propose several variations of their model. They use MDC data sets which are not MLC, as well as MLC data sets. We compare WOVE against their best performing ensemble variation, namely PM$_s$CC, which out-performs ECC.

Tsoumakas and Katakis [7] describe several methods for transforming multi-label

classification problems into single-output problems. One method creates an output value for each distinct output vector (set of label values) in training set T. A multiclass classifier is then trained to predict the output values. When predicting, the output values are transformed back to the vectors they represent. This technique has limitations, however. It creates many more labels than were in the original data set (as many as 2^d), which can make learning very slow. It loses information about dependencies between outputs by hiding the original output dimensions from the learning algorithm. It also gives the learner no way to generalize to output vectors not seen in the training set, though it is an open question as to whether that would be desirable. McClanahan et al. [8] use this technique when making predictions on a data set which has MDC structure, and show improved results over predicting each output independently. They refer to each combined output as a "monolithic tag". We refer to it as MONO. Section 5 shows how to predict multiple output vectors with this technique as well as with three other methods, and Section 6 compares it against other methods.

Read et al. [3] present ESC, a model which partitions the output dimensions into subsets based on a measurement of conditional dependence among them. They then use multi-label classifiers to predict each partition. They combine several such models into an ensemble. They show improved results over ECC. We also compare WOVE against ESC.

Morris and Martinez [4] work on learning dependence in MOD problems. They build a model named HMONN (hierarchical multi-output nearest neighbor) which uses two layers of sub-models. In the first layer, a separate MLP is trained using error backpropagation to predict each output independently. In the second layer, a k-nearest neighbor approach is used to find the k instances in the training set which are closest to the input/output pair predicted by the independent models. The final output vector is the one which occurs most in the k nearest instances. The KNN approach constrains the model to return an output vector in the training set, thus avoiding invalid combinations when predicting. Their model returns only one output vector, but Section 5 shows a way to modify it to predict multiple output vectors, and Section 6 shows results comparing it against other models.

3 Chain Classifiers

Chain classifiers are a model introduced by Read et al. [5] for capturing conditional dependence among multiple outputs. They use multiple sub-models to learn conditional dependence between outputs in multi-label and multi-dimensional classification problems.

For a d-output problem, a chain classifier trains d models $M_1 ... M_d$, to predict each output. Model M_i uses as input both $\mathbf{x}$ and all predictions from models M_1 through M_{i-1}. Any type of multiclass classifier can be used. Figure 1 shows a diagram of the structure of a chain classifier for a 3-output problem. In Figure 1 and the present discussion, $\mathbf{M}$ is a vector containing models $\langle M_1, ..., M_d \rangle$.

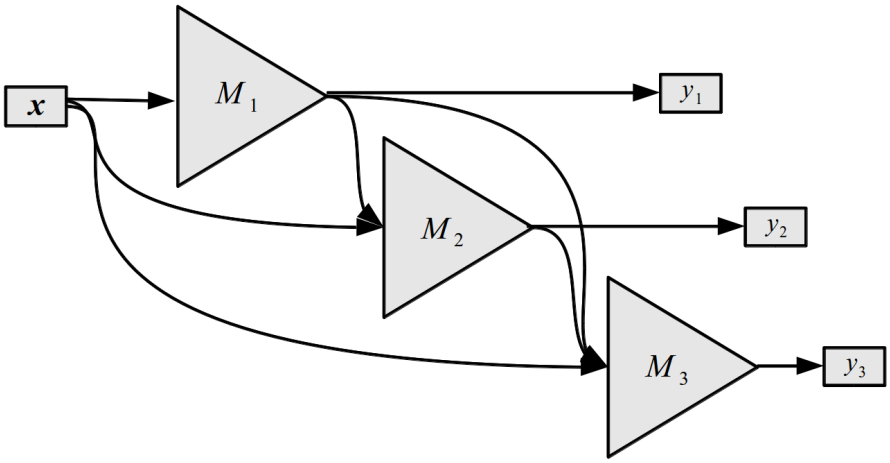

Figure 1: The structure of a chain classifier with 3 outputs. Each sub-model M_1 through M_3 is a model which takes as input the original input features as well as all predictions from previous models in the chain (or when training it can use ground truth output values from the training data). Each model predicts one of the elements of the output vector.

3.1 Predicting

A chain classifier makes a prediction with each model in $\mathbf{M}$ in turn, starting with M_1 and ending with M_d. Each model receives as input the predictions of all previous models, as well as the original input vector. This is shown in Algorithm 1. Note that the sub-routine `predictML(`M_i, $\boldsymbol{x}$`)` gives $\mathbf{x}$ as input to the model M_i and returns the resulting prediction,

which is a categorical value.

```
Function predict(M, x)
    input  : M is a d-dimensional vector of models created by Algorithm 2.
    input  : x is a 1 × m vector of input features.
    p ← a length d vector of categorical values.
    for i ← 1 to d do
        p_i ← predictML(M_i, x)
        x ← concatenate(x, p_i)
    end
    return p
```

Algorithm 1: Given an input vector, this returns the chain classifier's prediction of the output vector.

When implementing a chain classifier one design decision is to decide the order in which to predict outputs, namely the chain order. In theory, if exact inference were used on a chain classifier[3], and each sub-model produced no errors, chain order would not matter. However, exact inference is often intractable and sub-models may produce errors, so order can affect results. For example, if an erroneous output is predicted near the beginning of the chain, it will be a noisy input to models downstream and so will increase error. Many solutions have been proposed for reducing errors caused by chain order (1, 5, 6, 9–13). We discuss our solutions to chain order in Sections 4 and 5.1.

3.2 Training

Each model in $\mathbf{M}$ is trained individually in order from M_1 to M_d. Before each is trained, the inputs to the next model are augmented with the correct values from previous outputs.

The algorithm for training is shown in Algorithm 2. The sub-routine `trainML(`X, $Y_{:,i}$`)` trains a model with the inputs X and target outputs $Y_{:,i}$. Note that the notation $Y_{:,i}$ selects the i'th column (attribute) of the output matrix Y. The sub-routine `addColumn(`X, $Y_{:,i}$`)` appends the column $Y_{:,i}$ to the right hand side of matrix X.

[3]Exact inference means exploring every possible output vector while predicting. It is equivalent to using RANKED-CC (described in Section 5.1) with an unlimited beam width.

```
Function train(X, Y)
    input  : X is a |T| × m matrix of input instances, where m is the number of
             inputs per instance.
    input  : Y is a |T| × d matrix of outputs, where d is the number of outputs per
             instance.
    M ← a 1 × d vector of models.
    for i ← 1 to d do
        M_i ← trainML(X, Y_{·,i})
        X ← addColumn(X, Y_{·,i})
    end

    return M
```

Algorithm 2: The algorithm for training a chain classifier. The agorithm returns a vector **M** to be used for prediction.

When training a chain classifier, it is possible to use predictions from the previous models instead of label values from T as input to each model in the chain. However, we find that during training, giving each model the true labels from T gives very similar or better results, so we choose to use the label values[4] as inputs at training time as shown in Algorithm 2. This is the method used by Read et al. [5].

4 Weighted Output Vector Ensemble

The ensemble technique from Read et al. [5] used in ECC (and adapted to MDC by 3) is not satisfactory for MOD tasks because it does not preserve the integrity of the output vectors predicted by each sub-model and it does not produce a ranked list of predictions. The ensemble technique used by Read et al. [6] does preserve the output vectors predicted by each sub-model, but does not combine predictions into a ranked list.

We present weighted output vector ensemble (WOVE). It is a simple technique, but it maintains the integrity of the predicted output vectors and combines them into a ranked list with scores. For sub-models, WOVE can use any MOD models (such as chain classifiers)

[4]By using values from T it is also possible to train each sub-model separately, so they can be trained in parallel.

which predict a list of output vectors, each having an assigned score. The score can be thought of as the model's confidence in a predicted output vector. WOVE produces a scored list of predictions for a given input by combining the scored lists of its sub-models as follows: the score of each WOVE prediction is simply the sum of the scores assigned to that prediction by sub-models. More formally,

$$score_{\text{WOVE}}(\hat{\mathbf{y}}) = \sum_{i=1}^{|\mathbf{h}|} score_{model}(\hat{\mathbf{y}}, h_i)$$

where $\hat{\mathbf{y}}$ is an output vector to be scored, $\mathbf{h}$ is a vector of MOD models which each produce a ranked list of predictions with scores, and $score_{model}(\hat{\mathbf{y}}, h_i)$ returns the score that model i gives to prediction $\hat{\mathbf{y}}$, or zero if $\hat{\mathbf{y}}$ is not in the ranked list predicted by h_i. It is important to note that $\mathbf{h}$ is not the same as $\mathbf{M}$ from Section 3. If chain classifiers are used, then $\mathbf{h}$ is a vector of chain classifiers whereas $\mathbf{M}$ is the base classifiers within a single chain classifier. WOVE produces a ranked list of predictions by sorting these summed scores. For more intuitive scores, one could use averages instead of sums without changing the rank of predictions.

For sub-models WOVE uses chain classifiers modified as described in Section 5.1. Each chain classifier in WOVE is given a random chain order to collectively reduce errors from chain order. Although WOVE can work with any of the models in Section 5, we choose to only use chain classifiers because chain order randomization gives more diversity within the ensemble.

5 Model Adaptations For Predicting Multiple Output Vectors

We focus on problems that have multiple correct output vectors, so ideally solutions to such problems will not give just one predicted output vector, but a ranked list of predictions. This section discusses novel adaptations of existing methods to allow them to predict a list

of output vectors, each with a score, rather than just a single output vector.

5.1 Chain Classifiers

We create a modified chain classifier named RANKED-CC, which predicts a ranked list of output vectors with scores by using a beam search for prediction instead of the greedy search used by Read et al. [5]. This beam search also reduces errors introduced by chain order. The beam search is similar to that used by Kumar et al. [11] but adapted to categorical outputs greater than binary. One novel contribution of this work is to use a beam search to predict multiple output vectors.

As described in Section 3, training a chain classifier produces models $M_1 ... M_d$. Given an input $\mathbf{x}$, RANKED-CC first passes $\mathbf{x}$ to M_1, but instead of having M_1 predict a single categorical value, it instead predicts a score for each categorical value (it predicts a distribution over output values). It then passes each $y_1 \in \mathcal{Y}_1$ (as a categorical value) in turn to M_2. This process continues until all outputs have been predicted. To give a score to every possible output vector for a given input, the total predictions made by M_1 would be $|\mathcal{Y}_1|$. The total predictions made by M_2 would be $|\mathcal{Y}_1| \times |\mathcal{Y}_2|$. The total predictions made by M_d would be $|\mathcal{Y}_1| \times |\mathcal{Y}_2| \times ... \times |\mathcal{Y}_d|$. This number grows exponentially in d, so we implement a beam search to make predictions for problems with large d tractable. Each output vector from RANKED-CC can be thought of as a path through a state space, where a state along the path is the output values predicted thus far to be passed to the next model in $\mathbf{M}$. Each path is scored by the product of the scores (such as MLP activations) corresponding to the output values along that path. Consequently the path with the highest score is the one in which the models showed the most confidence, up to any pruning errors in the beam search. If the base classifiers give probabilities, then the score of an output vector is the probability that it is correct for the given input.

5.2 Monolithic Tag Technique

The monolithic tag technique (MONO) used by McClanahan et al. [8] is a good baseline method against which to compare our methods. It transforms an MDC problem into multiclass classification by transforming every distinct output vector in the training set to a categorical value. When predicting, a single categorical value is predicted which is easily transformed into the output vector it represents. The monolithic tag technique is easy to adapt for predicting multiple output vectors because multiclass classifiers can typically predict a distribution over output values. We simply use those scores (or probabilities) as scores for the corresponding output vectors to create a ranked list with scores.

5.3 Independent Classifiers

Independent classifiers (IC) is a baseline that predicts each output using a separate independent classifier. Following the example of Read et al. [5], we refer to multiclass classifiers used inside other models as *base classifiers*. In IC, each base classifier is ignorant of predictions made by other base classifiers. Each base classifier in IC gives a distribution of scores over output values, indicating confidence in each value. We require that these scores be in the range $[0, 1]$. To give a score to an entire output vector, we simply multiply the scores from each model. More precisely, let $P_i(y_j|\mathbf{x})$ be the score for the jth output value given by M_i (the model which predicts the ith dimension of the output) for an input vector $\mathbf{x}$, where $1 \leq i \leq d$ and $1 \leq j \leq |Y_i|$. If the base classifier is probabilistic, then this is a probability. The score IC gives to an entire output vector $\mathbf{y}$ is $P(\mathbf{y}|\mathbf{x}) = \Pi_{i=1}^{d} P_i(y_i|\mathbf{x})$.

It is easy to find the output vector to which IC gives the highest score. For every output dimension, simply choose the value to which the corresponding classifier gives the highest score. Finding a ranked list of output vectors with scores is less trivial. One way is to exhaustively iterate over every possible output vector and find the score of each. However, there are $\Pi_{i=1}^{d}|Y_i|$ possible output vectors, so that would only be tractable when d is small. In our experiments, we only care about the n output vectors with the highest scores. To find

those efficiently we use a greedy best-first search starting from the highest scoring output vector, and searching for the next highest scoring output vector which has not yet been explored. At each step of the search, the independent way in which IC calculates scores allows us to consider only candidate output vectors which differ from an explored vector in a single dimension. The upper bound on the run time is $O(dvn^2log(n))$, where v is the number of output values per output (assuming all outputs have the same number of values). This search is much faster than exhaustive search when $n \ll \Pi_{i=1}^{d}|Y_i|$. It is guaranteed to find the top n output vectors according to the scores given by IC. We omit the proof.

5.4 Hierarchical Multi-Output Nearest Neighbor

HMONN, as presented by Morris and Martinez [4], uses two layers of sub models. The first predicts an output vector using independent classifiers, the same as IC. The second layer is a k-nearest neighbor (KNN) model. It finds the k instances from the training set which are closest (using Euclidean distance) to the original input combined with the prediction from the first layer. We use the k output vectors from these k instances to create a list of predictions with scores. The score of each output vector is the number of times it occurs in the list. Duplicates are removed, so the number of predicted output vectors may be less than k.

6 Experiments

This section describes our experiment setup and gives results on synthetic and real-world data sets. It compares WOVE against other solutions for predicting both a single output vector, and for predicting multiple output vectors. It also empirically validates our model adaptations from section 5. Source code for our solutions is made available in SMODeLKit[5], an open-source toolkit.

[5]See https://github.com/jeheydorn/smodelkit

6.1 Data Sets

We show results on synthetic and real-world data sets. We generate 18 synthetic data sets using varied parameters. Algorithm 3 shows how we generate synthetic data. It creates a mapping function f which maps input vectors to distributions over output vectors. It then generates instances by sampling from those distributions. Finally, it adds noise to the generated instances. Noise can be seen as sensor or labeling error, or it can be seen as valid values which are not as likely as the input and output values before random variation is added.

The sub-routines in Algorithm 3 are defined as follows:

- `append`(l,e) $\equiv$ Returns a new list in which e is appended to the end of list l.
- `genCatVector`(d,VPO) $\equiv$ Returns a vector of categorical values of length d in which each element is randomly generated from a uniform distribution of VPO values. VPO (values per output) is a positive integer.
- `genGaussianSamples`(m) $\equiv$ Returns a vector of m samples from a standard normal distribution.
- `sampleUniform`(l) $\equiv$ Returns a sample generated from a uniform distribution over all values in list l.
- `sampleWithoutReplacement`(l,n) $\equiv$ Returns a set of n samples from list l, sampled uniformly without replacement.
- `normalize`(v) $\equiv$ Returns a vector in which each element is the corresponding element in v divided by the sum of all elements in v.
- `createCatDist`(s,w) $\equiv$ Returns a categorical distribution over the elements in s using elements in w as probabilities.
- `sampleCategorical`(c) $\equiv$ Returns a sample from categorical distribution c.

Function `generateSyntheticDataSet`(m,d,z,p,q,N,r,VPI,VPO)

input : m is the number of desired input dimensions.
input : d is the number of desired output dimensions.
input : z is the number of desired valid output vectors before adding noise.
input : p is the number of desired valid input vectors before adding noise.
input : q is the maximum number of output vectors for single input vector.
input : N is the number of instances to generate.
input : r is a binary value: *true* means inputs are real, $false$ means categorical.
input : VPI is the number of categorical values per input dimension if $r = true$.
input : VPO is the number of categorical values per output dimension.

$inputVectors \leftarrow$ an empty list
for $i \leftarrow 1$ **to** p **do**
  if r **then**
    $inputVectors \leftarrow$ `append` ($inputVectors$, `genGaussianSamples` (m))
  else
    $inputVectors \leftarrow$ `append` ($inputVectors$, `genCatVector` (m, VPI))
  end
end
$outputVectors \leftarrow$ an empty list
for $i \leftarrow 1$ **to** z **do**
  $outputVectors \leftarrow$ `append` ($outputVectors$, `genCatVector` (d, VPO))
end
$f \leftarrow$ a function, implemented as a dictionary, which maps input vectors to categorical distributions over sets of output vectors. It is defined as follows:
for $input \in inputVectors$ **do**
  $numVectors \leftarrow$ `sampleUniform` ($1..q$)
  $mappedOutputVectors \leftarrow$ `sampleWithoutReplacement` ($outputVectors$, $numVectors$)
  $weights \leftarrow$ `normalize` (`genGaussianSamples` ($|mappedOutputVectors|$))
  $f(input) \leftarrow$ `createCatDist` ($mappedOutputVectors$, $weights$)
end
`// Generate the instances:`
$T \leftarrow$ an empty list
for $i \leftarrow 1$ **to** N **do**
  $input \leftarrow$ `sampleUniform` ($inputVectors$)
  $outputVec \leftarrow$ `sampleCategorical` (f($input$))
  $T \leftarrow$ `append` (T, `createInstance` (`addNoise` ($input$, r), `addNoise` ($outputVec$, r)))
end
return T

Algorithm 3: The algorithm used for generating synthetic data.

Parameter	Value
m	3
d	2, 4, 8, 16
z	20
p	20
q	3
N	2000
r	true, false
VPI	4
VPO	2, 4, 8, 16

Table 1: Values of parameters passed to Algorithm 3 to create synthetic data sets. Descriptions of these parameters are given in Algorithm 3.

- `createInstance(`*input,output*`)` $\equiv$ Returns an instance in which *input* is labeled with *output*.
- `addNoise(`v, r`)` $\equiv$ Adds noise to each element of v. For each element of e in v, if r is false (meaning e is categorical), then with probability 0.1, it is incremented or decremented to the next or previous categorical value. If r is true (meaning e is continuous), then a sample from a normal distribution with mean 0 and standard deviation 0.1 is added to it.

In Algorithm 3 we limit the number of generated input and output vectors (before adding noise) to p and z respectively because we assume that in a real-world data set there are some input and output values that will not appear together no matter how many samples that data set contains. This creates unconditional dependence among outputs. We create conditional dependence among outputs by requiring that each input vector can map to only a subset of these z output vectors, which will typically be much fewer than z.

When adding noise, we choose to increment or decrement values rather than select a value from a uniform distribution of all possible values because we believe that in real-world problems, some values are more likely to be seen due to noise than others.

Table 1 gives the parameters used to created 18 synthetic data sets. We vary the

Name	Instances	m	d	VPO	UOV
real_2d4	2000	3	2	4	16
real_2d8	2000	3	2	8	51
real_4d2-16	2000	3	4	2-16	194
real_4d2	2000	3	4	2	16
real_4d4	2000	3	4	4	149
real_4d8	2000	3	4	8	230
real_8d2	2000	3	8	2	215
real_8d4	2000	3	8	4	643
real_8d8	2000	3	8	8	560
cat_2d4	2000	3	2	4	16
cat_2d8	2000	3	2	8	52
cat_4d2-16	2000	3	4	2-16	186
cat_4d2	2000	3	4	2	16
cat_4d4	2000	3	4	4	164
cat_4d8	2000	3	4	8	243
cat_8d2	2000	3	8	2	220
cat_8d4	2000	3	8	4	575
cat_8d8	2000	3	8	8	572

Table 2: Synthetic data sets. The first 9 data sets have continuous inputs, and the latter have categorical inputs. Column m is the number of inputs, d is the number of outputs, VPO is the number of values per output, and UOV is the number of distinct output vectors. UOV is not a parameter in data set creation; it is a consequence of it.

number of output dimensions d from 2 to 8, and we vary VPO from 2 to 8. We skip the case where $d = 2$ and $VPO = 2$ because it would be too easy for the models to learn. We also generate 2 data sets for which VPO is different for each output dimension[6], with values of 2, 4, 8, and 16. Table 2 gives details about these data sets.

We use 9 real-world MDC data sets and 6 MLC data sets from the Meka and Mulan[7] toolkits[8], as well 2 versions of a proprietary data set from InsideSales, one with 2 outputs and one with 3 outputs. Table 3 shows details of these data sets. Several of the MDC data sets are for multi-variate regression. In those cases we discretize the data by applying equal-

[6]This is functionality which, for simplicity, is not given in Algorithm 3.

[7]See [14].

[8]These data sets (except the proprietary InsideSales data set) can be obtained from https://github.com/jeheydorn/smodelkit/tree/master/Datasets/mdc.

Name	Instances	m	d	VPO	UOV
atp1d	337	411	6	10	210
atp7d	296	411	6	10	124
bridges	*89	7	5	2-6	28
edm	154	16	2	3	5
flare1	323	10	3	2-4	14
flare2	1066	10	3	3-8	22
is_2out	32774	22	2	2-6	12
is_3out	32544	22	3	2-10	108
rf1	9125	64	8	10	2391
rf2	9125	576	8	10	2391
thyroid	9172	29	7	2-5	31
enron	1702	1001	53	2	753
genbase	662	1186	27	2	32
medical	978	1449	45	2	94
music	592	71	6	2	27
scene	2407	294	6	2	15
yeast	2417	103	14	2	198

Table 3: Real-world data sets. Column m is the number of inputs, d is the number of outputs, VPO is the number of values per output, and UOV is the number of distinct output vectors. InsideSales data sets are prefixed with "is". The last 6 data sets are for multi-label classification. *We remove instances with unknown output values from the bridges data set. It originally had 107 instances.

frequency binning with 10 bins to outputs with more than 10 real values. The data sets from InsideSales are for the task of deciding how to contact a sales contact, given information about them. A two-output data set is for predicting how to contact the person and how long to wait. A three-output data set also predicts who should contact the person.

It is difficult to know *a priori* if a data set has conditional dependence among outputs, so some of these data sets may not be MOD tasks. With the InsideSales data sets, the nature of the task being solved implies they are MOD. For example, for a given sales contact, it might make sense for the CEO of the company to make a phone call or for a sales representative to send an email, yet a combination of these solutions might not make sense. The synthetic data sets are MOD because we generate them to have conditional dependence. We include the other real-world data sets to give more variety in our experiments, but they are not guaranteed to have conditional dependence. For this reason we consider results on the synthetic and InsideSales data sets to be most important for our problem.

6.2 Evaluation Measure

To measure results with multiple output vectors, we use top-n accuracy on the test set D, in which a prediction is counted correct if it is one of the top n predictions from the model for a given value of n. If the model makes less than n predictions, we only consider the predictions made.

$$Top\text{-}n\ accuracy = \frac{\sum_{i=1}^{|D|} I(\exists_{j \leq n}(\hat{\mathbf{y}}_{ij} = \mathbf{y}_i))}{|D|}$$

where $\mathbf{y}_i$ is the target output vector for instance i, and $\hat{\mathbf{y}}_{ij}$ is the j'th prediction of the ranked list of predictions for instance i. $I(x)$ is the indicator function which returns 1 if x is true, or 0 if false.

For n we use values of 1 and 3. When $n = 1$, top-n accuracy is just exact match accuracy of the target output vector, which is useful for comparing against other work. When

$n = 3$, up to three output vectors are considered from each predicted list. This is useful to test each model's ability to correctly predict multiple output vectors. We do not use larger values of n because if n is too large, models can get a high score by guessing many output vectors without preferring those that are likely for the given inputs. Also, in a practical UI, the right answer should be presented near the top of the list.

One disadvantage of top-n is that a score of 100% may not be possible. Even with $n = 3$ there may be more than three valid output vectors to which an input can be mapped, and if all of those output vectors appear in the test set, then no deterministic model can predict them all correctly. Future work includes finding better evaluators. One improvement could be to include the model's confidence scores in the evaluation and list all high-confidence predictions. Another option would be to use the average (or mean reciprocal) rank the model gives to each target output vector.

6.3 Solutions to Compare

We compare WOVE against the four models discussed in Section 5 and three models implemented in the machine learning toolkit Meka[9]. Meka extends the Weka machine learning toolkit [15] to MDC. The three models are ECC from Read et al. [5], EM_2CC from Read et al. [6], and ESC from Read et al. [3]. We use the settings described in the associated publications with the exception that we control for the base classifiers used by using MLPs with every model. Meka does not support predicting ranked lists, so top-n evaluations for these three models are restricted to $n = 1$.

6.4 Experiment Setup

For HMONN we use $k = 7$, and $\theta = 0.5$ to match the settings used by Morris and Martinez [4]. For RANKED-CC (including those in WOVE), we set the beam width to 20 based on performance on synthetic data sets. When RANKED-CC is not an ensemble, the chain order

[9]Meka can be found at http://meka.sourceforge.net/.

is the order of the outputs in the data set. All ensembles have 10 members. We perform 5-fold cross validation for all experiments.

6.4.1 Base classifiers

All of the models in our experiments require one or more base classifiers. By *base classifiers* we mean multiclass classifiers used within the models for which we show experiments. For base classifiers we use MLPs (multi-layer perceptrons) trained using error backpropagation, and SVMs implemented in Weka with default parameters.

MLPs expect real-valued inputs, so to convert any categorical inputs to real values, we convert each categorical value to a vector of real values using a 1-of-n encoding as follows. For categorical input x_i (where $1 \leq i \leq m$), we compute its index into $\mathcal{X}_i$ and convert this index into a vector of real values with length $|\mathcal{X}_i|$, such that every element is zero except position x_1, which is 1.

Each MLP M_i (where $1 \leq i \leq d$) outputs a vector of real values, one for each element in $\mathcal{Y}_i$. Each value corresponds to the MLP's confidence that the categorical value corresponding to that output is correct. To convert this vector to a categorical value, a greedy approach is to choose the index of the highest output. Several methods in Section 5 consider non-greedy approaches to converting to categorical values.

We train each MLP until there is no more decrease in sum squared error on a validation set for 10 epochs, or until 500 epochs are reached, whichever comes first. The validation set for each MLP is made of 20% of the instances of the MLP's training set, chosen at random. To help avoid over-training, every time an MLP shows an improvement on the validation set it stores its weights. When it finishes training it uses those weights which performed best on the validation set. The learning rate is 0.1. Each MLP has a single layer of hidden nodes containing $min(2m, 100)$ nodes, where m is the number of input features. We limit the number of hidden nodes to speed up MLP learning on data sets with many inputs. Each categorical attribute counts as only 1 input. Error backpropagation is done using online

updates. All real-valued inputs to MLPs are normalized by scaling to be within the range [-1, 1].

6.5 Results

Results on synthetic data sets are shown in Tables 4 and 5. Results on real-world data sets are shown in Tables 6 and 7. The highest results in each row are in bold. Results which are significant for $p = 0.1$ according the the Wilcoxon test are given in the caption of each table. The Wilcoxon test does a pairwise comparison of 2 models in isolation of all others. We choose to do pairwise tests because tests which take into account the number of models, such as the Nemenyi test discussed by Demšar [16], become weak when a large number of models are tested. We also show average ranks of every model at the bottom of each table.

On synthetic data sets WOVE has a statistically significant advantage over all other methods in pairwise comparisons using the Wilcoxon test. Notably it always out-performs RANKED-CC, which shows the advantage of the ensemble technique over a single chain classifier with only one chain order. The only synthetic data sets on which WOVE does not have the highest scores are 4 out of 9 categorical input data sets when $n = 1$, but even in these cases WOVE only loses by a small margin. This shows that WOVE performs well even with varying numbers of outputs and varying numbers of values per output. WOVE performs better than other methods for both $n = 1$ and $n = 3$, showing that it works well both at predicting the most likely output vector for an input and at predicting multiple output vectors.

On real-world data sets (Tables 6 and 7) WOVE has a statistically significant advantage over almost all other solutions in pairwise comparisons. The exception is that for $n = 1$ no significant difference is found between WOVE and HMONN, although WOVE does achieve a lower overall rank. For $n = 1$ it is worth noting that WOVE outperforms the other chain classifier ensemble techniques, ECC and PM$_s$CC. This shows the advantage of our method for combining chain classifiers in an ensemble along with using a beam search when predicting in RANKED-CC. For $n = 3$ WOVE is significantly better than all other methods tested. We

are unable to compare WOVE against ECC, PM$_s$CC, and ESC with $n = 3$ because they only predict one output vector, but results for $n = 1$ suggest that WOVE would do better even if these models were adapted to predict multiple output vectors.

If we compare results for $n = 1$ versus $n = 3$, we can see that scores consistently improve for those models that predict multiple output vectors. This shows that our model adaptations were successful. HMONN improves less than other models, so future work could include finding a better way to make multiple predictions with it, although results with $n = 1$ on synthetic and real-world data sets indicate that HMONN would still be unlikely to do better than WOVE.

On the two versions of the InsideSales data set and on synthetic data sets, WOVE achieves the highest score for both $n = 1$ and $n = 3$, meaning that it does best at the problem we are most interested in. On the flare1 and flare2, data sets, IC does well compared to other methods, indicating that those data sets do not have conditional dependence among outputs. These are likely not MOD data sets, so results on them are less interesting for our problem.

The publications for ECC, ESC, and PM$_s$CC all experiment with support vector machines (SVMs) with a linear kernel, as base classifiers. For comparison, we also ran our models with the same kind of SVMs and find several interesting conclusions. Results are in Tables 8, 9, 10, and 11. We find that on real-world data sets, with some data sets SVMs did better, and on others MLPs did better. This indicates that some data sets are more linearly separable than others. In those cases, SVMs are likely to do better because they learn a maximum margin between classes. For $n = 1$ and $n = 3$, the only method which was significantly better with SVMs than MLPs across all data sets was MONO. However, it is not significantly different from WOVE with SVMs or MLPs. It is worth noting also that when all models use SVMs WOVE is significantly better than IC and RANKED-CC. This shows that WOVE is still better than independent predictions and a single chain classifier even when all models use a new base classifier. On the InsideSales data sets, every method did better

with MLPs for both $n = 1$ and $n = 3$ except MONO with $n = 3$. The highest results on the InsideSales data sets overall were obtained by WOVE with MLPs, so results with MLPs are more interesting for our problem.

On synthetic data with $n = 1$, 6 out of 8 models are significantly better with MLPs than with SVMs. Only ESC and MONO show no significant change. For $n = 3$ all models are significantly better with MLPs, except MONO, which is significantly better with SVMs. One reason MONO is better with SVMs, and yet the chain classifier methods are worse with SVMs, is that our synthetic data sets require the model to learn higher order combinations in the inputs and outputs. MONO transforms the labels in such a way that combinations of outputs can still be learned by a linear model, but chain classifiers require the base classifier to learn such combinations. The SVMs with linear kernels used by Read et al. [5], Read et al. [6], and Read et al. [3] cannot do this because they learn a linear model. To verify this observation, we tested MONO and RANKED-CC with a polynomial kernel with exponent 3 and found MONO improves very little, but RANKED-CC improves much more. MONO is still better in most cases, but the difference is much smaller. This indicates that when outputs have higher order dependencies, chain classifiers need a base classifier that can learn those dependencies. For both values of n, WOVE with MLPs is significantly better than MONO with SVMs, so results with MLPs are again more interesting.

7 Other Related Work

This section discusses the following related work in order: multi-dimensional classification, multi-label classification, multi-task learning, and structured prediction. Among these works,

Data Set	ECC	ESC	HMONN	IC	MONO	PM$_s$CC	RANKED-CC	WOVE
real_2d4	0.594	0.573	0.562	0.549	0.562	0.593	0.596	**0.604**
real_2d8	0.539	0.532	0.492	0.484	0.527	0.533	0.529	**0.549**
real_4d2	0.546	0.543	0.51	0.477	0.547	0.555	0.54	**0.561**
real_4d2-16	0.439	0.4	0.393	0.344	0.334	0.455	0.462	**0.463**
real_4d4	0.535	0.449	0.489	0.459	0.338	0.538	0.494	**0.538**
real_4d8	0.376	0.328	0.348	0.259	0.211	0.384	0.378	**0.394**
real_8d2	0.275	0.089	0.187	0.19	0.05	0.336	0.297	**0.341**
real_8d4	0.236	0.079	0.206	0.129	0.056	0.259	0.258	**0.264**
real_8d8	0.276	0.109	0.271	0.119	0.067	0.299	0.236	**0.303**
cat_2d4	0.432	**0.452**	0.414	0.392	0.44	0.45	0.452	0.449
cat_2d8	0.441	0.447	0.438	0.38	0.435	0.453	0.448	**0.458**
cat_4d2	0.39	0.422	0.369	0.366	**0.428**	0.411	0.414	0.424
cat_4d2-16	0.341	0.36	0.348	0.283	0.339	0.368	**0.368**	0.368
cat_4d4	0.388	0.388	0.386	0.346	0.394	0.41	0.404	**0.411**
cat_4d8	0.383	0.383	0.382	0.328	0.306	0.383	0.386	**0.394**
cat_8d2	0.237	0.079	0.216	0.199	0.069	0.251	0.248	**0.261**
cat_8d4	0.227	0.07	0.233	0.166	0.07	0.241	**0.244**	0.239
cat_8d8	0.254	0.222	0.252	0.216	0.087	**0.262**	0.253	0.259
avg. rank	4.194	5.306	5.722	7.056	6.611	2.528	3.111	**1.472**

Table 4: Top-1 accuracy on synthetic data with MLPs as base classifiers. WOVE is significantly better than all other models in pairwise comparisons on the data sets tested.

Data Set	HMONN	IC	MONO	RANKED-CC	WOVE
real_2d4	0.568	0.707	0.721	0.773	**0.81**
real_2d8	0.527	0.617	0.718	0.744	**0.752**
real_4d2	0.51	0.73	0.711	0.746	**0.763**
real_4d2-16	0.42	0.427	0.5	0.604	**0.645**
real_4d4	0.511	0.582	0.466	0.635	**0.661**
real_4d8	0.4	0.334	0.377	0.576	**0.648**
real_8d2	0.213	0.309	0.132	0.439	**0.465**
real_8d4	0.236	0.182	0.136	0.39	**0.434**
real_8d8	0.302	0.171	0.164	0.397	**0.434**
cat_2d4	0.464	0.678	0.718	0.753	**0.785**
cat_2d8	0.497	0.557	0.689	0.693	**0.714**
cat_4d2	0.425	0.548	0.61	0.644	**0.662**
cat_4d2-16	0.372	0.356	0.511	0.578	**0.6**
cat_4d4	0.406	0.407	0.514	0.577	**0.593**
cat_4d8	0.402	0.389	0.463	0.574	**0.598**
cat_8d2	0.258	0.281	0.141	0.377	**0.387**
cat_8d4	0.242	0.191	0.161	0.387	**0.397**
cat_8d8	0.257	0.24	0.181	0.366	**0.384**
avg. rank	4.167	3.944	3.889	2	**1**

Table 5: Top-3 accuracy on synthetic data with MLPs as base classifiers. WOVE is significantly better than all other models in pairwise comparisons on the data sets tested.

Data Set	ECC	ESC	HMONN	IC	MONO	PM$_s$CC	RANKED-CC	WOVE
atp1d	0.11	0.113	0.146	0.057	0.045	0.113	0.107	**0.152**
atp7d	0.263	0.308	0.268	0.169	0.18	0.351	0.278	**0.359**
bridges	0.179	0.101	0.176	0.153	0.118	0.204	0.106	**0.212**
edm	0.48	0.532	0.533	0.48	0.527	0.474	0.54	**0.567**
flare1	0.82	0.82	0.819	0.816	0.816	**0.821**	0.819	0.819
flare2	0.811	0.811	**0.812**	0.811	0.809	0.811	0.811	0.811
is_2out	0.493	0.497	0.498	0.485	0.491	0.494	0.488	**0.506**
is_3out	0.34	0.364	0.348	0.324	0.359	0.362	0.365	**0.376**
rf1	0.329	0.181	**0.392**	0.28	0.114	0.358	0.302	0.382
rf2	0.16	0.078	**0.286**	0.103	0.017	0.157	0.126	0.194
thyroid	0.828	0.846	0.835	0.818	0.838	0.842	0.829	**0.846**
enron	0.144	0.116	0.111	0.109	0.135	0.12	0.137	**0.145**
genbase	0.878	0.885	**0.926**	0.839	0.885	0.8	0.791	0.894
medical	0.507	0.417	**0.603**	0.452	0.325	0.521	0.567	0.545
music	0.311	0.343	0.325	0.229	0.308	0.329	0.29	**0.358**
scene	0.672	0.697	0.697	0.592	0.696	0.693	0.654	**0.715**
yeast	0.201	0.23	0.239	0.135	0.205	0.23	0.23	**0.246**
avg. rank	5	4.265	3.118	6.882	6.147	4	4.941	**1.647**

Table 6: Top-1 accuracy on real-world data with MLPs as base classifiers. WOVE is significantly better than all other models in pairwise comparisons with the exception that no significant difference is found between WOVE and HMONN.

Data Set	HMONN	IC	MONO	RANKED-CC	WOVE
atp1d	0.254	0.099	0.066	0.203	**0.275**
atp7d	**0.522**	0.251	0.254	0.39	0.505
bridges	0.4	0.318	0.224	0.318	**0.482**
edm	0.72	0.92	0.96	0.947	**1**
flare1	0.838	0.909	0.922	0.916	**0.925**
flare2	0.825	**0.935**	0.934	0.933	0.933
is_2out	0.632	0.784	0.804	0.804	**0.817**
is_3out	0.462	0.501	0.542	0.55	**0.565**
rf1	0.533	0.457	0.163	0.485	**0.616**
rf2	**0.481**	0.176	0.025	0.214	0.326
thyroid	0.845	0.954	0.952	0.956	**0.972**
enron	0.167	0.176	0.2	0.209	**0.225**
genbase	**0.971**	0.938	0.926	0.933	0.95
medical	0.741	0.72	0.537	0.743	**0.788**
music	0.522	0.48	0.571	0.564	**0.641**
scene	0.883	0.819	0.908	0.896	**0.917**
yeast	0.349	0.21	0.342	0.375	**0.399**
avg. rank	3.412	3.912	3.529	2.824	**1.324**

Table 7: Top-3 accuracy on real-world data with MLPs as base classifiers. WOVE is significantly better than all other models in pairwise comparisons on the data sets tested.

Data Set	ECC	ESC	HMONN	IC	MONO	PM*s*CC	RANKED-CC	WOVE
cat_2d4	0.404	0.439	0.406	0.394	**0.453**	0.381	0.434	0.389
cat_2d8	0.423	0.433	0.414	0.374	**0.449**	0.432	0.414	0.437
cat_4d2	0.276	0.423	0.289	0.275	**0.425**	0.266	0.261	0.278
cat_4d2-16	0.325	0.353	0.322	0.268	**0.366**	0.358	0.342	0.355
cat_4d4	0.345	0.396	0.364	0.269	**0.406**	0.363	0.338	0.348
cat_4d8	0.342	0.393	0.376	0.307	**0.4**	0.373	0.347	0.369
cat_8d2	0.12	0.253	0.118	0.083	**0.257**	0.142	0.151	0.147
cat_8d4	0.162	0.229	0.189	0.096	**0.235**	0.201	0.212	0.202
cat_8d8	0.25	0.244	0.251	0.172	0.252	0.252	**0.257**	0.255
real_2d4	0.299	0.271	0.213	0.214	**0.328**	0.31	0.305	0.318
real_2d8	0.362	0.39	0.236	0.243	**0.408**	0.375	0.379	0.37
real_4d2	0.237	0.327	0.233	0.231	**0.356**	0.273	0.24	0.25
real_4d2-16	0.215	0.287	0.203	0.164	0.309	**0.32**	0.24	0.301
real_4d4	0.224	0.264	0.158	0.093	**0.387**	0.204	0.203	0.216
real_4d8	0.141	0.296	0.148	0.033	**0.311**	0.202	0.202	0.193
real_8d2	0.057	0.173	0.039	0.038	**0.192**	0.06	0.046	0.064
real_8d4	0.066	0.129	0.069	0.002	**0.155**	0.126	0.112	0.113
real_8d8	0.079	0.16	0.079	0	**0.182**	0.129	0.079	0.144
avg. rank	5.75	2.778	5.889	7.667	**1.194**	4.056	4.833	3.833

Table 8: Top-1 accuracy on synthetic data with SVMs as base classifiers. Wilcoxon significance for WOVE: WOVE $\succ$ ECC; ESC $\succ$ WOVE; WOVE $\succ$ HMONN; WOVE $\succ$ IC; MONO $\succ$ WOVE; WOVE $\succ$ RANKED-CC.

Data Set	HMONN	IC	MONO	RANKED-CC	WOVE
cat_2d4	0.443	0.615	**0.763**	0.699	0.698
cat_2d8	0.475	0.535	**0.724**	0.669	0.695
cat_4d2	0.412	0.437	**0.631**	0.475	0.489
cat_4d2-16	0.354	0.301	**0.598**	0.519	0.569
cat_4d4	0.386	0.335	**0.594**	0.487	0.49
cat_4d8	0.411	0.36	**0.591**	0.545	0.567
cat_8d2	0.197	0.12	**0.376**	0.239	0.235
cat_8d4	0.218	0.118	**0.377**	0.316	0.327
cat_8d8	0.266	0.203	**0.378**	0.37	0.372
real_2d4	0.237	0.484	**0.66**	0.629	0.613
real_2d8	0.249	0.363	**0.667**	0.632	0.652
real_4d2	0.238	0.456	**0.603**	0.494	0.471
real_4d2-16	0.227	0.2	**0.499**	0.403	0.477
real_4d4	0.167	0.132	**0.57**	0.37	0.382
real_4d8	0.204	0.052	**0.457**	0.387	0.377
real_8d2	0.042	0.08	**0.362**	0.137	0.141
real_8d4	0.085	0.004	**0.298**	0.195	0.209
real_8d8	0.087	0	**0.296**	0.254	0.271
avg. rank	4.389	4.611	**1**	2.722	2.278

Table 9: Top-3 accuracy on synthetic data with SVMs as base classifiers. Wilcoxon significance for WOVE: WOVE $\succ$ HMONN; WOVE $\succ$ IC; MONO $\succ$ WOVE; WOVE $\succ$ RANKED-CC.

Data Set	ECC	ESC	HMONN	IC	MONO	PMsCC	RANKED-CC	WOVE
atp1d	0.163	0.154	0.119	0.14	0.146	0.208	0.179	**0.23**
atp7d	0.409	0.341	0.264	0.336	0.417	0.429	0.39	**0.434**
bridges	**0.26**	0.225	0.2	0.224	0.165	0.248	0.247	0.235
edm	0.526	0.565	0.487	0.44	0.56	**0.617**	0.587	0.573
flare1	0.818	0.808	0.819	0.812	**0.822**	0.786	0.778	0.794
flare2	0.809	0.811	**0.812**	0.811	0.808	0.802	0.799	0.8
is_2out	0.465	0.474	**0.477**	0.457	0.474	0.461	0.462	0.461
is_3out	0.312	0.352	0.333	0.312	**0.355**	0.313	0.317	0.316
rf1	0.142	**0.188**	0.164	0.065	0.184	0.167	0.145	0.179
rf2	0.34	-	0.339	0.288	-	0.366	0.349	**0.373**
thyroid	0.785	0.786	0.779	0.782	0.786	**0.845**	0.839	0.841
enron	0.138	0.147	0.109	0.11	**0.162**	0.096	0.088	0.112
genbase	0.971	0.967	0.944	**0.977**	0.971	0.908	0.909	0.903
medical	0.704	0.681	0.642	0.668	**0.711**	0.621	0.626	0.641
music	0.338	0.345	0.339	0.268	0.334	**0.36**	0.331	0.336
scene	0.646	0.693	0.696	0.529	**0.697**	0.674	0.644	0.684
yeast	0.203	0.247	0.247	0.148	**0.252**	0.223	0.219	0.236
avg. rank	4.412	3.618	4.647	6.176	**3.382**	4.235	5.353	4.176

Table 10: Top-1 accuracy on real-world data with SVMs as base classifiers. Entries marked with a '-' failed to complete with 15 GB of RAM, and are treated as zeros. Wilcoxon significance for WOVE: WOVE $\succ$ IC; WOVE $\succ$ RANKED-CC.

Data Set	HMONN	IC	MONO	RANKED-CC	WOVE
atp1d	0.266	0.188	0.31	0.284	**0.361**
atp7d	0.512	0.427	0.593	0.569	**0.641**
bridges	**0.494**	0.318	0.388	0.4	0.376
edm	0.693	0.953	**1**	0.973	0.98
flare1	0.831	**0.906**	0.9	0.9	0.9
flare2	0.825	0.917	**0.919**	0.909	0.915
is_2out	0.618	0.753	**0.805**	0.784	0.786
is_3out	0.453	0.489	**0.552**	0.437	0.434
rf1	0.242	0.096	0.307	0.264	**0.316**
rf2	0.537	0.329	-	0.544	**0.584**
thyroid	0.784	0.836	0.956	0.969	**0.973**
enron	0.162	0.11	**0.254**	0.122	0.148
genbase	0.976	**0.982**	0.974	0.948	0.944
medical	0.783	0.675	**0.859**	0.77	0.798
music	0.561	0.32	0.644	0.592	**0.651**
scene	0.884	0.613	**0.936**	0.891	0.908
yeast	0.358	0.15	**0.421**	0.359	0.376
avg. rank	3.706	3.941	**1.941**	3.118	2.294

Table 11: Top-3 accuracy on real-world data with SVMs as base classifiers. Entries marked with a '-' failed to complete with 15 GB of RAM, and are treated as zeros. Wilcoxon significance for WOVE: WOVE $\succ$ HMONN; WOVE $\succ$ IC; WOVE $\succ$ RANKED-CC.

only those for MDC can be directly applied to MOD.

7.1 Multi-Dimensional Classification

MOD is a special case of MDC, so solutions in this sub-section are applicable to MOD problems, but these solutions only focus on predicting a single output vector. Solutions which we compare against are discussed in Section 2. We do not compare against methods in this section because they either do not show results on MDC data sets, do not report exact match accuracy, use proprietary software, only show results on synthetic data, or predict outputs independently of each other. MLC is a special case of MDC, so some methods in this section are applied to both MLC and MDC.

Bielza et al. [17] describe methods for building Bayesian networks to find the MAP (maximum a posteriori) hypothesis for each output vector, given an input. The resulting networks they create can capture dependence between outputs. Zaragoza et al. [1] use the marginal dependence of each output to create an undirected maximum weight spanning tree over the outputs. This spanning tree is an approximation of the dependency structure of the outputs. Then for each output they create a chain classifier where that output is the first sub-model, and the order of the others is determined by the spanning tree. They use these d classifiers as an ensemble when predicting. Although both of these works frame their problem as MDC, they show results only on MLC data sets.

Rodríguez and Lozano [18], de Waal and van der Gaag [19], and [20] also discuss ways to construct Bayesian networks for solving MDC problems. Rodríguez and Lozano [18] treat their task as multi-objective optimization, and report accuracy of each output independently. Qazi et al. [21] build a classifier which determines if a heart is normal or abnormal given readings from an ultrasound. Their method employs 16 class variables representing different regions of the heart. They construct a Bayesian network over the outputs using an algorithm for automatically learning network structure.

Theeramunkong and Lertnattee [22] experiment with transforming text classification tasks into multi-dimensional classification tasks. They transform multi-label tasks for text classification into multi-dimensional tasks by defining categories (dimensions) for the labels. This gives more training data per label than creating a new label for every set of labels in the data set. They also discuss the advantages of creating multiple dimensions over defining a hierarchy over the labels. They predict each dimension independently, so they do not take into account conditional dependence between dimensions.

7.2 Multi-label Classification

Works in the following sub-sections are related to MOD but are not directly applicable to such problems. MLC problems can be treated as MOD problems in which all outputs are binary. Many MLC solutions seek to learn conditional or unconditional dependence among outputs (9–13, 23–26).

7.3 Multivariate Regression

In multivariate regression, the output to predict is a vector of real values. Multivariate regression is related to MOD because there are multiple outputs, and there could exist dependencies between them. One solution for multivariate regression problems is a Gaussian process, which is a probabilistic model for regression. Rasmussen [27] defines a Gaussian process as a collection of random variables, any finite number of which have (consistent) joint Gaussian distributions. They have been adapted for multi-class classification [28, Ch. 3], sequence tagging [29], and multivariate regression [30].

7.4 Multi-task Learning

In multi-task learning, multiple tasks are learned simultaneously using a common representation to improve results on one of the tasks. The goal, however, does not include learning

dependence among the outputs of the two tasks. Caruana [31] presents a model for multi-task learning where each output is predicted using only the input features, without taking into account other outputs. Thus, this model does not model dependence between task outputs.

7.5 Structured Prediction

In structured prediction (SP), inputs and outputs can be structured objects. The structure of those objects depends on the specific task, and can include lists, trees, and graphs. MOD could be thought of as a variant of SP in which inputs and outputs are lists of fixed length and each output can assume a different range of values. However, current work in SP [32–37] has not yet been applied to MDC nor MOD. Finley and Joachims [38] apply SP to MLC, but not general MDC problems. Although relevant, we leave the application of SP methods to MOD to future work.

8 Conclusions and Future Work

We have presented a theorem and its proof that conditional dependence in multi-dimensional classification problems implies that a single input can map to multiple different output vectors. This means that solutions should predict multiple output vectors, and evaluation measures should take into account the possibility of multiple correct output vectors per input.

We have modified four existing models to predict multiple output vectors and shown improved results by doing so. Furthermore, we have created WOVE, a new ensemble technique for chain classifiers which shows significant improvements over other methods when evaluated using pairwise comparisons. It is able to combine multiple predictions from each of its sub-models in such a way that conditional dependence among those predictions is preserved.

In future work we plan to consider better ways to select chain orderings for ensemble members in WOVE. We also plan to look for better ways to evaluate predictions with multiple

output vectors.

A Proof of Theorem 1

Lemma 1. *Given continuous (or categorical) input random variable X and categorical output random variables Y_1 and Y_2, if*

$$P(Y_1, Y_2|\boldsymbol{x}) = \begin{cases} 1 & \text{if } \langle y_1, y_2 \rangle = \boldsymbol{y}^* \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

then

$$P(Y_1|\boldsymbol{x}) = \begin{cases} 1 & \text{if } y_1 = y_1^* \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

$$P(Y_2|\boldsymbol{x}) = \begin{cases} 1 & \text{if } y_2 = y_2^* \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

where y_1^* and y_2^* are the first and second elements of $\boldsymbol{y}^*$ respectively.

Proof: We prove this by contradiction. Assume that equation 2 is false. This means that $\exists y_1(P(y_1|\mathbf{x}) \neq 0 \wedge y_1 \neq y_1^*)$. Let that y_1 be named y_1'. By marginalization,

$$P(y_1'|\mathbf{x}) = \sum_{y_2} P(y_1', y_2|\mathbf{x}). \tag{4}$$

By the product rule and the definition of conditional probability,

$$P(y_1', Y_2|\mathbf{x}) = \frac{P(Y_2|y_1', \mathbf{x})P(y_1', \mathbf{x})}{P(\mathbf{x})}. \tag{5}$$

Because we know that $P(y_1'|\mathbf{x}) \neq 0$, we know by equation 4 that $\exists y_2(P(y_1', y_2|\mathbf{x}) \neq 0)$. By equation 5 we can see that if $\forall y_2(P(y_2|y_1', \mathbf{x}) = 0)$ then $\forall y_2(P(y_1\prime, y_2|\mathbf{x}) = 0)$, which

contradicts our conclusion from equation 4. Thus $\exists y_2(P(y_2|y_1', \mathbf{x}) \neq 0)$. Let this y_2 be named y_2'. Thus there exists a pair y_1', y_2', such that $P(y_1', y_2'|\mathbf{x}) \neq 0$, and $y_1' \neq y_1^*$. This contradicts equation 1. Thus our assumption that equation 2 is false is incorrect, and so equation 2 is correct. It follows trivially that equation 3 is also correct.

Theorem 1. *Given continuous (or categorical) input random variable X and categorical output random variables Y_1 and Y_2, if Y_1 is conditionally dependent on Y_2 given an observed $\boldsymbol{x}$, then the probability distribution $P(Y_1, Y_2|\boldsymbol{x})$ assigns non-zero probability to at least 2 distinct output vectors $\boldsymbol{y}$ and $\boldsymbol{y'}$.*

Proof: Assume Y_1 and Y_2 are conditionally dependent given $\mathbf{x}$. Next assume for some output vector $\mathbf{y}^*$ equation 1 (in lemma 1) is true. We show that this leads to a contradiction. Conditional dependence gives us $P(Y_1, Y_2|\mathbf{x}) \neq P(Y_1|\mathbf{x})P(Y_2|\mathbf{x})$. By our second assumption and lemma 1, equations 2 and 3 must be true.

By equations 2 and 3, however, it follows that

$$P(Y_1|\mathbf{x})P(Y_2|\mathbf{x}) = \begin{cases} 1 & \text{if } \langle y_1, y_2 \rangle = \mathbf{y}^* \\ 0 & \text{otherwise.} \end{cases} = P(Y_1, Y_2|\mathbf{x}).$$

This means Y_1 and Y_2 are conditionally independent, which contradicts our first assumption.

□

References

[1] Julio H Zaragoza, Luis Enrique Sucar, Eduardo F Morales, Concha Bielza, and Pedro Larranaga. Bayesian chain classifiers for multidimensional classification. In *IJCAI*, volume 11, pages 2192–2197. Citeseer, 2011.

[2] Krzysztof Dembszynski, Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier. On label dependence in multilabel classification. In *LastCFP: ICML Workshop on Learning from Multi-label Data*. Ghent University, KERMIT, Department of Applied Mathematics, Biometrics and Process Control, 2010.

[3] Jesse Read, Concha Bielza, and Pedro Larranaga. Multi-dimensional classification with super-classes. *Knowledge and Data Engineering, IEEE Transactions on*, 26(7):1720–1733, 2014.

[4] Richard Morris and Tony Martinez. A hierarchical multi-output nearest neighbor model for multi-output dependence learning. Master's thesis, Brigham Young University, Department of Computer Science, Brigham Young University, Provo, UT 84602, 2013.

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

[6] Jesse Read, Luca Martino, and David Luengo. Efficient monte carlo methods for multi-dimensional learning with classifier chains. *Pattern Recognition*, 47:1535–1546, 2014.

[7] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

[8] Peter McClanahan, George Busby, Robbie Haertel, Kristian Heal, Deryle Lonsdale, Kevin Seppi, and Eric Ringger. A probabilistic morphological analyzer for Syriac. In *2010 Conference on Empirical Methods in Natural Language Processing*, pages 810–820. Association for Computational Linguistics, 2010.

[9] Weiwei Cheng, Eyke Hüllermeier, and Krzysztof J Dembczyński. Bayes optimal multilabel classification via probabilistic classifier chains. In *27th International Conference on Machine Learning (ICML-10)*, pages 279–286, 2010.

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

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

[12] Nan Li and Zhi-Hua Zhou. Selective ensemble of classifier chains. In *Multiple Classifier Systems*, pages 146–156. Springer, 2013.

[13] Eduardo Corrêa Gonçalves, Alexandre Plastino, and Alex A Freitas. A genetic algorithm for optimizing the label ordering in multi-label classifier chains. In *2013 25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 469–476. IEEE Computer Society Conference Publishing Services (CPS), 2013.

[14] Grigorios Tsoumakas, Eleftherios Spyromitros-Xioufis, Jozef Vilcek, and Ioannis Vlahavas. Mulan: A Java library for multi-label learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011.

[15] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The Weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[16] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[17] Concha Bielza, Guangdi Li, and Pedro Larranaga. Multi-dimensional classification with Bayesian networks. *International Journal of Approximate Reasoning*, 52(6):705–727, 2011.

[18] Juan Diego Rodríguez and Jose Antonio Lozano. Multi-objective learning of multi-dimensional Bayesian classifiers. In *Hybrid Intelligent Systems, 2008. HIS'08. Eighth International Conference on*, pages 501–506. IEEE, 2008.

[19] Peter R de Waal and Linda C van der Gaag. Inference and learning in multi-dimensional Bayesian network classifiers. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pages 501–511. Springer, 2007.

[20] Linda C Van Der Gaag and Peter R De Waal. Multi-dimensional Bayesian network classifiers. In *Probabilistic Graphical Models*, pages 107–114, 2006.

[21] Maleeha Qazi, Glenn Fung, Sriram Krishnan, Romer Rosales, Harald Steck, R Bharat Rao, Don Poldermans, and Dhanalakshmi Chandrasekaran. Automated heart wall motion abnormality detection from ultrasound images using Bayesian networks. In *IJCAI*, volume 7, pages 519–525, 2007.

[22] Thanaruk Theeramunkong and Verayuth Lertnattee. Multi-dimensional text classification. In *19th International Conference on Computational Linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics, 2002.

[23] Shuiwang Ji, Lei Tang, Shipeng Yu, and Jieping Ye. Extracting shared subspace for multi-label classification. In *14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 381–389. ACM, 2008.

[24] Liang Sun, Shuiwang Ji, and Jieping Ye. Hypergraph spectral learning for multi-label classification. In *14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 668–676. ACM, 2008.

[25] Eneldo Loza Mencia and Johannes Fürnkranz. Efficient pairwise multilabel classification for large-scale problems in the legal domain. In *Machine Learning and Knowledge Discovery in Databases*, pages 50–65. Springer, 2008.

[26] Shantanu Godbole and Sunita Sarawagi. Discriminative methods for multi-labeled classification. In *Advances in Knowledge Discovery and Data Mining*, pages 22–30. Springer, 2004.

[27] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced Lectures on Machine Learning*, pages 63–71. Springer, 2004.

[28] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. ISBN 26218253X.

[29] Yasemin Altun, Thomas Hofmann, and Alexander J Smola. Gaussian process classification for segmenting and annotating sequences. In *21st International Conference on Machine Learning*, page 4. ACM, 2004.

[30] Shiliang Sun. Infinite mixtures of multivariate gaussian processes. In *Machine Learning and Cybernetics (ICMLC), 2013 International Conference On*, volume 3, pages 1011–1016. IEEE, 2013.

[31] Richard Caruana. Multitask learning: A knowledge-based source of inductive bias. In *10th International Conference on Machine Learning*, pages 41–48. Morgan Kaufmann, 1993.

[32] Hal Daumé III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *22nd International Conference on Machine Learning*, pages 169–176. ACM, 2005.

[33] Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. Learning structured prediction models: A large margin approach. In *22nd International Conference on Machine Learning*, pages 896–903. ACM, 2005.

[34] Aron Culotta and Andrew McCallum. Reducing labeling effort for structured prediction tasks. In *AAAI*, pages 746–751, 2005.

[35] Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine Learning*, 75(3):297–325, 2009.

[36] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *21st International Conference on Machine Learning*, page 104. ACM, 2004.

[37] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. In *Journal of Machine Learning Research*, pages 1453–1484, 2005.

[38] Thomas Finley and Thorsten Joachims. Training structural SVMs when exact inference is intractable. In *25th International Conference on Machine Learning*, pages 304–311. ACM, 2008.