2013-03-08

A Hierarchical Multi-Output Nearest Neighbor Model for Multi-Output Dependence Learning

Richard Glenn Morris
Brigham Young University - Provo

A Hierarchical Multi-Output Nearest Neighbor Model for Multi-Output Dependence Learning

Richard G. Morris

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
Christophe Giraud-Carrier
Dennis Ng

Department of Computer Science

Brigham Young University

March 2013

ABSTRACT

A Hierarchical Multi-Output Nearest Neighbor Model for Multi-Output Dependence Learning

Richard G. Morris
Department of Computer Science, BYU
Master of Science

Multi-Output Dependence (MOD) learning is a generalization of standard classification problems that allows for multiple outputs that are dependent on each other. A primary issue that arises in the context of MOD learning is that for any given input pattern there can be multiple correct output patterns. This changes the learning task from function approximation to relation approximation. Previous algorithms do not consider this problem, and thus cannot be readily applied to MOD problems. To perform MOD learning, we introduce the Hierarchical Multi-Output Nearest Neighbor model (HMONN) that employs a basic learning model for each output and a modified nearest neighbor approach to refine the initial results. This paper focuses on tasks with nominal features, although HMONN has the initial capacity for solving MOD problems with real-valued features. Results obtained using UCI repository, synthetic, and business application data sets show improved accuracy over a baseline that treats each output as independent of all the others, with HMONN showing improvement that is statistically significant in the majority of cases.

Keywords: Multi-Output Dependence, Machine Learning, KNN

ACKNOWLEDGMENTS

Thanks go out to my graduate committee for all of their help and support in finishing out this project.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Classification is a common problem in machine learning. Given an input vector $\vec{x}$ and a set of possible outputs C, classification is the task to give a mapping from the input $\vec{x}$ to some output y, where $y \in C$. We define a classifier as $f : X_1 \times X_2 \times \cdots \times X_m \mapsto C$ where m is the number of input features. However, some problems require multiple outputs given the same input vector. In these cases the input $\vec{x}$ maps to a vector of outputs $\vec{y}$ where $y_i \in C_i$ where C_i is the set of possible outputs for output i. The mapping $\vec{x} \mapsto \vec{y}$ has been done before in the case where the components of $\vec{y}$ are independent [10] and where $\vec{y}$ is structured [2]. Typically, it is unknown *a priori* if the outputs are independent, and in many cases the outputs are dependent. Thus, any one output may be considered correct or incorrect only when considered in the context of other outputs. In this sense, outputs should be seen as correct or incorrect collectively, and not individually.

These multiple output decisions are decisions that humans make daily. For example, when planning a meal the task is to give a mapping from meal times to meal contents. Orange juice is normally a good choice of drink for breakfast, but if the main or side dish contains chocolate, orange juice may be a poor choice. If pancakes are the main dish, scrambled eggs are a better choice for a side dish than if an omelet is the main dish.

We define a training data set T to be a set of input vectors $\vec{x}$ each labeled with an appropriate output $\vec{y}$. It is possible for there to be some input vector $\vec{x}$ associated with multiple instances in T, each labeled with a different output vector $\vec{y}$ where $|\vec{y}| \geq 1$. In this case, there are multiple correct outputs for $\vec{x}$. Each of these outputs do not necessarily need

to be equally good, some outputs may be more desirable than others, but there are still multiple outputs that would be acceptable given the input $\vec{x}$.

This changes the task from finding a mapping function $\vec{x} \mapsto \vec{y}$ to finding a relation from $\vec{x}$ to $\vec{y}$. We consider the relation where there are multiple outputs (where $|\vec{y}| > 1$) and there is a dependency between the outputs. This gives rise to interesting questions about which correct solutions to choose when there are multiple correct solutions available. Different output selection methods could include random selection, a predetermined bias or preference, a learned bias or preference, and weighted random selection. This is similar to what relaxation models accomplish, in that they can give a relation from $\hat{y}$ to $\vec{y}$ where $\hat{y}$ is an initial setting for the solution $\vec{y}$, but current models are auto-associative, unable to handle arbitrary input and output mappings, as opposed to the hetero-associative model, which we support.

Many current algorithms fail to directly support multiple outputs, and any current model will assume either independence or structure in the outputs. Current approaches either induce one model per output or create a single model that gives multiple outputs without explicitly modeling the dependencies. Different models will support multiple independent outputs with a varying degree of success, without further modification. Decision Tree learning algorithms must either induce multiple trees in order to produce multiple outputs or must induce a single tree that blows up exponentially, but neither of these approaches will model any dependence between the output variables. K-Nearest Neighbor algorithms can support multiple outputs with little change to the basic algorithm. Multi-Layer Perceptron (MLP) models can give multiple outputs with a single model or multiple model approach. However, none of these algorithms explicitly model dependent outputs. An example of the multiple model approach can be seen in Figure 1.1.

Structured Prediction (SP) is a single model approach to dependent output where the output has some structure [2]. The output structure could be a graph, a tree, a sequence, or any other structure. While SP algorithms support multiple dependent outputs, current SP

algorithms do not account for multiple correct outputs.

We introduce Multi-Output Dependence (MOD) learning as an algorithmic family that models dependencies between multiple outputs. The MOD classification task is to give a relation from $\vec{x}$ to C in the case where there exists an $i \neq j$ such that $y_i \in C_i$ depends on $y_j \in C_j$. MOD problems can be seen as those problems where the outputs are important in addition to the inputs when making a decision. The context of the other outputs along with the input needs to be considered when making a decision rather than the input alone. MOD learning requires approximating a relation, as opposed to the more traditional function approximation. This work is restricted to tasks with nominal features, although the proposed algorithm does show potential for tasks with real-valued features.

This paper introduces the Hierarchical Multi-Output Nearest Neighbor model (HMONN) in order to solve the MOD problem. This hierarchical model has two different layers. The first layer is a naïve approach with one learning model per output. The models that comprise the first layer can be any traditional machine learning model. The second layer is a modified nearest neighbor model that refines the predictions made on the first layer. An example of this system can be seen in Figure 1.2. Though HMONN focuses on tasks with nominal features, it also gives improvement for some tasks with real-valued features by implicitly modeling a similarity function for the feature space.

The rest of the paper is organized as follows. Chapter 2 gives an overview of related work. We present HMONN to address the MOD classification problem in Chapter 3. In Chapter 4 experimental results are presented where HMONN is compared to a baseline model that assumes independence between outputs. Results are obtained from comparisons using UCI data, synthetic data, and real-world data. Finally, Chapter 5 presents a summary and discussion of future work.

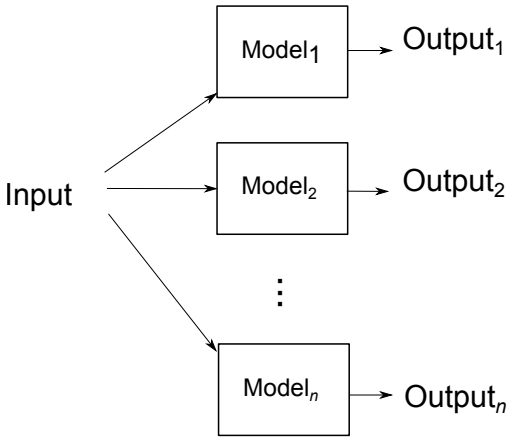


Figure 1.1: An example of a system where multiple models are used to give a multiple output prediction. Each separate model could use a different algorithm or they could all use the same algorithm. There is, however, no modeling of the dependence between the outputs.

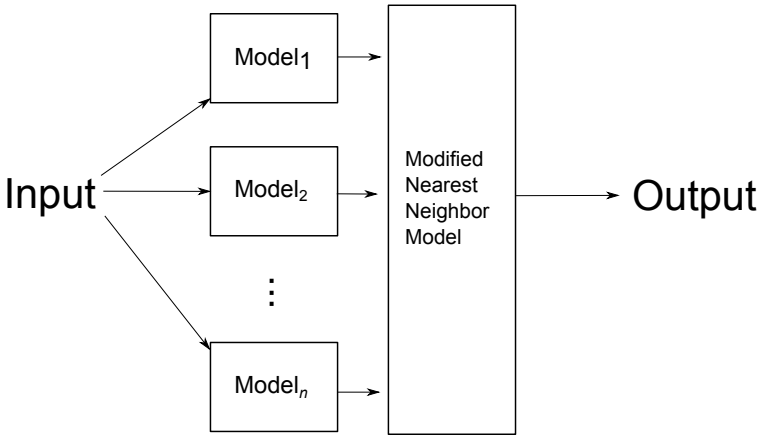


Figure 1.2: The HMONN model used to solve MOD problems. The models used as input to the modified nearest neighbor model can use any algorithm to produce an initial prediction. This models the dependence between the outputs in terms of the local context from the nearest neighbor algorithm.

Chapter 2

Related Work

MOD learning is an intersection of learning in the context of multiple outputs and in the context of statistical dependence. Multi-label classification, relaxation networks, transfer learning, multi-dimensional Bayesian network classifiers, and structured prediction all have multiple outputs. Recurrent neural networks, transfer learning, multi-dimensional Bayesian network classifiers, and structured prediction all present different angles on statistical dependence. MOD supports multiple outputs that are dependent on each other, with the added complexity that some inputs have multiple correct outputs, a problem that has not been considered by previous work. The unique nature of this dependence prevents previous algorithms from being applied to MOD problems.

Tsoumakas et al. [16] give an overview of multi-label classification (where $\forall i\ C_i = \{0, 1\}$). They define the two main solution concepts for multi-label classification. The first approach is problem transformation, where the given data is transformed into a single problem that already has a well defined solution. An example of problem transformation is binary relevance, where a binary classifier is trained for each possible label. The second approach is algorithm adaptation, where current algorithms are modified to solve the multi-label classification problem directly. Zhang and Zhou [17] use an algorithm similar to k-nearest neighbor to perform multi-label classification. This is an example of algorithm adaptation. Their algorithm first finds the k-nearest neighbors to an input vector, then finds the most likely label using a form of maximum *a posteriori* inference. Heath et al. [10] compare binary relevance to several different types of algorithm adaptation. They also give the

surprising result that binary relevance outperforms the algorithm adaptation methods that were implemented. The binary relevance approach is similar to the baseline used in our experiments.

Many problems have a structure to them that is missed by standard classification algorithms [14]. Structured Prediction (SP) seeks to solve this problem by modeling the dependence structure of the outputs. This structure could be a sequence, a tree, a graph, or an image. This allows for multi-output as well as output dependencies. However, these dependencies are almost always limited to Markovian dependencies — they are related by time or space.

Theoretically, SP algorithms are capable of modeling any problem with structure, and MOD problems would be an example of this kind of problem. The main difference between MOD and SP is that MOD problems are assumed to have some inputs with multiple correct outputs, whereas with current SP algorithms there is a single correct output assumed for each input.

Bakır et al.[2] give an overview of the state of the art in SP. Most of these state-of-the-art algorithms are based on a maximum margin approach such as the approach proposed by Taskar et al. [14]. This work by Taskar et al. is one of the first to combine the strength of kernels and maximum margin methods with the language and algorithms of graphical models. They compare this novel approach to standard Support Vector Machine and Conditional Random Field models.

Tsochantaridis et al. [15] build off of the same foundational concepts used by other maximum margin methods for SP. Their algorithm is based on Support Vector Machines. Their proposed method tends to produce quadratic programs with an exponential number of constraints. While this would normally make learning near impossible, they also present an algorithm that utilizes the specific nature of maximum margin problems to examine a much smaller subset of the constraints. This solution avoids an exponential running time for most problems considered to be important in SP.

Daumé et al. [5] present a slightly different algorithm for SP called SEARN. SEARN operates by deconstructing each decision into multiple binary decisions. Daumé et al. also give a theoretical guarantee that good performance on the binary decision problems translates into good performance on the larger problem. They demonstrate promising results when comparing SEARN to several standard algorithms used on various natural language processing tasks.

The Multi-dimensional Bayesian network classifier (MBC) [3] is an algorithm that has the graph of class and feature variables for a given problem (as seen in a standard Bayesian approach) divided into a class subgraph, a feature subgraph, and a bridge subgraph. It is a type of SP, where there is some assumed structure to the output. The algorithm computes the most probable explanation given those class and feature variables. Decomposing the graph into three subgraphs is meant to ameliorate the computation of the most probable explanation, which is known to be in NP for Bayesian networks. However, the structure of the three subgraphs must be defined in order to use MBCs. This is another example of a multi-output problem that exhibits some dependencies, but not necessarily between outputs. Our approach is the first approach to consider the multi-output problem to be associated with multiple correct outputs, and as such to treat the problem as relation approximation.

While MOD learning is relation approximation, this should not be confused with relational learning. Statistical Relational Learning [9, 12] and Multi-Relational Learning [6] both handle relational data, not relation approximation. These relational learning models learn a function from relational data and handle specially formatted and structured data.

A relaxation network is a type of network that is allowed to settle to an output pattern given some input pattern. This approach is the closest to the relation approximation that occurs with MOD learning. Hopfield networks [11] and Boltzmann machines [1] are two examples of relaxation networks. The behavior of these networks is very sensitive to the architecture of the network. They can easily have multiple outputs. They are, however, auto-associative, mapping a pattern of n inputs to a pattern of n outputs, or more specifically,

they map from an initial prediction $\hat{y}$ to a final prediction $\vec{y}$. These outputs can be dependent on each other anywhere there is a non-zero weight between nodes. Although a Boltzmann machine could support higher order dependencies through hidden nodes, current learning algorithms have difficulties learning these higher order dependencies. Another limitation of relaxation network models is finding an appropriate representation of the problem in the network structure to allow these models to give valid solutions. Finally, our approach does not require an initial $\hat{y}$ like the relaxation networks do, but it maps from an arbitrary $\vec{x}$ to $\vec{y}$.

Some distantly related problems include the following. Recurrent Neural Networks are a type of neural network that considers dependencies between different time steps rather than different outputs [7]. In transfer learning [13] and multitask learning [4] there are multiple tasks being learned at the same time. These multiple tasks may not share the same inputs, and are solving several separate, but related, problems. MOD learning is solving a single problem with highly dependent outputs, or subcomponents.

Chapter 3

HMONN

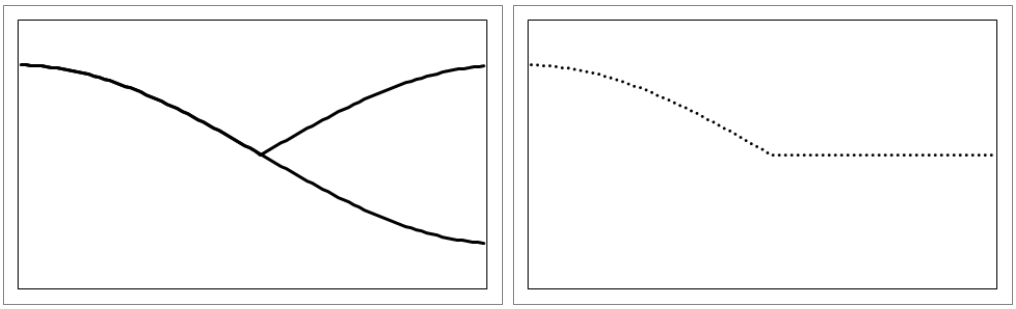
We present the Hierarchical Multi-Output Nearest Neighbor model (HMONN) to solve the MOD problem. This model is compared against a baseline that is an implementation of the naïve independence model.

In this work, we define an output prediction $\hat{y}$ to be correct for a given input vector $\vec{x}$ if there is some training instance in the training data T that has $\vec{x}$ labeled with output $\vec{y}$ and $\hat{y} = \vec{y}$ (or if $\hat{y} = \vec{y}$ for the current test instance). The traditional definition of a correct prediction only takes into account the labeling on the instance currently being tested. This definition allows the model to use information contained within the training data to determine which output predictions should be counted as correct.

Traditional models are not able to model the dependencies between outputs. This is, in part, due to the fact that traditional models are function approximators, and MOD problems are, at their heart, relations. For example, a traditional MLP-based solution will thrash between the multiple possible outputs, and may not give any of the possible correct output vectors. As an example of this, consider a training set that contains the following two training patterns.

x_1	x_2	x_3	x_4	y_1	y_2
1	0	0	1	1	0
1	0	0	1	0	1

An MLP will adjust weights towards outputting $\{1, 0\}$ whenever the first instance is encountered, and whenever the second instance is encountered it will adjust the weights



(a) Actual Relation (b) Possible Learned Function

Figure 3.1: A graphical example of a relation with multiple correct outputs. The solid curve represents the relation itself. The dotted curve represents the function learned by an MLP model. The dotted curve follows the solid curve exactly until the relation branches, at which point the dotted curve is in the center of the two branches.

towards outputting $\{0, 1\}$. The network will consequently adjust the weights towards the output $\{0.5, 0.5\}$ (without ever stabilizing), given the input $\{1, 0, 0, 1\}$, rather than towards either of the two possible correct outputs.

A graphical example of the problem faced by an algorithm trying to learn a problem with multiple correct outputs is shown in Figure 3.1. The solid curve represents the relation in the training data and the dotted curve represents the function that could be learned, for example by an MLP. The algorithm could output both branches of the relation, following the relation exactly, choose one of the branches arbitrarily, or choose one based on some criteria. It should not, however, output something completely different.

The method presented here will favor one output vector over the others. Even though we could give a distribution of potential outputs from the neighborhood of the initial prediction, we only give one of the possible correct output vectors for the given input vector $\vec{x}$. This output vector is the most common among the given neighborhood, and thus varies with neighborhood size and makeup.

As there is no other MOD algorithm to compare against, we chose to compare the highest output of the HMONN with the output from the naïve independence model. Due to the lack of alternative models, the naïve model was chosen as a reasonable comparison because it is the most likely to be used (in part, due to its simplicity) when working with

any MOD data without a specific MOD algorithm available.

HMONN is only a first step towards solving the MOD problem. Finding a full solution to the MOD problem is beyond the scope of this work, which is limited to HMONN. Future work will investigate more complete solutions to the MOD problem. This will include other machine learning models, such as relaxation network based approaches.

HMONN starts with some initial prediction and then uses the extra information provided by that initial prediction to give the output. The initial prediction is obtained using any machine learning method. We use the output from the naïve independence model as the initial prediction. Here, we choose to train an MLP classifier for each output. The outputs from each of the MLP classifiers are combined into an initial prediction. We present a modified K-Nearest Neighbor (KNN) algorithm to give the final prediction. A typical distance function for KNN is the Euclidean distance of the input features.

$$Dist(\vec{x}_1, \vec{x}_2) = \sqrt{\sum_{i=1}^{N} (x_{1,i} - x_{2,i})^2} \tag{3.1}$$

N is the number of features in the input space and $\vec{x}_1$ and $\vec{x}_2$ are input vectors. HMONN uses a different distance function where the initial prediction is used as part of the features for the distance function.

$$Dist(\{\vec{x}_1, \vec{y}_1\}, \{\vec{x}_2, \vec{y}_2\}) = \sqrt{\theta \sum_{i=1}^{N} (x_{1,i} - x_{2,i})^2 + (1-\theta) \sum_{i=1}^{M} (y_{1,i} - y_{2,i})^2} \tag{3.2}$$

N is the number of features in the input space, M is the number of outputs, θ is a weight on the range $[0, 1]$, each $\vec{x}$ is an input vector, and each $\vec{y}$ is an output vector. The value for θ emphasizes either the input space or the output space as more important. This modification of KNN captures the dependency between output variables by incorporating them into the input feature space. This emphasizes the importance of local context for the

final output given. HMONN takes the initial prediction from the MLP classifiers, uses this initial prediction as part of the features in a KNN algorithm, and picks the majority output vector from the neighborhood as the final prediction.

Chapter 4

Experimental Results

Accuracy of MOD classifiers was evaluated on three different types of data, UCI repository data, synthetic data, and real-world business data. This accuracy was compared to a baseline model that consists of a single classifier trained separately for each output, which we call the naïve model, where each separate prediction is combined into a single output vector. Accuracy is defined as follows.

$$MOD\_accuracy = \frac{\sum_i^D I(\{\vec{x}_i, \vec{z}_i\} \in T \cup \{D_i\})}{|D|} \tag{4.1}$$

where D is the test data set, $\vec{z}_i$ is the predicted output vector for instance i, $\vec{x}_i$ is the input for instance i, T is the training data set, and $I(x)$ is the indicator function returning 1 if the expression x is true and 0 otherwise. This accuracy metric counts any prediction as correct if some input vector $\vec{x}$ is labeled with the predicted output vector $\vec{z}$. This considers all possible correct output vectors as equally good.

Standard machine learning tasks with only nominal input features are common, and we assume that the same will hold for MOD data sets. HMONN shows clear improvement on these data sets. Many tasks also have real-valued features. While it is more difficult to find a duplicate $\vec{x}$ in these data sets, real-valued features will often have some level of discretization done to them, through either binning or rounding. This discretization that is already inherent in many current datasets, increases the likelihood of finding duplicate $\vec{x}$ vectors in the dataset. This alters the amount of dependence between the output variables (see Theorem 4.0.1). Thus, in many current datasets, real-valued features do not necessarily

take on a large range of values. This allows the given definition of accuracy to work in many cases with real-valued features.

To better handle real-valued features, the definition of accuracy could be extended to allow for *similar* values, as opposed to requiring values to be exactly equal. For example, assume that the test set has $\vec{x} \mapsto \vec{t}$ and $\vec{z} \neq \vec{t}$. If in $T \cup \{D_i\}$ there is an $\vec{x}_a$ with $\vec{x}_a \mapsto \vec{z}$ and $\vec{x}$ is similar to $\vec{x}_a$, $\vec{x} \mapsto \vec{z}$ could be counted as a correct MOD classification. We are currently working on extending MOD accuracy metrics to better support real-valued features.

We can demonstrate that duplicate $\vec{x}$ values within the training data are necessary for two output variables to be dependent. Theorem 4.0.1 claims that we can observe the dependence between two output variables directly in the training data. There may be a case for loose dependence that relies on different $\vec{x}$ vectors being only similar, but this work requires exact equality.

Theorem 4.0.1 *Given the random variables X, Y_i, and Y_j, with X being the input vector and both Y_i and Y_j being scalars from the output vector, if the two output variables, Y_i and Y_j, are conditionally dependent on each other given the input variable X and the training data T, then there is some input vector, $\vec{x}$, associated with multiple output vectors, $\vec{y}$, in T.*

Proof Assume that outputs $Y_i = y_1$ and $Y_j = y_2$ are conditionally dependent given the input variable X and the training data T. By the definition of statistical dependence this implies that, for some input vector $X = \vec{x}$, $P(y_1 \mid y_2, \vec{x}, T) \neq P(y_1 \mid \vec{x}, T)$. Assume that the output vector $\vec{y} = [y_1, y_2]^T$ is the only possible correct output for $\vec{x}$. Then it is the case that $P(y_1 \mid y_2, \vec{x}, T) = P(y_1 \mid \vec{x}, T) = 1$. This contradicts the definition of statistical dependence. Thus, there must be multiple possible output vectors for the input vector $\vec{x}$. ∎

We do show that HMONN can improve on accuracy in the case of real-valued inputs with some of the experiments on synthetic and UCI data, in spite of the issue of the frequency of exact $\vec{x}$ vectors for real-valued features. This is due to the fact that the nearest neighbor portion of the algorithm creates an implicit similarity function for the feature space. The

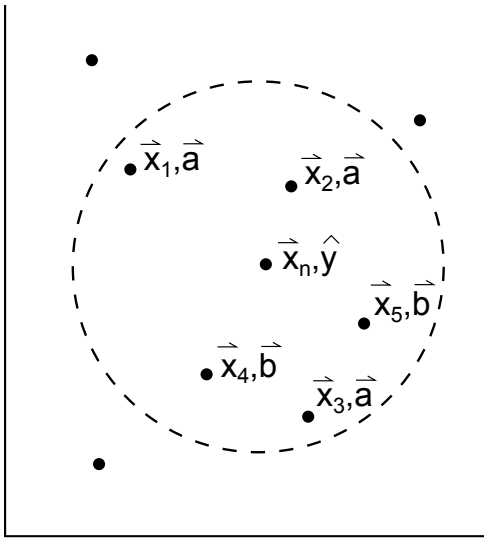


Figure 4.1: A graphical example of the implicit similarity function used by HMONN.

similarity function will behave differently based on the neighborhood size. This gives a distance-based voting for which outputs are correct for any given portion of the feature space. This causes the majority class for any given neighborhood in the feature space to always be the correct value. A graphical example of the implicit similarity function is shown in Figure 4.1. In the figure, the new input to be classified is $\vec{x}_n$ with an initial prediction of $\hat{y}$. The neighborhood of $\vec{x}_n, \hat{y}$, for $k = 5$, is shown by the dotted circle. Each of the points in the neighborhood is labeled with either $\vec{a}$ or $\vec{b}$. All of the points within the neighborhood of $\vec{x}_n, \hat{y}$ will be seen as similar to $\vec{x}_n, \hat{y}$. In this example, the input $\vec{x}_n$ would be labeled as $\vec{a}$, which is the majority vote in the neighborhood. Selecting outputs in this fashion avoids some of the difficulty with real-valued features, even though it does not solve the problem completely. We are currently exploring ways to fully resolve this problem, but such a solution is outside the scope of this paper.

4.1 Synthetic Data

Two different types of synthetic data were created. One used real-valued features in order to determine whether HMONN implicitly models a similarity function for the feature space, as hypothesized. The other used only nominal features.

Real-valued synthetic data was created stochastically using the following process. Given o output variables, a synthetic data set is generated by selecting c points in the input space as centroids. These points are each randomly assigned a number of probability vectors. A probability vector contains a probability distribution over possible output vectors. To generate an instance, a centroid is selected at random, the input values for that instance are generated by randomly perturbing the centroid according to a Gaussian distribution. An output vector is chosen by randomly selecting an output vector according to the probability distribution of a randomly chosen probability vector for that centroid. A graphical view of the generation process can be seen in Figure 4.2. This generation process attempts to model the fact that, for MOD problems, a portion of the input space can belong to more than one output vector.

Nominal synthetic data was created using the process outlined above with one difference. To generate a centroid, a center point for each feature was chosen from $\{0, 1, 2, 3\}$. New inputs were generated by adding a randomly selected value from $\{-1, 0, +1\}$ to the center point for that feature. Values above 3 were set to 3 and values below 0 were set to 0.

4.1.1 Experiments

Data was generated using the following values. The parameters were set to $o \in \{2, 3, 4\}$ (with four possible values for each output) and $c \in \{2, 4, 6, 8\}$. The number of inputs was set to 3 times the number of outputs. The number of probability vectors was the same as the number of centroids, 1.5 times the number of centroids, or 2 times the number of centroids. 5000 instances were generated for each data set. Some initial experimentation was used to determine values for k (the neighborhood size) and θ (the weight of the input versus the output space). We tested values of k from 1 to 11, and found that there was little difference between values of k in the experiments. We tested values of θ from $\{0, 0.25, 0.5, 0.75, 1\}$, and found that all values performed equally well in the experiments except $\theta = 0$, which performed slightly worse than the rest of the values tested. Thus, we used representative values $k = 7$

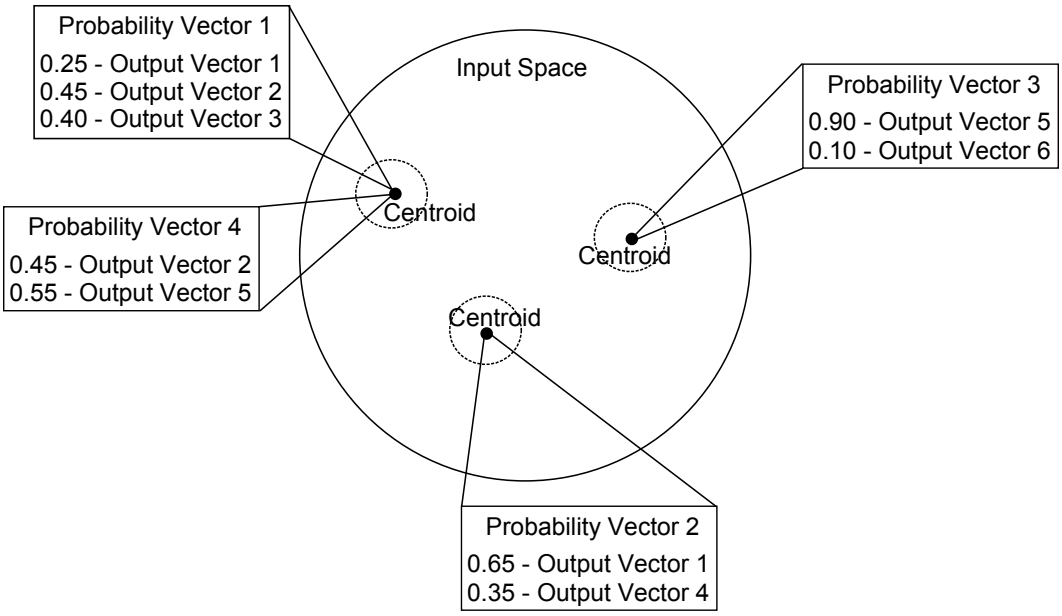


Figure 4.2: Example of a model used for data generation. Each centroid resides in a specific portion of the input space (denoted by the dotted lines). Any number of probability vectors can be associated with a given centroid. Each output vector that belongs to a given probability vector also has an emission probability for that output vector.

and $\theta = 0.5$ in the rest of the experiments. The value $k = 7$ allows for a reasonably sized neighborhood, and would give a distribution of output vectors if we were comparing more than the most common output in the neighborhood. The value $\theta = 0.5$ gives an equal balance between the input and output features, while avoiding the bad value of $\theta = 0$. This results in twelve datasets for each of two outputs, three outputs, and four outputs, giving a total of 36 datasets used.

The two sets of experiments show a comparison of HMONN versus the naïve model for real-valued features (first) and for nominal features (second). Experiments were run using 10-fold cross validation. The naïve neural network layer had a standard MLP with a single hidden layer of $2n$ nodes for each output with n being the number of attributes (this includes the outputs) in the corresponding data set. All experiments were run with a learning rate of 0.1 and stopped after 10 epochs without any improvement on a held-out validation set. Statistical significance was determined by using the Wilcoxon signed rank test with significance at $p < 0.05$.

4.1.2 Results

The results of comparing HMONN to the naïve model for real-valued features are given in Table I.

Model	2-Output	3-Output	4-Output	Total
HMONN	**0.760**	**0.877**	**0.905**	**0.847**
Naïve	0.718	0.770	0.762	0.750

Table I: Results of comparing HMONN to the naïve model for real-valued features. Bold values indicate that the values are statistically significant. The p-value for the total is $p < .0001$.

HMONN outperformed the naïve model for the real-valued synthetic data, and the improvement was always statistically significant. This is likely due to the fact that HMONN exploits the information contained in the local neighborhood in order to produce outputs. HMONN will have more information available with more outputs. This will make the neighborhood more specific, thus giving the algorithm a higher chance of finding a correct output.

The results of comparing HMONN to the naïve model for nominal features are given in Table II.

Model	2-Output	3-Output	4-Output	Total
HMONN	**0.703**	**0.864**	**0.893**	**0.820**
Naïve	0.655	0.758	0.713	0.709

Table II: Results of comparing HMONN to the naïve model for nominal features. Bold values indicate that the values are statistically significant. The p-value for the total is $p < .0001$.

HMONN outperformed the naïve model for the nominal synthetic data, and the improvement was always statistically significant. The exact same pattern is seen for the nominal synthetic data sets as for the real-valued data sets.

	Total	Nominal	Instances	2-Output	3-Output	4-Output
adult	15	9	48842	8	28	56
anneal	39	33	898	32	496	4960
autos	26	11	205	10	45	120
cars	7	7	1728	6	15	20
chess	7	7	28056	6	15	20
cmc	10	8	1473	7	21	35
colic	23	16	368	15	105	455
credit-a	16	10	690	9	36	84
heart-h	14	8	294	7	21	35
hepatitis	20	14	155	13	78	286
mushroom	23	23	8124	22	231	1540
nursery	9	9	12960	8	28	56
poker-hand	11	11	74987	10	45	120
post-operative	9	8	90	7	21	35
SPECT	23	23	267	22	231	1540
teachingAssistant	6	5	151	4	6	4
tic-tac-toe	10	10	958	9	36	84
vote	17	17	435	16	120	560
zoo	17	17	101	16	120	560

Table III: UCI Data Set Information. Includes total number of features, number of nominal features, number of instances, number of derived 2-output datasets, number of derived 3-output datasets, and number of derived 4-output datasets.

4.2 UCI Data

The bulk of the data used came from the UCI repository [8]. The UCI repository does not contain any data sets that are MOD decision problems. To allow for MOD learning, MOD data sets were created from the original UCI Data sets. This was done by allowing each nominal features to act as an output class for a derivative data set. If, for example, the number of outputs was set to two, each data set would become n derived data sets where n is the number of nominal features for the chosen data set. Each of these derived data sets consists of a nominal feature combined with the original output class acting as the output classes, with all of the other features acting as inputs. Similarly, for three or four outputs the original output class was combined with two or three (respectively) nominal features to act as the outputs. The number of data sets scales linearly in the number of inputs with two outputs,

quadratically with three outputs, and cubicly with four outputs. This is a contrived solution, but we assume that there is some dependency between input variables and the output variable — especially for data sets from the UCI repository. Allowing the input variable to become one of the output variables gives potential that there will be dependence between the outputs. Twenty different UCI repository data sets were used for the experiments. These data sets were chosen arbitrarily from those that had more than five nominal input features. Nominal input features were necessary in order to create the derivative data sets. Table III gives information for each of the data sets used. We varied the number of outputs between two to four outputs. Statistical significance was determined using the Wilcoxon signed rank test with significance at $p < 0.05$.

4.2.1 Experiments

Experiments were run using 10-fold cross validation. The neural networks used were a standard MLP with a single hidden layer of $2n$ nodes for each output with n being the number of attributes (this includes the output class) in the corresponding data set. All experiments were run with a learning rate of 0.1 and stopped after 10 epochs without any improvement on a held-out validation set. Missing values were replaced by the mean/mode.

4.2.2 Results

Table IV shows the results for the UCI dataset experiments. The table contains values for both HMONN and the naïve model compared by number of outputs. Statistically significant results are highlighted. HMONN outperformed the naïve model 79% of the time (with 68% of the time being statistically significant, see the Total columns). In some cases there was not a significant difference. In four cases, the naïve model outperformed HMONN. Figure 4.3 gives a graphical view of the difference in accuracy between the two models. Each number is obtained by averaging the results across all the derived data sets from the original UCI dataset for the given algorithm. The error bars give the standard deviation of the observed

differences. HMONN outperforms the naïve model in the majority of cases, as can be seen for the anneal, heart-h, and zoo datasets. Occasionally, the naïve model performs better, but never with the same magnitude. This, along with the other experiments performed, demonstrates the potential of HMONN as a model to solve MOD decision problems. This also validates the assumption that there is some dependence between the output variable and the input variables in the UCI data sets.

	2-Output		3-Output		4-Output		Total	
Dataset	H	NI	H	NI	H	NI	H	NI
adult	0.255	0.268	0.136	0.139	0.075	**0.083**	0.109	**0.116**
anneal	**0.756**	0.511	**0.700**	0.364	**0.659**	0.254	**0.664**	0.265
autos	**0.265**	0.192	**0.208**	0.118	**0.132**	0.068	**0.159**	0.088
car	**0.233**	0.229	**0.066**	0.063	**0.018**	0.017	0.067	0.065
chess	**0.100**	0.090	**0.020**	0.017	**0.004**	0.003	**0.024**	0.021
cmc	**0.358**	0.281	**0.243**	0.203	**0.173**	0.132	**0.217**	0.172
colic	**0.330**	0.255	**0.174**	0.106	**0.106**	0.048	**0.124**	0.064
credit-a	0.424	0.434	0.278	**0.297**	0.178	**0.202**	0.223	**0.245**
crx	**0.441**	0.438	0.265	**0.290**	0.161	**0.191**	0.210	**0.236**
heart-h	**0.509**	0.279	**0.401**	0.152	**0.300**	0.067	**0.357**	0.119
hepatitis	0.591	0.605	0.443	0.431	**0.381**	0.341	**0.401**	0.369
mushroom	**0.795**	0.775	**0.630**	0.590	**0.497**	0.451	**0.518**	0.473
nursery	**0.282**	0.278	**0.091**	0.089	**0.028**	0.027	**0.069**	0.067
poker-hand	**0.095**	0.091	**0.016**	0.015	**0.003**	0.002	**0.011**	0.011
post-operative	**0.360**	0.346	**0.248**	0.238	0.128	0.136	0.194	0.194
SPECT	**0.661**	0.651	**0.513**	0.494	0.397	0.372	0.415	0.391
teachingAssistant	0.217	0.183	**0.139**	0.061	0.088	0.017	**0.146**	0.083
tic-tac-toe	**0.453**	0.425	0.190	0.187	0.065	**0.074**	0.127	**0.130**
vote	**0.729**	0.708	**0.564**	0.540	**0.442**	0.417	**0.470**	0.445
zoo	**0.816**	0.568	**0.726**	0.492	**0.654**	0.400	**0.670**	0.420
# Significant	16	0	15	2	13	4	13	4

Table IV: Table with the results for the UCI experiments. The H columns signify the accuracy for HMONN and the NI columns signify the accuracy for the naïve independence model. Bold indicates that the model had significantly greater accuracy. The *# Significant* line indicates how many of the entries in each column were statistically significant.

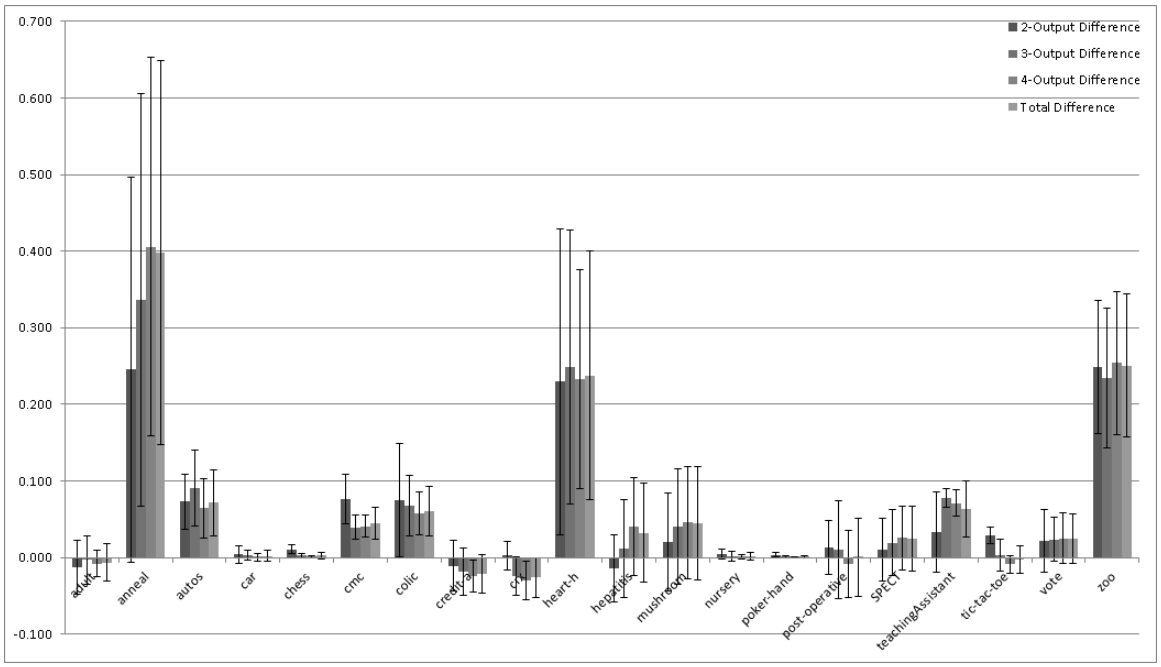


Figure 4.3: Graph of the results from the UCI dataset experiments. The graph represents the difference between the accuracy for HMONN and the accuracy for the naïve model. Thus, a positive value denotes HMONN performing better than the naïve model. Conversely, a negative value denotes the naïve model outperforming HMONN. The error bars give the standard deviation of the observed differences.

4.3 Business Application Data

Due to the novelty of the MOD problem space, we are not aware of any currently available MOD data sets. Not enough people have considered MOD problems to have compiled data sets to work with. A local business, InsideSales.com, was able to provide data for a real world MOD task. Due to the proprietary nature of this business data, we are only permitted to reproduce a de-identified version of this data. This data includes a two output data set and a three output data set. The data sets have fourteen nominal features and eight real-valued features. The two output data set has 32544 instances, and the three output data set has 32774 instances. Table V contains seven example instances from the two output data set.

The task is to determine the timing and method to contact business leads. Business practices would imply that these variables are dependent (given the input $\vec{x}$), the time you contact a lead depends on the method used, and the method used depends on the timing.

n	n	n	n	n	n	n	n	n	n	n	r	n	n	n	r	r	r	r	r	r	r	o	o
0	?	0	?	0	?	0	0	0	0	13	2	0	0	3	1	1	0	0	0	0	0	0	0
0	?	1	?	0	?	1	1	0	3	10	2	1	1	0	1	1	0	0	0	0	0	1	0
0	?	1	?	0	?	1	1	0	3	12	2	1	0	2	2	5	0	0	0	0	0	2	0
0	?	2	?	0	?	1	2	0	3	13	3	2	1	1	1	2	0	0	0	0	0	1	0
0	?	1	?	0	?	1	1	0	2	11	3	2	0	3	0	1	0	0	0	0	0	2	0
0	?	2	?	0	?	0	2	0	1	12	3	2	0	3	1	2	0	0	0	0	0	2	1
0	?	2	?	0	?	1	1	0	3	10	3	2	0	3	1	4	0	0	0	0	0	2	1

Table V: Example instances from the two output data set from InsideSales.com. Values of ? signify missing values, n signifies a nominal feature, r signifies a real-valued feature, and o signifies an output feature.

The experiment setup is the same as that for the UCI data sets with results coming from 10-fold cross validation and using same settings for the model parameters. The results are in Table VI. HMONN outperformed the naïve model in both cases. This shows that the improvement of HMONN seen in the UCI and synthetic data will also be seen in real-world MOD problems.

	HMONN	Naïve
2-Output	0.508	0.465
3-Output	0.346	0.307

Table VI: The table showing the results of the real-world business data experiments.

Chapter 5

Conclusions

We have given a formal definition for MOD problems, as a well as one method to solve such problems. We have defined the Hierarchical Multi-Output Nearest Neighbor model, with a naïve independence model as a first layer and a modified nearest neighbor model as the second layer. This model is based on the assumption that local context is a key element to solving MOD problems. HMONN consistently outperforms the baseline model, typically with statistical significance. This holds true for synthetic data, UCI repository data, and for one real-world business task. The synthetic data and the real-world business data are definitely MOD problems, however the synthetic data is not necessarily representative of real data, and there is not enough of the real data. The UCI data is thus used to supplement the other data sources, although it is necessary to create datasets that can only be assumed to represent MOD data.

Future work should develop solutions using other types of models (such as relaxation networks), an improved method for calculating accuracy on MOD problems, improved methods for validating new MOD algorithms, and new methods for identifying and collecting MOD data. With MOD problems, it is difficult to know how much dependency any given problem may hold. Many of the data sets that we used for validation could only be assumed to have a sufficient level of dependency. A method to identify the degree of dependency on a given data set would be extremely helpful for future work with MOD problems.

References

[1] D.H. Ackley, G.E. Hinton, and T.J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

[2] G. Bakır, T. Hofmann, and B. Schölkopf. *Predicting structured data*. The MIT Press, 2007.

[3] C. Bielza, G. Li, and P. Larrañaga. Multi-dimensional classification with bayesian networks. *International Journal of Approximate Reasoning*, 52(6):705–727, 2011.

[4] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[5] H. Daumé, J. Langford, and D. Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

[6] S. Džeroski. Multi-relational data mining: an introduction. *ACM SIGKDD Explorations Newsletter*, 5(1):1–16, 2003.

[7] J.L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[8] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[9] L. Getoor and L. Mihalkova. Learning statistical models from relational data. In *Proceedings of the 2011 international conference on Management of data*, pages 1195–1198. ACM, 2011.

[10] D. Heath, A. Zitzelberger, and C.G. Giraud-Carrier. A Multiple Domain Comparison of Multi-label Classification Methods. *Working Notes of the 2nd International Workshop on Learning from Multi-Label Data*, page 21, 2010.

[11] J.J. Hopfield and D.W. Tank. Neural computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

[12] J. Neville, M. Rattigan, and D. Jensen. Statistical relational learning: Four claims and a survey. In *Workshop SRL, Int. Joint. Conf. on AI*, 2003.

[13] S.J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, pages 1345–1359, 2009.

[14] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. "MIT Press", 2004.

[15] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(2):1453, 2006.

[16] G. Tsoumakas, I. Katakis, and I. Vlahavas. Mining multi-label data. *Data Mining and Knowledge Discovery Handbook*, pages 667–685, 2010.

[17] M.L. Zhang and Z.H. Zhou. A k-nearest neighbor based algorithm for multi-label classification. In *Granular Computing, 2005 IEEE International Conference on*, volume 2, pages 718–721. IEEE, 2005.