TASK DRIVEN EXTENDED FUNCTIONS OF MULTIPLE INSTANCES (TD-eFUMI)

A Thesis presented to
the Faculty of the Graduate School
at the University of Missouri

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
MATTHEW COOK
Dr. Alina Zare, Thesis Supervisor
DEC 2015

The undersigned, appointed by the Dean of the Graduate School, have examined the thesis entitled:

TASK DRIVEN EXTENDED FUNCTIONS OF MULTIPLE INSTANCES

(TD-*e*FUMI)

presented by Matthew Cook,

a candidate for the degree of Master of Science and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Alina Zare

Dr. Dominic Ho

Dr. Mihail Popescu

ACKNOWLEDGMENTS

I would like to thank my adviser, Dr. Alina Zare, for all of her invaluable guidance, support, and encouragement during my graduate studies. Also I would like to thank the member of my thesis committee Dr. Dominic Ho and Dr. Mihail Popescu for their help and valuable suggestions.

Thank you to Dr. Tory Cobb of NSWC-PCD and his team for their support and insight during my time working with them.

Additionally, thank you to all of my lab mates and in particular Changzhe Jiao, Xiaoxiao Du, and Brendan Alvey for the valuable discussions and insight throughout my studies.

And finally, thank you to my parents, Carolyn and Keith, and to my brother Jared for your continuous support throughout my studies.

TABLE OF CONTENTS

APPENDIX

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Dictionary learning techniques have proven to be a powerful method in the pattern recognition literature. Recently supervised dictionary learning has been used to achieve very good results on a number of different data types and applications. However, these supervised dictionary learning algorithms do not perform as well when the data contains a number of mislabeled data points. They rely on accurate labels

To solve this problem, an algorithm designed to incorporate multiple instance learning into the supervised dictionary learning framework. The proposed method combines the Task-Driven Dictionary Learning algorithm and Extended Functions of Multiple Instances. This new framework then allows the model deal with uncertainty in the labeling of training data while also maintaining the high degree of discrimination available through the Task-Driven Dictionary Learning model.

Results indicate that the proposed method, Task Driven Extended Functions of Multiple Instances, can maintain a high level of discriminatory performance with high levels of uncertainty in the labeling of training data. Test on real world wideband electromagnetic induction and synthetic aperture sonar data also indicate that these benefits also help the algorithm to outperform Task-Driven Dictionary Learning in classification tasks on these datasets.

Chapter 1

Introduction

In signal processing, the term *dictionary* can be used to describe a collection of generic signals, commonly called atoms, that can be combined, typically in a sparse manner, to recreate more complex signals. These atoms function very similarly to the bricks in LEGOS, the bricks can be combined in infinitely many ways to create very elaborate structures, but yet not every brick is needed for every design. This is very similar to how the dictionary functions in signal processing, the atoms can be combined to generate far more complex signals that what is present in the dictionary, just like how the bricks are not that complex by themselves, but when combined they can create very complex signals or structures. Again not every atom is needed to generate every signal, this idea of not including every atom is called sparsity.

These dictionaries can be defined in two different ways either learned from data or based on a mathematical model when an appropriate model of the data is known. For the dictionaries based off a mathematical model the data must all follow a generic form. An example of a method where the dictionary was defined based on a mathematical model is in [1]. In this model prior to using the dictionary much research went into defining what the response from targets should look like. Because of this the authors of JOMPP were then able to define a dictionary that followed this model in order to do target detection.

Methods where the dictionary is derived from a mathematical model however have one large drawback, the dictionary is created using assumptions about the data. Therefore, the dictionary may not describe the data or provide the best discriminating information if the model is incorrect. For this reason methods to learn dictionaries from data have been created. Learning the dictionary from the data alleviates the problem of having to know ahead of time what the data will look like.

1.1 Dictionary Learning

Dictionary learning, as mentioned before, is a method of learning how to recreate a data set using a much smaller set of building blocks called atoms [2]. There are numerous methods available that make this possible, several of which will be discussed later on. Many of these methods follow an unsupervised learning approach to learning the dictionaries which means that the atoms that are learned are only affected by the data itself and that there are no labels that tell the algorithm which signal from the data set uses a specific atom in the dictionary [3]. There are, however, methods that do incorporate supervised learning they will be detailed later.

In the unsupervised form, dictionary learning has become a popular tool in signal and image processing for its ability to extract discriminatory features from a data set. One of the biggest advantages of dictionary learning methods is that since they require so much data in order to train, often the atoms themselves will have little to no noise because it will cancel out in the training process [4]. Then when the learned dictionary is used to recreate the original data the reconstructed data will actually have less noise than the original data while still maintaining all of the details [5], [6].

Dictionary learning is typically an unsupervised method to learn a set of fundamental building blocks for a set of signals that can then be used to recreate the input signals [2]. In general dictionary learning corresponds to the problem of solving the optimization problem

$$\arg\min_{\boldsymbol{\alpha},\boldsymbol{D}} ||\boldsymbol{x} - \boldsymbol{D}\boldsymbol{\alpha}||_2^2, \tag{1.1}$$

where $\boldsymbol{x}$ is a set of signals of size $L \times 1$,L is the number of features, $\boldsymbol{D}$ is a $L \times K$ dictionary with K atoms, $\boldsymbol{\alpha}$ is $K \times 1$ and is the sparse weight vector that determines how to combine the elements of the dictionary, $||.||_2^2$ is the squared L_2-norm, and the product of $\boldsymbol{D}$ and $\boldsymbol{\alpha}$ is computed via matrix multiplication.

The dictionary learning problem can be seen graphically in Figure 1.1 for the case where both $\boldsymbol{D}$ and $\boldsymbol{\alpha}$ are already known. In the figure it is assumed that there is some sparsity constraint on the columns of $\alpha(x, D)$, which are the sparse weights of x. These constraints will be discussed in more detail later.

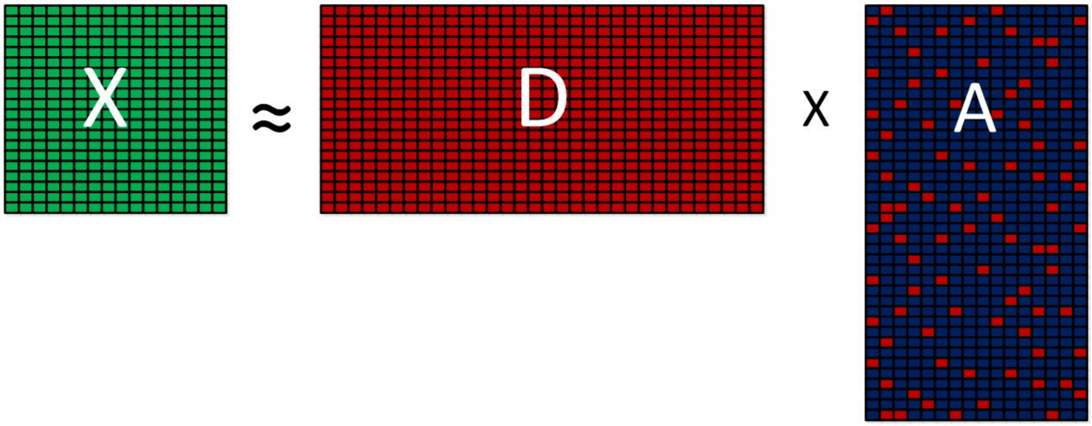


Figure 1.1: Illustration of how a dictionary is used to recreate signals. $\boldsymbol{X}$ is the actual data, the dictionary $\boldsymbol{D}$ is a collection of atoms that contain basic signatures of the data, and $\boldsymbol{A}$ is the matrix of all sparse profiles, $\boldsymbol{\alpha}$, that select which elements of the dictionary are needed to recreate the corresponding signal in $\boldsymbol{X}$.

1.2 Problem Statement

Dictionary Learning can also be extended so that it can be used in supervised learning methods as in [7]. However much like any other supervised learning problem, the training data must be labeled consistently for the algorithm to learn meaningful information from the data. There are some supervised dictionary learning algorithms that can handle missing data label, [8] and even others that can handle ambiguously labeled data, data points with multiple assigned classes, [9]. However, even in these methods the data that is labeled must be correct, and in the other case the actual class label must exist in the list of possible class labels.

Another way to look at the dictionary learning problem is through multiple instance learning algorithms. In multiple instance learning the true labels of a data point are not necessarily known, and instead groups of data points, called bags, have known labels. In the two class classification problem the bags are labeled target if there exists any target points in the bag and not target if the bag consists of purely non-target points. The goal of these algorithms is then to learn a model that can classify new test points into the target or not target classes, [10]. An example of when this is helpful is when the task is to do object level detection and the ground truth is given for the entire object and not each individual pixel. For generating labels from the ground truth a typical method is to label all points within a given radius as target and use these labels to train, even though this will label many non-target points as target. The multiple instance learning algorithm will be able to differentiate between these points and will treat the non-target points labeled as targets as non-targets, while a standard supervised learning algorithm will group all of the data together and treat it as target data which can severely hurt the results.

In particular the Extended Functions of Multiple Instances algorithm in [11] is a multiple instance learning algorithm that learns a dictionary that consists of one target element and many non-target elements and an extension to this algorithm is discussed in [12] that adds the ability to handle multiple target elements. These dictionary learning methods help overcome inconsistently labeled data which is a big problem in the supervised dictionary learning algorithms as mentioned before. However, in this implementation of multiple instance learning the label is used as a way to determine only which part of the dictionary is being updated and not directly in the classification function as is done in the Task Driven Dictionary Learning algorithm in [7], these algorithms will be explained in much greater detail in Sections 2.5 and 2.7. Having the classifier built into the Task Driven Dictionary Learning algorithm allows it to learn a more discriminative dictionary then the Extended Functions of Multiple Instances even though the latter will be able to overcome inconsistencies in the training data much better.

There is a gap that has not been thoroughly explored, a multiple instance learning algorithm that also includes the discriminatory boosting provided via a supervised dictionary learning model. In this thesis, the proposed method to bridge this gap is to merge the Task Driven Dictionary Learning algorithm in [7] with the Multi Target Extended Functions of Multiple Instances algorithm [12] thereby combining the benefits of both algorithms. The end result will be a multiple instance dictionary learning algorithm that will learn a highly discriminative dictionary from data that does not have precise pixel-wise ground truth information available.

Chapter 2

Literature Review

Dictionary learning algorithms encompass a very broad and complex group of learning methods even though most of the algorithms are based on small modifications of Equation 1.1. However, most dictionary learning algorithms encompass a sparse coding step and a dictionary update step. The sparse coding step decomposes the original signal into a sparse set of weights while the dictionary update step uses these sparse weights to compute a new dictionary. In this chapter, we will review these steps.

Once the basic structure of dictionary learning algorithms is known it is time to look at what these algorithms can do. Dictionary learning has been used in many applications including signal denoising [4], feature extraction [13], and dimensionality reduction [14] just to name a few. In many of these applications the sparse weights are exploited as feature vectors. Also they have incorporated the classification model directly into the dictionary learning model. The advantage of learning the classification model and the dictionary simultaneously is that the classification performance will have an effect on the dictionary instead of only the reconstruction performance.

From the supervised models another form of dictionary learning can be used that relaxes the requirement for completely labeled data, semi-supervised algorithms. These semi-supervised learning algorithms can be used in cases where not all of the data is labeled or when each data point has multiple labels.

The idea of labeled data can then be relaxed even further to where the labels are no longer point-wise and instead the labels apply to a group of points. Learning from this type of labeling can be exploited through multiple instance learning algorithms. In these mostly supervised algorithms, the labels are used only as an estimate for the true class label and the true labels do not directly affect the dictionaries.

2.1 Sparse Coding

Sparse coding is the first step in nearly all dictionary learning algorithms and the intent is to find the sparse combination of dictionary atoms that minimizes the error between the reconstruction and the actual signal. In this step, the dictionary is assumed to be known and fixed while the sparse weights are unknown. The sparse weights are found by minimizing:

$$\arg\min_{\boldsymbol{\alpha}} \frac{1}{2} \|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{\alpha}\|_2^2 + S, \tag{2.1}$$

where $\boldsymbol{x}$ is the input data, $\boldsymbol{D}$ is a known dictionary, $\boldsymbol{\alpha}$ is the sparse weights that are being learned, $\boldsymbol{D}\boldsymbol{\alpha}$ is the reconstruction of x using $\boldsymbol{D}$ and $\boldsymbol{\alpha}$, and S is a collection of sparsity promoting terms for the sparse weights $\boldsymbol{\alpha}$. There are several different ways to go about solving this minimization problem. Three of the most common methods are Iterative Thresholding (IT), Matching Pursuits (MP), and Basis Pursuits (BP). The main difference in these methods is what sparsity constraints they use and many of the different sparsity constraints also require different methods in order to solve for the sparse weights, as will be shown shortly.

The IT algorithms [15] have a sparsity term of the form $\lambda\|\boldsymbol{\alpha}^*\|_1$ which is the L_1 norm. These algorithms then set out to solve the optimization problem using a slight modification to gradient descent. The modification is that the sparsity term, $\lambda\|\boldsymbol{\alpha}^*\|_1$, is not included as part of the gradient. In these algorithms the sparse weights are allowed to be negative which means that the L_1 norm is non differentiable. Instead the sparsity is enforced on each iteration by thresholding the sparse weights after they have been updated via gradient descent. The update equation for the IT algorithms sparse coders is

$$\boldsymbol{a}^{i+1} = T_\lambda\left(\boldsymbol{\alpha}^i + \rho(\boldsymbol{x} - \boldsymbol{D}\boldsymbol{\alpha})\right) \tag{2.2}$$

where

$$T_\lambda(\boldsymbol{\alpha}) = \begin{cases} \alpha_k, & |\alpha_k| \geq \lambda \\ 0, & |\alpha_k| < \lambda \end{cases}, \quad \forall\alpha_k \in \boldsymbol{\alpha} \tag{2.3}$$

when hard thresholding is being used, Iterative Hard Thresholding Algorithm (IHTA), and

$$T_\lambda(\boldsymbol{\alpha}) = \begin{cases} \alpha_k - \operatorname{sign}(\alpha_k)\lambda, & |\alpha_k| \geq \lambda \\ 0, & |\alpha_k| < \lambda \end{cases}, \quad \forall\alpha_k \in \boldsymbol{\alpha} \tag{2.4}$$

when soft thresholding, Iterative Soft Thresholding Algorithm (ISTA), is being used.

The MP algorithms typically use a sparsity term of the form $\|\boldsymbol{\alpha}^*\|_0$, which is the L_0 pseudonorm that counts the number of nonzero entries in a vector. In this case, the sparsity term is not added to the general equation and is instead treated as a constraint similar to $\|\boldsymbol{\alpha}\|_0 \leq C$. Since this sparsity term is not well defined through derivatives, the solution to the MP algorithms do not involve differentiation, which tends to make them faster than the other approaches. One specific example and probably the most commonly used MP algorithm is the Orthogonal Matching Pursuits (OMP) algorithm in [16]. In general the OMP algorithm works by selecting a new dicitonary atom, by finding the index that maximizes the inner product of the dictionary and the input data, $\langle\boldsymbol{D}, \boldsymbol{x}\rangle$, and adding the index to the active list then directly computing the sparse weights for the active atoms via

$$\boldsymbol{\alpha}^*_{act} = (\boldsymbol{D}^\intercal_{act}\boldsymbol{D}_{act})^{-1}\boldsymbol{D}^\intercal_{act}x, \tag{2.5}$$

with the remaining non-active elements of $\boldsymbol{\alpha}$ being set to zero. Equation 2.5 is the solution to Equation 2.1 if there are no sparsity constraints. These steps are then repeated until either the residual error, $\|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{\alpha}\|_2^2$ falls below a set threshold or the number of chosen elements reaches C.

The final method discussed for sparse coding are the BP algorithms. These methods use the L_1 norm, just as the IT methods do, but they can also combine this with the L_2 norm as is done in [17] to form a sparsity promoting term of the form $\lambda_1\|\boldsymbol{\alpha}\|_1 + \frac{1}{2}\lambda_2\|\boldsymbol{\alpha}\|_2^2$. These methods require much more sophisticated methods to solve them. One example of an a solution to these problems is the algorithm presented in [18], the Least Angle Regression (LARS) model. The LARS algorithm is a method that generates a linear regression model by adding predictors to the model one at a time. These methods typically are more accurate than the MP family of algorithms but they are also considered to be slower.

2.2 Dictionary Update

The next step after having found the sparse profiles is to update the dictionary to reduce the error based on the current estimates of the profiles. The simplest method is the Method of Optimal Directions (MOD) in [19]. In this method the dictionary is updated as

$$\boldsymbol{D}_i = \boldsymbol{X}\boldsymbol{A}^\intercal_{n-1}(\boldsymbol{A}_{i-1}\boldsymbol{A}^\intercal_{i-1})^{-1}, \tag{2.6}$$

where $\boldsymbol{X}$ is the matrix of all input signals, $\boldsymbol{A}$ is the matrix of all the computed sparse profiles $\boldsymbol{\alpha}$, and the subscript i indicates the current iteration. This is the direct solution of the problem $\boldsymbol{X} = \boldsymbol{D}\boldsymbol{A}$ when solving for $\boldsymbol{D}$ when the other variables are known and it simply utilizes the pseudo-inverse of $\boldsymbol{A}$.

Another more commonly used dictionary update algorithm is the K-Singular Value Decomposition (K-SVD) algorithm [20]. This method is a direct generalization of the k-means algorithm where the constraints that only one center can be chosen and the sum of the elements is one are dropped, leaving the objective function as

$$\|\boldsymbol{X} - \boldsymbol{D}\boldsymbol{A}\|_F^2 \text{ subject to } \forall n \|\boldsymbol{\alpha}_n\|_0 \leq C, \tag{2.7}$$

where $\boldsymbol{X}$ is the matrix of all data points and $\boldsymbol{A}$ is the matrix of all the sparse weights. To perform the updates, Equation 2.7 is rewritten to pull the error term out;

$$\begin{aligned} \|\boldsymbol{X} - \boldsymbol{D}\boldsymbol{A}\|_F^2 &= \left\| \boldsymbol{X} - \sum_{k=1}^{K} \boldsymbol{d}_k \boldsymbol{a}_k \right\|_F^2 && (2.8) \\ &= \left\| \left(\boldsymbol{X} - \sum_{k \neq k_0}^{K} \boldsymbol{d}_k \boldsymbol{a}_k \right) - \boldsymbol{d}_{k_0} \boldsymbol{a}_{k_0} \right\|_F^2 && (2.9) \\ &= \|\boldsymbol{E}_{k_0} - \boldsymbol{d}_{k_0} \boldsymbol{a}_{k_0}\|_F^2 && (2.10) \end{aligned}$$

where $\boldsymbol{a}_k$ is the k^{th} row of the matrix A, i.e. all of the weights corresponding to the k^{th} dictionary element. At this point if this was k-means minimizing Equation 2.10 would result in computing the mean of the error term $\boldsymbol{E}_{k_0}$ over the samples. Doing this same operation here results in a solution that does not take into account the sparsity of the variable $\boldsymbol{a}$. Since the sparsity of a is known Equation 2.10 can be rewritten as

$$\|\boldsymbol{X} - \boldsymbol{D}\boldsymbol{A}\|_F^2 = \left\|\boldsymbol{E}_{k_0}^{\Lambda} - \boldsymbol{d}_{k_0} \boldsymbol{a}_{k_0}^{\Lambda}\right\|_F^2 \tag{2.11}$$

using only the active (non-zero), indicated by a superscript Λ, elements of $\boldsymbol{a}$ so that only the input signals in which $\boldsymbol{d}_{k_0}$ is active partake in the update. Instead of using the mean to solve Equation 2.11 the K-SVD algorithm calls for using Singular Value Decomposition (SVD) [21] to solve for the new dictionary element. In SVD, a matrix $\boldsymbol{X}$ is said to be equal to $\boldsymbol{U\Sigma V}^\intercal$, where the diagonal entries of $\boldsymbol{\Sigma}$ are the singular values of the matrix $\boldsymbol{X}$ and $\boldsymbol{U}$ and $\boldsymbol{V}$ are respectively the left-/right-singular vectors for the singular values on the diagonal of $\boldsymbol{\Sigma}$. For the K-SVD algorithm the first column of the matrix $\boldsymbol{U}$ is the new dictionary atom $\boldsymbol{d}_{k_0}$.

With this method each atom is updated one at a time in an iterative manner. This dictionary update method has been used extensively in [22], [23], [24], [4], and [25] and has shown good results.

Yet another method for updating the dictionary is to use stochastic or online gradient descent [26]. This method differs from both of the previous two methods in that the previous dictionary is actually used in the update to the new dictionary. The dictionary update in this method is of the form

$$\boldsymbol{D}^i = \boldsymbol{D}^{i-1} - \nabla_{\boldsymbol{D}} F(\boldsymbol{x}, \boldsymbol{D}^{i-1}), \tag{2.12}$$

where $\boldsymbol{D}^i$ is the new dictionary, $\boldsymbol{D}^{i-1}$ is the previous iteration's dictionary, and $F(x, D)$ is the objective function. This method is similar to the MOD update since it also updates the entire dictionary at one time. However, unlike the MOD method, instead of redefining the entire dictionary each iteration when gradient descent is used the dictionary is slowly adjusted in order to achieve a better fit to the data. A gradient descent dictionary update has been shown to work well in [27], [28], and [29].

An extension that can be used for the both the KSVD and Gradient Descent dictionary update methods is using a structured dictionary, or a dictionary whose atoms take on specific functions. In these models, the dictionary is not updated directly but instead the parameters that define the dictionary atoms are updated to better fit the data. This method has been used in [30], [31], and [32] to learn parametric dictionaries for specific types of signals.

2.3 Applications of Dictionary Learning

As mentioned earlier there are many applications for dictionary learning. The first application to be discussed is the use of dictionary learning as a denoising algorithm. This application can be for many applications including image denoising [33] or signal processing [34]. The dictionary learning framework works very well in denoising applications because the dictionaries can only learn patterns that appear many times in multiple data points. Since noise is assumed to be random, it is difficult for the dictionary to adapt and learn the noise and instead learns the underlying signal. Denoising can then be implemented by simply reconstructing the original data using the dictionary, which will give a very low error estimate to the original data without any noise.

Another application of dictionary learning is as a method to learn discriminative features that can be used to classify data. When given a data set and the goal is to classify the data, the first step is usually to find a set of features that can be used to classify the data. This means that the features need to discriminatory, coming up with features that are discriminatory is sometimes a very difficult task. Dictionary learning can be used to define a new feature space for classification as was done in [35] and [36]. When used in this sense, the dictionary elements are analogous to the feature definitions in the original feature space, and the sparse weights are used as the features in the new feature space. This new feature space will contain features that may lead to better classification because when a group of signals are very similar in the original feature space they will have very similar sparse weights.

An extension to the feature learning capability of dictionary learning is the ability to do dimensionality reduction. This application is a more constrained take on the feature extraction technique previously discussed in that the number of dictionary elements is less than the dimensionality of the original data. This methodology, as described in [37] and [14], is useful because the dictionary contains features that allow the data to be reconstructed with very low error. In this application the dictionary is being used to learn a smaller number of features that can still reconstruct the data and provide discriminatory ability as was discussed previously.

2.4 Supervised Dictionary Learning

The next logical progression after using the sparse profiles as features for a supervised learning algorithm is to include the supervision directly into the dictionary learning model. In this way, the dictionaries learned will not only be able to accurately reconstruct the data but also be more discriminatory since the classification error will affect the update of the dictionary itself.

There are several different ways in which to morph the dictionary learning method into supervised learning methods. One approach is to create multiple dictionaries where each one is assigned to be used for a specific class [13]. In these methods, each time a data point is encountered, only the dictionary corresponding to the class label will be updated. Doing this with enough data will then lead to multiple dictionaries that are good at characterizing data from their class and not very good for characterizing other classes. During testing, the class label is not known so to determine the class label the data point is reconstructed using all of the dictionaries and the class label of the dictionary that results is the lowest reconstruction error is then the assigned class. Another slight modification of this method is shown in [38].

In [39] and [7] the authors derive several methods that incorporate a classification model on top of the dictionary learning framework. These methods use the sparse representations as a feature vector to generate a linear classifier. More detail will be provided on the specifics of these methods, and in particular [7], in the next section.

Other methods of supervised dictionary learning include [40], [41], [42], and [43]. These methods directly use the individual values of the sparse weights to determine the actual class by associated individual dictionary atoms with a specific class. Then classification can be done in numerous ways either by simply selecting the class with the largest sparse weight or by performing more complex operations.

2.5 Task-Driven Dictionary Learning

In [7], the authors present an algorithm for supervised dictionary learning that simultaneously learns the dictionary along with a parameter that transforms the sparse profile returned by a sparse coder into the classification output .

Their algorithm, like most dictionary learning algorithms, consists of two main steps; the sparse coder and the dictionary update. In this algorithm the dictionary update stage comprises of both the dictionary update and the classification parameter update, this terminology was chosen simply to show similarity with previous algorithms. In their paper the authors chose to use projected stochastic gradient descent to iterate between these two phases of the dictionary learning problem.

For their algorithm, the authors chose to use elastic-net regularization,

$$\boldsymbol{\alpha}^*(\boldsymbol{x}, \boldsymbol{D}) \triangleq \underset{\boldsymbol{\alpha} \in \mathbb{R}^K}{\arg\min} \frac{1}{2}||\boldsymbol{x} - \boldsymbol{D}\boldsymbol{\alpha}||_2^2 + \lambda_1||\boldsymbol{\alpha}||_1 + \frac{\lambda_2}{2}||\boldsymbol{\alpha}||_2^2, \tag{2.13}$$

to complete the sparse coding step. Where λ_1 and λ_2 are parameters that control the sparseness of the solution and the solution $\boldsymbol{\alpha}(\boldsymbol{x}, \boldsymbol{D})$ is the profile of data point $\boldsymbol{x}$ using dictionary $\boldsymbol{D}$, where the dictionary is subject to the constraint,

$$\mathcal{D} \triangleq \{D \in \mathbb{R}^{L \times K} \text{s.t.} \forall k \in \{1, \ldots, K\}, ||\boldsymbol{d}_k||_2 \leq 1\}, \tag{2.14}$$

where $\mathcal{D}$ is the set of all possible $\boldsymbol{D}$ and $\boldsymbol{d}_k$ is the k^{th} column of $\boldsymbol{D}$. To solve the elastic-net problem the least angle regression (LARS) [18] algorithm is used.

The objective function that is used in [7] for the specific application of binary classification is,

$$\min_{\boldsymbol{D}\in\mathcal{D},\boldsymbol{w}\in\mathcal{W}} \mathbb{E}_{y,x}[\ell_s(y,\boldsymbol{w},\boldsymbol{\alpha}^*(\boldsymbol{x},\boldsymbol{D}))] + \frac{v}{2}||\boldsymbol{w}||_F^2, \tag{2.15}$$

where y is the true class label, $\boldsymbol{w}$ is the model parameter, $\boldsymbol{\alpha}(\boldsymbol{x},\boldsymbol{D})$ was defined in Equation 2.13, v is a smoothing parameter, and ℓ_s is any twice differentiable convex loss function such as the square or logistic loss, for example.

From Equation 2.15, the update equations can be derived by finding the gradient with respect to the variables $\boldsymbol{w}$ and $\boldsymbol{D}$, see [7] for more details. Once these equations have been derived they must be slightly modified in order to keep the constraints on the dictionary atoms intact. To do this the authors elect to project the resulting updates back onto the corresponding sets for $\boldsymbol{w}$ and $\boldsymbol{D}$, this is done by re-normalizing the dictionary columns so that they have an L_2 norm of one. For the second parameter $\boldsymbol{w}$ the values are also re-normalized in a similar fashion.

2.6 Semi-Supervised Dictionary Learning

Beyond the scope of supervised dictionary learning is the scenario where the data is either not entirely labeled or there are portions of the data that are unclear as to which class it belongs. In these situations, the supervised methods described previously will not function properly. To handle these unique situations, semi-supervised dictionary learning can be used.

In [44] and [8] the authors derive two semi-supervised dictionary learning algorithms that use multiple dictionaries, one for each class, concatenated together. Each sub-dictionary is updated using only data points from its own class and the unlabeled data points are used to update the entire dictionary. In this way the unlabeled data points will have a soft assignment to the class that most of its dictionary elements come from.

Another way to do semi-supervised dictionary learning is presented in [7] and simply adds the unsupervised dictionary learning problem onto its supervised dictionary learning objective function and adds a parameter that controls the tradeoff between the supervised and unsupervised functions. This method of semi-supervised dictionary learning lessens the impact of inconsistent or inaccurate labeling by not relying solely on the supervised learning dictionary update.

2.7 Functions of Multiple Instance (FUMI)

In [45] the FUMI algorithm is first presented as a way to learn a dictionary that consists of a single target element along with multiple non-target elements. In this model, a data sample labeled as a target can be written as a linear combination of the target dictionary element and the non-target elements,

$$\boldsymbol{x}_n^+ = \alpha_{nT}\boldsymbol{d}_T + \sum_{m=1}^{M} \alpha_{nm}\boldsymbol{d}_m. \tag{2.16}$$

Then, points labeled as non-targets are just a linear combination of the background dictionary elements,

$$\boldsymbol{x}_n^- = \sum_{m=1}^{M} \alpha_{nm}\boldsymbol{d}_m. \tag{2.17}$$

In both equations, α_{nT} is the weight of the target element, α_{nm} is the weight of the m^{th} non-target dictionary element, and d_T is the target dictionary element, and d_m is the m^{th} element in the non-target dictionary. In these equations, all of the weights are constrained such that they are positive and sum to one. This algorithm has been used to do sub-pixel target detection in hyperspectral images with good results [46].

This algorithm works well when the labeling for training points is accurate, but when the accuracy of the labeling is inaccurate the performance of the algorithm suffers greatly.

Thus, the Extended Functions of Multiple Instances (eFUMI) [11] aims to fix this problem. To do this the eFUMI no longer considers point-wise labeling and instead uses bag labels to identify the data. In this way, a group of points is labeled as a target bag if it contains at least one target point, and a bag is labeled as non-target only if it contains exclusively non-target points.

With the new method of assigning labels to training points the objective function for the eFUMI model takes the form,

$$\begin{aligned} F = \frac{-(1-u)}{2} \sum_{n=1}^{N} \left\| \boldsymbol{x}_n - z_n \alpha_{nT} \boldsymbol{d}_T - \sum_{m=1}^{M} \alpha_{nm} \boldsymbol{d}_m \right\|_2^2 \\ - \frac{u}{2} \left(\sum_{m=1}^{M} \| \boldsymbol{d}_m - \boldsymbol{\mu}_0 \|_2^2 + \| \boldsymbol{d}_T - \boldsymbol{\mu}_0 \|_2^2 \right) - \sum_{m=1}^{M} \gamma_m \sum_{n=1}^{N} \alpha_{nm}, \end{aligned} \tag{2.18}$$

where the first term is a measure of the error between the reconstruction of the actual signal $\boldsymbol{x}_n$ given the current target element $\boldsymbol{d}_T$, non-target elements $\boldsymbol{d}_m$, and weight values α_{nT} and α_{nm}. The second term holds the dictionary elements to have a mean roughly the same as the global data mean μ_0 and the last term is a sparsity promoting term for the proportions w_{nm} where $\gamma_k = \frac{\Gamma}{\sum_{n=1}^{N} \alpha_{nm}^{n-1}}$.

In order to minimize Equation 2.18, Expectation Maximization (EM) algorithm is used as the dictionary updates cannot be derived unless the probability of target z_n is known. When taking the expectation with respect to the two possible values of z_n, the objective function becomes

$$\begin{aligned} \mathbb{E}[F] = \sum_{z_n \in \{0,1\}} \left[-\frac{1}{2}(1-u) \sum_{n=1}^{N} P(z_n | \mathbf{x}_n, \boldsymbol{\theta}^{(i-1)}) \left\| \mathbf{x}_n - z_n \alpha_{nT} \mathbf{d}_T - \sum_{m=1}^{M} \alpha_{nm} \mathbf{d}_m \right\|_2^2 \right] \\ - \frac{u}{2} \sum_{m=1}^{M} \| \mathbf{d}_m - \boldsymbol{\mu}_0 \|_2^2 - \frac{u}{2} \| \mathbf{d}_T - \boldsymbol{\mu}_0 \|_2^2 - \sum_{m=1}^{M} \gamma_m \sum_{n=1}^{N} \alpha_{nm}, \end{aligned} \tag{2.19}$$

where

$$P(z_n|\mathbf{x}_n, \theta^{(i-1)}) = \begin{cases} p(z_n = 0|\mathbf{x}_n \in B_j^+, \boldsymbol{\theta}^{(i-1)}) = \exp\left(-\beta \left\| \mathbf{x}_n - \sum_{m=1}^{M} \alpha_{nm}\mathbf{d}_m \right\|_2^2\right) \\ p(z_n = 1|\mathbf{x}_n \in B_j^+, \boldsymbol{\theta}^{(i-1)}) = 1 - \exp\left(-\beta \left\| \mathbf{x}_n - \sum_{m=1}^{M} \alpha_{nm}\mathbf{d}_m \right\|_2^2\right) \\ p(z_n = 0|\mathbf{x}_n \in B_j^-, \boldsymbol{\theta}^{(i-1)}) = 1 \\ p(z_n = 1|\mathbf{x}_n \in B_j^-, \boldsymbol{\theta}^{(i-1)}) = 0 \end{cases} . \tag{2.20}$$

Here $\boldsymbol{B}_j^+$ indicates the j^{th} target bag and similarly $\boldsymbol{B}_j^-$ indicates the j^{th} non-target bag. From these equations the update equations for the dictionary elements and weight values can be computed.

This model allows for inconsistency in the labeling of the training data since the labeling is bag-wise instead of element-wise. This helps overcome the labeling inconsistency since the first step is to estimate the target probability before the updates are computed for the dictionary elements and the proportions. This allows the algorithm to learn which of the points in the positively labeled bags are actually targets and then updating the dictionary and proportions based on the probability that each point is actually a target point.

Chapter 3

Proposed Method

The proposed method in this thesis is a combination of two previously developed algorithms, Task Driven Dictionary Learning (TDDL) [7] and Multi Target Extended Functions of Multiple Instances (MT-eFUMI) [12]. Thus the proposed method will be called Task Driven Extended Functions of Multiple Instances (TD-eFUMI).

3.1 Generation of the Objective Function

In the proposed method the TDDL and MT-eFUMI algorithms are combined by utilizing the MT-eFUMI algorithm as a sparse coding method for the TDDL algorithm. In implementing this change almost the entire workings of the TDDL algorithm need to be revisited in order to cooperate correctly with the new sparse coding method. Since the MT-eFUMI algorithm utilizes multiple instance learning principles the new method must modify the TDDL algorithm to accomodate these new ideas.

The first modification to the TDDL algorithm is that the objective function, Equation 2.15, must be converted into a form that allows the the output value to shift as the algorithm learns more details as to whether each data point is actually a target or non-target point.

This functionality is added into the objective function by changing out the actual data point label y with the current estimated label indicated by z'. Following this change the actual objective function becomes,

$$\min_{\boldsymbol{D}\in\mathcal{D},\boldsymbol{w}\in\mathcal{W}} \mathbb{E}_{y,x}\left[\frac{1}{2}\left(z'_n - \boldsymbol{w}^{\intercal}\boldsymbol{\alpha}^{*}\right)^2\right] + \frac{v}{2}\|\boldsymbol{w}\|_2^2, \tag{3.1}$$

where ℓ_s has been replaced here with the explicit form of the squared error loss function which will be used for the new method, $\boldsymbol{w}$ is the classification parameter described in Section 2.5, $\boldsymbol{\alpha}$ is the sparse weight for a given data point, v is a regularization paramter for $\boldsymbol{w}$, and

$$z'_n := \begin{cases} 1, & z_n = 1 \\ -1, & z_n = 0 \end{cases}. \tag{3.2}$$

Where z_n comes from the MT-eFUMI objective function and is the probability that the current point, n, is considered a target point. This small change achieves the desired effect because the assigned label, y_n, no longer has a direct affect on the objective value and instead only the actual estimate of the current class label, z'_n affects the objective value.

The next modification involves the some of the constraints on the MT-eFUMI objective function in Equation 2.18. In the MT-eFUMI objective function, the second term is needed to help keep the dictionary elements so that they are close to the mean of the entire data set. If this term is left as it is the term will no longer have an effect on the dictionary elements. This is due to how the derivations are done for the update equation, more on this in the following section. For the proposed method, the second term in the MT-eFUMI objective function is moved into the objective function in Equation 3.1. With this change there also needs to be some way to control how important it is for the dictionary elements to be close to the global mean versus having good classification performance. This is done by adding in a weighting parameter u into the objective function so that the final objective function becomes

$$\begin{aligned}\mathcal{F}(\boldsymbol{w},\boldsymbol{\alpha}_n^*(\boldsymbol{x}_n,\boldsymbol{D})) = \mathbb{E}_z &\left[\frac{(1-u)\delta_n}{2}\left(z_n' - \boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n^*(\boldsymbol{x}_n,\boldsymbol{D})\right)^2\right] \\ &+ \frac{u}{2}\left(\sum_{t=1}^{T}\left\|\boldsymbol{d}_t^{(T)}-\boldsymbol{\mu}_0\right\|_2^2 + \sum_{m=1}^{M}\left\|\boldsymbol{d}_m^{(NT)}-\boldsymbol{\mu}_0\right\|_2^2\right) + \frac{v}{2}\left\|\boldsymbol{w}\right\|_2^2 \\ &+ \sum_{k=1}^{K}\sum_{l=2}^{L}\frac{s}{2}\left(\boldsymbol{d}_k(l)-\boldsymbol{d}_k(l-1)\right)^2,\end{aligned} \tag{3.3}$$

where $\boldsymbol{d}_t^{(T)}$ indicates the t^{th} element in the target dictionary, $\boldsymbol{d}_m^{(NT)}$ indicates the m^{th} element in the non-target dictioanry, and the variable δ_n also comes from the MT-eFUMI objective function and is defined as

$$\delta_n = \begin{cases} \epsilon\dfrac{N_M}{N_T}, & \text{for target points} \\ 1, & \text{for background points}\end{cases}, \tag{3.4}$$

while N_M and N_T are the number of labeled background and target samples respectively. The last term in the objective function is a smoothing term and the parameter s tunes the importance of this portion and the notation $\boldsymbol{d}_k(l)$ indicates the l^{th} dimension in the k^{th} dictionary element, k is used to reference the combined target and non-target dictionaries.

The smoothness term is added to the objective function here to help the algorithm deal with data that has a large amount of variability in the structure. This applies a penalty to the objective function whenever there is a large difference between consecutive dimensions in the dictionary atom. In most cases this is not needed but there are cases, discussed in Section 4 where this term is beneficial.

With these modifications the objective function for the proposed method, TD-eFUMI, is nearly complete, the only remaining part left to define is the sparse coding function $\boldsymbol{\alpha}^*$.

As mentioned previously the sparse coding function used in the new method is the MT-eFUMI algorithm; however, it is not used in its original form. Some of the constraints have already been moved out of the MT-eFUMI objective function to work better with the new algorithm. The next modification is to switch out the sparsity promoting term γ_m that was originally used with the L_1 norm that is more commonly used in dictionary learning applications. One might notice that the definition of the L_1 norm and the definition of γ_m are actually very close and this is indeed true. The L_1 norm is a generalization of the γ_m term that does not require the sparse weights $\boldsymbol{\alpha}$ to be positive as was required in the original MT-eFUMI algorithm. Not requiring the $\boldsymbol{\alpha}$'s to be positive is also more typical of traditional dictionary learning models, after this change the sparse coding function is defined to be

$$\boldsymbol{\alpha}_n^*(\boldsymbol{x}_n, \boldsymbol{D}) = \arg\min_{\boldsymbol{\alpha}} \frac{1}{2} \left\| \boldsymbol{x}_n - z_n \sum_{t=1}^{T} \boldsymbol{\alpha}_{nt} \boldsymbol{d}_t^{(T)} - \sum_{m=1}^{M} \boldsymbol{\alpha}_{nm} \boldsymbol{d}_m^{(NT)} \right\|_2^2 + \lambda \|\boldsymbol{\alpha}_n\|_1. \tag{3.5}$$

This concludes the modifications that are needed to define the objective function for the proposed method.

3.2 General Overview

Repeating from the previous section the objective function in its entirety is

$$\begin{aligned} \mathcal{F}(\boldsymbol{w}, \boldsymbol{\alpha}_n^*(\boldsymbol{x}_n, \boldsymbol{D})) = \mathbb{E}_z & \left[\frac{(1-u)\delta_n}{2} \left(z_n' - \boldsymbol{w}^\intercal \boldsymbol{\alpha}_n^*(\boldsymbol{x}_n, \boldsymbol{D}) \right)^2 \right] \\ & + \frac{u}{2} \left(\sum_{t=1}^{T} \left\| \boldsymbol{d}_t^{(T)} - \boldsymbol{\mu}_0 \right\|_2^2 + \sum_{m=1}^{M} \left\| \boldsymbol{d}_m^{(NT)} - \boldsymbol{\mu}_0 \right\|_2^2 \right) + \frac{v}{2} \|\boldsymbol{w}\|_2^2 \\ & + \sum_{k=1}^{K} \sum_{l=2}^{L} \frac{s}{2} \left(\boldsymbol{d}_k(l) - \boldsymbol{d}_k(l-1) \right)^2, \end{aligned} \tag{3.6}$$

where

$$\delta_n = \begin{cases} \epsilon \dfrac{N_M}{N_T}, & \text{for target points} \\ 1, & \text{for background points} \end{cases}, \tag{3.7}$$

$$\boldsymbol{\alpha}_n^*(\boldsymbol{x}_n, \boldsymbol{D}) = \arg\min_{\boldsymbol{\alpha}} \frac{1}{2} \left\| \boldsymbol{x}_n - z_n \sum_{t=1}^{T} \boldsymbol{\alpha}_{nt} \boldsymbol{d}_t^{(T)} - \sum_{m=1}^{M} \boldsymbol{\alpha}_{nm} \boldsymbol{d}_m^{(NT)} \right\|_2^2 + \lambda \|\boldsymbol{\alpha}_n\|_1, \qquad (3.8)$$

$\boldsymbol{w}$ is the classification parameter, $\boldsymbol{\alpha}_n^*(\boldsymbol{x}_n, \boldsymbol{D})$ is sparse weight vector for data point $\boldsymbol{x}_n$ using dictionary $\boldsymbol{D}$, $\boldsymbol{d}_t^{(T)}$ and $\boldsymbol{d}_m^{(NT)}$ are the t^{th} target and m^{th} non-target dictionary element, while $\boldsymbol{\mu}_0$ is the global data mean, and the notation $\boldsymbol{d}_k(l)$ indicates the l^{th} dimension in the k^{th} dictionary element, k is used to reference the combined target and non-target dictionaries.

Before detailing the derivation of the update equations the methodology that was used to create the proposed algorithm must be introduced. As the basis of the proposed method is the TDDL algorithm the main process of the proposed algorithm will be very similar to that of the original. As such Projected Stochastic Gradient Descent will be used to optimize over the dictionary and model parameter. Gradient descent is a proven optimization technique that works well in many situations, stochastic gradient descent adds some randomness to the updates by only using a single point or a small number of points to compute the updates [26].

Once the general minimization algorithm has been defined the next step is to layout the order of the components so that every step happens in the correct order. The first step in every iteration of the algorithm must always be the probability calculation, this is because all of the following computations rely on having a good estimate of the probability. The next variable that needs to be calculated in the sparse weights, these are the variables that are needed to update both the dictionary and the model parameter.

Computing the values for the sparse weights requires its own algorithm in order to find the optimal values. In the proposed implementation ISTA is used, described in Section 2.1, in order to minimize the formulation in Equation 3.5 to handle computing the sparse weights. The reason for choosing ISTA is that many of the more commonly used sparse coding algorithms, like OMP and the Lasso, cannot be easily adapted to work when there is only a probability that certain elements of the dictionary will be used. However, ISTA is proven to converge to the correct solution when certain parameters, again described in 2.1, are met and so it will work in this situation.

Finally, now that the sparse weights have been computed it is time to update the dictionary and the model parameter, but first the update equations must be derived.

3.3 Derivation

The method of optimization chosen for this algorithm is stochastic gradient descent, which is also what was used in the TDDL algorithm. In the gradient descent framework an objective function is minimized by taking steps in the direction of steepest descent. So these algorithms simply step down the objective function following the gradient, which means that gradient descent is guaranteed to find at a minimum a local minimum and, if the objective function is at a minimum strictly quasi-convex, a global minimum.

The general formulation for gradient descent based algorithm simply consists of updating the current parameter value with a small step in the direction of the negative gradient of the objective function or expressed mathematically,

$$A^i = A^{i-1} - \rho \nabla_A \mathcal{F}(A, \dots), \tag{3.9}$$

where A is the variable being optimized, t indicates the current iteration, ρ is a parameter that scales the amount of change in any one step and $\mathcal{F}(A, \dots)$ is the objective function in terms of A and possibly other variables.

Now moving on to the actual derivation of the proposed method, as stated before the objective function for the proposed method is defined as

$$\begin{aligned}\mathcal{F}(\boldsymbol{w}, \boldsymbol{\alpha}_n^*(\boldsymbol{x}_n, \boldsymbol{D})) = \mathbb{E}_z &\left[\frac{(1-u)\delta_n}{2}\left(z_n' - \boldsymbol{w}^\intercal \boldsymbol{\alpha}_n^*(\boldsymbol{x}_n, \boldsymbol{D})\right)^2\right] \\ &+ \frac{u}{2}\left(\sum_{t=1}^{T}\left\|\boldsymbol{d}_t^{(T)} - \boldsymbol{\mu}_0\right\|_2^2 + \sum_{m=1}^{M}\left\|\boldsymbol{d}_m^{(NT)} - \boldsymbol{\mu}_0\right\|_2^2\right) + \frac{v}{2}\|\boldsymbol{w}\|_2^2 \\ &+ \sum_{k=1}^{K}\sum_{l=2}^{L}\frac{s}{2}\left(\boldsymbol{d}_k(l) - \boldsymbol{d}_k(l-1)\right)^2.\end{aligned} \tag{3.10}$$

with the sparse representation $\boldsymbol{\alpha}^*(\boldsymbol{x}_n, \boldsymbol{D})$ being computed via a slight modification of the MT-eFUMI algorithm computed via

$$\boldsymbol{\alpha}_n^*(\boldsymbol{x}_n, \boldsymbol{D}) = \arg\min_{\boldsymbol{\alpha}} \frac{1}{2}\left\|\boldsymbol{x}_n - z_n\sum_{t=1}^{T}\boldsymbol{\alpha}_{nt}\boldsymbol{d}_t^{(T)} - \sum_{m=1}^{M}\boldsymbol{\alpha}_{nm}\boldsymbol{d}_m^{(NT)}\right\|_2^2 + \lambda\|\boldsymbol{\alpha}_n\|_1. \tag{3.11}$$

The first step in optimizing Equation 3.10 is to compute the probabilities for z_n, so that each of the following steps can be completed. In order to compute the probabilities the previous iteration's sparse weights for the background terms are used this part of the proposed algorithm is taken directly from the MT-eFUMI algorithm and does not need to be derived, the equation for computing these probabilities is shown in Equation 2.20.

The next step is to calculate the sparse weights $\boldsymbol{\alpha}^*$ by solving Equation 3.11. This is done by taking the expectation with respect to the hidden variable z_n

$$\mathbb{E}_{z_n}\left[\boldsymbol{\alpha}^*\right] = \mathbb{E}_{z_n}\left[\arg\min_{\boldsymbol{\alpha}_n} \frac{1}{2}\left\|\boldsymbol{x}_n - z_n\sum_{t=1}^{T}\boldsymbol{\alpha}_{nt}\boldsymbol{d}_t^{(T)} - \sum_{m=1}^{M}\boldsymbol{\alpha}_{nm}\boldsymbol{d}_m^{(NT)}\right\|_2^2 + \lambda\|\boldsymbol{\alpha}_n\|_1\right] \tag{3.12}$$

$$= \arg\min_{\boldsymbol{\alpha}_n} \mathbb{E}_{z_n}\left[\frac{1}{2}\left\|\boldsymbol{x}_n - z_n\sum_{t=1}^{T}\boldsymbol{\alpha}_{nt}\boldsymbol{d}_t^{(T)} - \sum_{m=1}^{M}\boldsymbol{\alpha}_{nm}\boldsymbol{d}_m^{(NT)}\right\|_2^2\right] + \lambda\|\boldsymbol{\alpha}_n\|_1 \tag{3.13}$$

$$\begin{aligned} = \arg\min_{\boldsymbol{\alpha}_n} \Bigg(& P(z_n = 1)\frac{1}{2}\left\|\boldsymbol{x}_n - \sum_{t=1}^{T}\boldsymbol{\alpha}_{nt}\boldsymbol{d}_t^{(T)} - \sum_{m=1}^{M}\boldsymbol{\alpha}_{nm}\boldsymbol{d}_m^{(NT)}\right\|_2^2 \\ & +P(z_n = 0)\frac{1}{2}\left\|\boldsymbol{x}_n - \sum_{m=1}^{M}\boldsymbol{\alpha}_{nm}\boldsymbol{d}_m^{(NT)}\right\|_2^2\Bigg) + \lambda\|\boldsymbol{\alpha}_n\|_1 \end{aligned} \tag{3.14}$$

Then using the values computed for the probabilities and the IST algorithm described in Section 2.1, Equation 3.14 can be minimized to find the sparse weights.

Now both the dictionary $\boldsymbol{D}$ and the model parameters $\boldsymbol{w}$ can be updated. The first step is to again evaluate the expectation of Equation 3.10, to make the math here easier to read only the error term will be shown here since the remaining terms can be excluded from the expectation,

$$\mathcal{F}(\boldsymbol{w}, \boldsymbol{x}_n, \boldsymbol{D}) = \mathbb{E}_{z_n}\left[\frac{(1-u)\delta}{2}\left(z_n' - \boldsymbol{w}^\intercal\boldsymbol{\alpha}_n^*\right)^2\right] \tag{3.15}$$

$$= \frac{(1-u)\delta}{2}[1 + 2(1 - 2P(z_n = 1))]\boldsymbol{w}^\intercal\boldsymbol{\alpha}_n + (\boldsymbol{w}^\intercal\boldsymbol{\alpha}_n)^2, \tag{3.16}$$

The full derivation has been relegated to the Appendix, Section B.1, for ease of reading.

Now moving on to the easier of the two update equations, computing the gradient of

Equation 3.16 with respect to $\boldsymbol{w}$,

$$\nabla_{\boldsymbol{w}}\mathcal{F}(\boldsymbol{w},\boldsymbol{x}_n,\boldsymbol{D}) = \nabla_{\boldsymbol{w}}\left[\frac{(1-u)\delta}{2}[1+2(1-2P(z=1))]\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n + (\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n)^2 \right.$$
$$\left. +\frac{u}{2}\left(\sum_{t=1}^{T}\left\|\boldsymbol{d}_t^{(T)}-\boldsymbol{\mu}_0\right\|_2^2+\sum_{m=1}^{M}\left\|\boldsymbol{d}_m^{(NT)}-\boldsymbol{\mu}_0\right\|_2^2\right)+\frac{v}{2}\left\|\boldsymbol{w}\right\|_2^2)\right] \tag{3.17}$$

$$= \frac{(1-u)\delta}{2}[(1-2P(z=1))\boldsymbol{\alpha}_n + 2(\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n)\boldsymbol{\alpha}_n] + v\boldsymbol{w}. \tag{3.18}$$

Now plugging the gradient derived into the general form shown in Equation 3.9 leaves

$$\boldsymbol{w}^t = \boldsymbol{w}^{t-1} - \rho\left(\frac{(1-u)\delta}{2}[(1-2P(z=1))\boldsymbol{\alpha}_n + 2(\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n)\boldsymbol{\alpha}_n] + v\boldsymbol{w}\right), \tag{3.19}$$

as the update equation.

Now moving to the more difficult update equation for the dictionary, here the objective function is differentiated with respect to the the individual atoms of the dictionary. In this way the update equations will be for the individual atoms and the dictionary will then be updated atom-by-atom. The first step in deriving the update equation for the dictionary is to derive the closed form solution for $\boldsymbol{\alpha}^*$, but since the dictionary will be updated in an atom-by-atom approach the closed form solution need only be in terms of a single element of the full solution, i.e. solving for α_t and α_m will be enough. In order to find the closed form solution of Equation 3.11 for both α_t and α_m the derivative must be taken with respect to both variables and then the results will be set equal to zero in order to solve for α_t and α_m.

First the equation for $\boldsymbol{\alpha}^*$ is rewritten to evaluate the expectation operator

$$\boldsymbol{\alpha}_n^*(\boldsymbol{x}_n, \boldsymbol{D}) = \mathbb{E}_{z_n}\left[\frac{1}{2}\left\|\boldsymbol{x}_n - z_n\sum_t^T \alpha_{nt}\boldsymbol{d}_t^{(T)} - \sum_m^M \alpha_{nm}\boldsymbol{d}_m^{(NT)}\right\|_2^2\right] + \lambda\|\boldsymbol{\alpha}_n\|_1 \tag{3.20}$$

$$\begin{aligned} &= \frac{P(z=1)}{2}\left\|\boldsymbol{x}_n - \sum_t^T \alpha_{nt}\boldsymbol{d}_t^{(T)} - \sum_m^M \alpha_{nm}\boldsymbol{d}_m^{(NT)}\right\|_2^2 \\ &\quad + \frac{P(z=0)}{2}\left\|\boldsymbol{x}_n - \sum_m^M \alpha_{nm}\boldsymbol{d}_m^{(NT)}\right\|_2^2 + \lambda\|\boldsymbol{\alpha}_n\|_1 \end{aligned} \tag{3.21}$$

Now that the expectation has been evaluated the closed forms of both α_t and α_m can be found by differentiating Equation 3.21 with respect to a specific value of t in $\boldsymbol{\alpha}_n$ indicated by α_{nt_0} and a specific value of m in $\boldsymbol{\alpha}_n$ indicated by α_{nm_0}, and then setting the resulting equations equal to zero. For α_{nt_0} this results in

$$\begin{aligned} \alpha_{nt_0} = \left(P(z=1)\left(\boldsymbol{d}_{t_0}^{(T)}\right)^{\mathsf{T}}\boldsymbol{d}_{t_0}^{(T)}\right)^{-1}\Bigg(P(z=1)\left(\boldsymbol{d}_{t_0}^{(T)}\right)^{\mathsf{T}}\Bigg(\boldsymbol{x}_n - \sum_{t,t\neq t_0}^T \alpha_{nt}\boldsymbol{d}_t^{(T)} \\ - \sum_m^M \alpha_{nm}\boldsymbol{d}_m^{(NT)}\Bigg) + \lambda\text{sign}(\alpha_{nt_0})\Bigg), \end{aligned} \tag{3.22}$$

and the result for α_{nm_0} is

$$\begin{aligned} \alpha_{nm_0} = \left(\left(\boldsymbol{d}_{m_0}^{(NT)}\right)^{\mathsf{T}}\boldsymbol{d}_{m_0}^{(NT)}\right)^{-1}\Bigg(\left(\boldsymbol{d}_{m_0}^{(NT)}\right)^{\mathsf{T}}\boldsymbol{x}_n - \boldsymbol{d}_{m_0}^{\mathsf{T}}\sum_{m,m\neq m_0}^M \alpha_{nm}\boldsymbol{d}_m \\ -P(z=1)\left(\boldsymbol{d}_{m_0}^{(NT)}\right)^{\mathsf{T}}\sum_t^T \alpha_{nt}\boldsymbol{d}_t^{(T)} + \lambda\text{sign}(\alpha_{m_0})\Bigg). \end{aligned} \tag{3.23}$$

Again the full derivation has been moved to the Appendix, Section B.2.

Now with Equations 3.22 and 3.23 in hand the next step is to differentiate Equation 3.16 with respect to the dictionary atom needing to be updated, $\boldsymbol{d}_t^{(T)}$ and $\boldsymbol{d}_m^{(NT)}$. Since the derivation for both of these two dictionary elements are the same for part of the derivation the subscript k will be used to generically indicate a dictionary element anywhere in the derivation that the equations for both are the same, when the equations are no longer the same the notation will switch back to the previous notation.

$$\begin{aligned}\nabla_{\boldsymbol{d}_k}&\mathcal{F}(\boldsymbol{w},\boldsymbol{x}_n,\boldsymbol{D}) = \\ &\frac{(1-u)\delta}{2}\left(2(1-2P(z=1))w_k\nabla_{\boldsymbol{d}_k}\alpha_{nk} + 2(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{\alpha}_n)w_k\nabla_{\boldsymbol{d}_k}\alpha_{nk}\right) + u(\boldsymbol{d}_k - \boldsymbol{\mu}_0)\end{aligned} \tag{3.24}$$

where $\nabla_{\boldsymbol{d}_{nk}}$ is replaced with either

$$\begin{aligned}\nabla_{d_t^{(T)}}\alpha_{nt_0} = &\frac{-2\boldsymbol{d}_{t_0}^{(T)}}{P(z=1)\left(\left(\boldsymbol{d}_{t_0}^{(T)}\right)^{\mathsf{T}}\boldsymbol{d}_{t_0}^{(T)}\right)^2}\left(P(z=1)\left(\boldsymbol{d}_{t_0}^{(T)}\right)^{\mathsf{T}}\boldsymbol{e}_t + \lambda\text{sign}(\alpha_{nt_0})\right) \\ &+ \left(P(z=1)\left(\boldsymbol{d}_{t_0}^{(T)}\right)^{\mathsf{T}}\boldsymbol{d}_{t_0}^{(T)}\right)^{-1}\boldsymbol{e}_t\end{aligned} \tag{3.25}$$

or

$$\begin{aligned}\nabla_{\boldsymbol{d}_{m_0}^{(NT)}}\alpha_{nm_0} = &\frac{-2\boldsymbol{d}_{m_0}^{(NT)}}{\left(\left(\boldsymbol{d}_{m_0}^{(NT)}\right)^{\mathsf{T}}\boldsymbol{d}_{m_0}^{(NT)}\right)^2}\left(P(z=1)\left(\boldsymbol{d}_{t_0}^{(T)}\right)^{\mathsf{T}}\boldsymbol{e}_m + \lambda\text{sign}(\alpha_{nt_0})\right) \\ &+ \left(\left(\boldsymbol{d}_{m_0}^{(NT)}\right)^{\mathsf{T}}\boldsymbol{d}_{m_0}^{(NT)}\right)^{-1}\boldsymbol{e}_m\end{aligned} \tag{3.26}$$

depending on whether a target or a background dictionary element is currently being updated. The error term in each equation above is then written as

$$\boldsymbol{e}_t = \left(\boldsymbol{x}_n - \sum_{t,t\neq t_0}^{T}\alpha_{nt}\boldsymbol{d}_t^{(T)} - \sum_{m}^{M}\alpha_{nm}\boldsymbol{d}_m^{(NT)}\right) \tag{3.27}$$

for the target elements and

$$\boldsymbol{e}_m = \left(\boldsymbol{x}_n - \sum_{m,m\neq m_0}^{M}\alpha_{nm}\boldsymbol{d}_m^{(NT)} - P(z=1)\sum_{t}^{T}\alpha_{nt}\boldsymbol{d}_t^{(T)}\right) \tag{3.28}$$

for the non-target elements.

Then combining Equations 3.24 by replacing the subscript k with t and 3.25 generates the update for all of the target dictionary atoms. Equations 3.24 and 3.26 generate the update equation for the non-target dictionary atoms when the subscript k's are changed to m's in 3.24.

The derivative of the smoothing term has been left out of the main derivation here as it cannot be defined in the vector notation the rest of the model uses. However, by the additive properties of gradients the smoothness term can be defined separately then added back into the model once it is derived. The derivative of this term is simple

$$\nabla_{\boldsymbol{d}_k(l)} \sum_{k=1}^{K} \sum_{l=2}^{L} \frac{s}{2} \left(\boldsymbol{d}_k(l) - \boldsymbol{d}_k(l-1)\right)^2 \tag{3.29}$$

$$= s\left(\boldsymbol{d}_k(l) - \boldsymbol{d}_k(l-1)\right). \tag{3.30}$$

To add this term onto the previously derived updates one can computed this gradient separately then form the vector from the outputs and add it back onto the gradient derived in Equation 3.24.

3.4 Algorithm

With the update equations being derived the next step is to organize the equations into the form of an algorithm in order to learn a dictionary and model parameter that can be used for the classification of new data points. The algorithm for this method follows very closely to the algorithm in [7], once the initial conditions are set the sparse weights are found and are then used to update the dictionary and the model parameter.

The elements are updated via the stochastic gradient descent algorithm [26] or online gradient descent. In this version of gradient descent either a single point or a small batch of points are used to compute the update to the dictionary and model parameters. In this way the variables are updated numerous times in a single iteration through all of the data. A summary of the algorithm is shown in Algorithm 1.

The initialization for $\boldsymbol{\alpha}$, $\boldsymbol{w}$, $\boldsymbol{D}$ is done sequentially since the value of each one depends on one of the other values. The dictionary is the first parameter to be initialized since all of the others must be derived from the dictionary. Coming up with a good initial dictionary is very important to the performance of the algorithm, if it is initialized poorly the coefficients of the sparse weights will also be poor. In this implementation the initial dictionary elements are chosen from the training data at random. For the target dictionary elements only points that were initially labeled as target will be considered, and the same follows for the non-target dictionary elements. At this point all of the randomly collected elements are concatenated together in order to form the initial dictionary, once combined the dictionary is normalized so that each of the dictionary elements has an L_2 norm of 1.

The next step in the initialization process is to compute the initial value of $\boldsymbol{\alpha}$. For this initialization Equation 1.1 is optimized directly for $\boldsymbol{\alpha}$ while holding $\boldsymbol{D}$ fixed, by differentiating with respect to $\boldsymbol{\alpha}$ and setting the result equal to zero. Therefore the initial value of the sparse weights is computed as

$$\boldsymbol{\alpha}_{\text{init}} = \left(\boldsymbol{D}_{\text{init}}^{\intercal}\boldsymbol{D}_{\text{init}}\right)^{-1}\boldsymbol{D}_{\text{init}}^{\intercal}\boldsymbol{x}. \tag{3.31}$$

This initialization does not incorporate any sparsity promotion and as such will most likely not be sparse, however it does minimize the unconstrained problem therefore it gives the best possible reconstruction error, assuming the objective function is convex. The reconstruction error is more important for the initialization since this estimate is used primarily to compute the probability that a point contains a target dictionary atom.

The last value that needs to be initialized is the model parameter w. This value is again

Algorithm 1 Summary of algorithm for proposed method.

```
Require: initial values for $\boldsymbol{\alpha}$, $\boldsymbol{w}$, $\boldsymbol{D}$, parameters $i_0$, $\rho$, $u$, $E$, $\lambda$, $v$, $\beta$, batch size
for i=1:I do
    Set counter = 1
    while counter $\leq$ number of samples do
        Draw next batch of samples
        Compute probabilities of current batch
            $$P(z_n|\mathbf{x}_n, \theta^{(i-1)}) = \begin{cases} p(z_n = 0|\mathbf{x}_n \in B_j^+) = \exp\left(-\beta \left\|\mathbf{x}_n - \sum_{m=1}^{M} \alpha_{nm}\mathbf{d}_m^{(NT)}\right\|_2^2\right) \\ p(z_n = 1|\mathbf{x}_n \in B_j^+) = 1 - \exp\left(-\beta \left\|\mathbf{x}_n - \sum_{m=1}^{M} \alpha_{nm}\mathbf{d}_m^{(NT)}\right\|_2^2\right) \\ p(z_n = 0|\mathbf{x}_n \in B_j^-) = 1 \\ p(z_n = 1|\mathbf{x}_n \in B_j^-) = 0 \end{cases}$$
        Compute sparse weights of current batch via ISTA by solving
            $$\arg\min_{\boldsymbol{\alpha}} P(z = 1)\frac{1}{2}\left\|\boldsymbol{x}_n - \sum_{t=1}^{T} \boldsymbol{\alpha}_{nt}\boldsymbol{d}_t^{(T)} - \sum_{m=1}^{M} \boldsymbol{\alpha}_{nm}\boldsymbol{d}_m^{(NT)}\right\|_2^2 + P(z = 0)\frac{1}{2}\left\|\boldsymbol{x}_n - \sum_{m=1}^{M} \boldsymbol{\alpha}_{nm}\boldsymbol{d}_m^{(NT)}\right\|_2^2 + \lambda\|\boldsymbol{\alpha}_n\|_1$$
        for $\boldsymbol{\alpha}_n$
        select learning rate $\rho_i = \min\left(\rho, \frac{\rho i_0}{i}\right)$
        Update model parameter $\boldsymbol{w}$ via
            $$\boldsymbol{w}^i = \boldsymbol{w}^{i-1} - \rho_t\left(\tfrac{1}{2}(1-u)\delta[(1-2P(z=1))\boldsymbol{\alpha}_n + 2(\boldsymbol{w}^\intercal\boldsymbol{\alpha}_n)\boldsymbol{\alpha}_n] + v\boldsymbol{w}\right)$$
        Update dictionary $\boldsymbol{D}$ via
            $$\boldsymbol{d}_{\boldsymbol{k}}^i = \boldsymbol{d}_{\boldsymbol{k}}^{i-1} - \rho_t\left(\nabla_{\boldsymbol{d}_k}\mathcal{F}(\boldsymbol{w}, \boldsymbol{x}_n, \boldsymbol{D})\right)$$
        where $\nabla_{\boldsymbol{d}_k}\mathcal{F}(\boldsymbol{w}, \boldsymbol{x}_n, \boldsymbol{D})$ is defined by Equations 3.24 and 3.25 for positive points and by Equations 3.24 and 3.26 for negative points.
        counter = counter + batch size
    end while
    if $(\|\Delta\boldsymbol{D}\|_F \leq E) \wedge (\|\Delta\boldsymbol{w}\|_2 \leq E)$ then
        break
    end if
end for
return $\boldsymbol{w}$ and $\boldsymbol{D}$
```

obtained directly from the classification function

$$\hat{Y} = \boldsymbol{\alpha}^{\intercal}\boldsymbol{w} \tag{3.32}$$

where $\hat{Y}$ is the estimated class label, by substituting the actual label Y in for $\hat{Y}$ and solving for $\boldsymbol{w}$ the initial value can be obtained from

$$\boldsymbol{w}_{\text{init}} = (\boldsymbol{\alpha}_{\text{init}}\boldsymbol{\alpha}_{\text{init}}^{\intercal})^{-1}\boldsymbol{\alpha}_{\text{init}}Y. \tag{3.33}$$

3.5 Parameters

With this method there comes several parameters that have to be tuned for a specific problem, in this section the most important parameters will be discussed. The first parameter to be discussed is λ, in the proposed method λ is the parameter that controls the importance of the L_1 norm in the sparse coder which in turn affects the sparsity of the vector $\boldsymbol{\alpha}$. The L_1 norm acts like a laplacian prior on the weights of the sparse weights $\boldsymbol{\alpha}$ and drives them to be zero, the parameter λ acts on the distribution similar to the diversity parameter. When the value of λ is large, there is very little diversity in the distribution thus leading to a more sparse solution, when the value of λ is small, the diversity of the distribution becomes much larger leading to a far less sparse solution. Typical values for this parameter are values that range from 0.01 up to around 0.15. However, the value of this parameter can be outside these bound as the data itself sometimes requires different values.

The next parameter to be discussed is β, which affects how many data points are assigned a high probability of being a target point. In the *e*FUMI and MT-*e*FUMI algorithms which this method is based on the affect of β on the probability could be shown graphically as in Figure 3.1. Larger β's lead to more points having a high probability of target, while lower values had the opposite effect. In the proposed model the graphical depiction becomes more

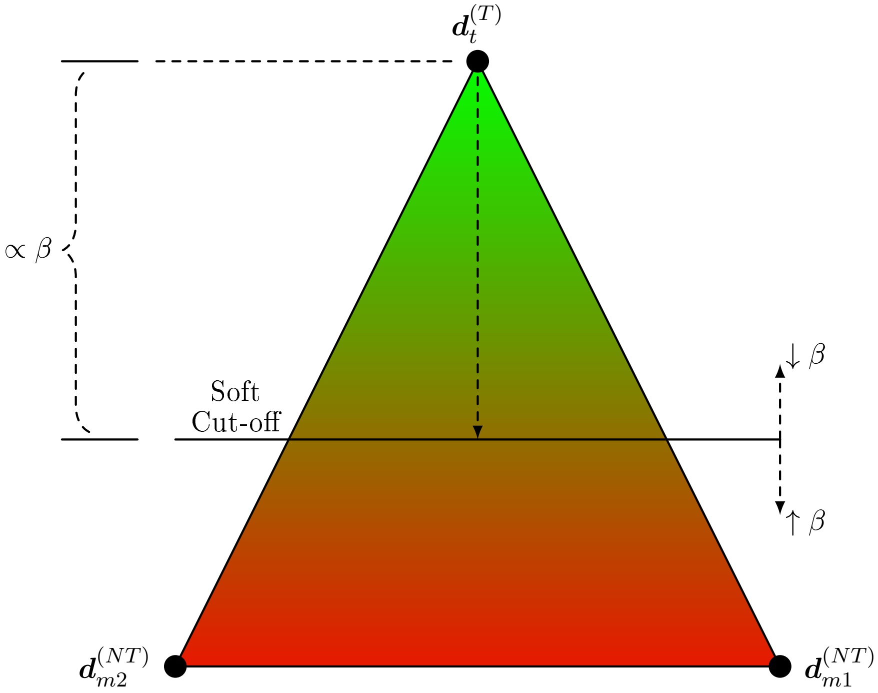


Figure 3.1: Graphical representation of how parameter β affects the algorithm. A dictionary of size three, one target and two non-targets, is used for this representation in order to more easily show the effect. The actual data points are then scattered inside the triangle as convex combinations of the three dictionary elements if the sparse weights are assumed to be positive and sum to one. The green region indicates the region receiving a high probability of target while the red indicates the region receiving a low probability.

abstract as the sparse weights no longer sparse convex combinations of the dictionary atoms. However the diagram is still accurate in how the probabilities are assigned in the proposed model. In principal β indicates how much target has to be present in the sample for it to have a high probability of target, even if the proportion is negative or even greater than one. The typical range of values for this parameter is quite large as any number can be used here but a good range would be values between 1 and 500. However for the testing later in this thesis the values chosen were around 5.

ϵ is a fine tuning parameter that affects the importance of the positive bags. This parameter directly affects the function $\delta = \epsilon\frac{N_B}{N_T}$ when the sample being trained is from a positive bag, it has no affect on points from negative bags. A large value of ϵ will make the points from the positive bags much more important towards the training of the proposed method by adding a multiplier to these data points. Typical values for this parameter would be in the range of 0.75 and 1.25.

Now moving on the the learning rate parameters i_0 and ρ. The effect of these parameters on the algorithms is that they determine how fast the dictionary and model parameters will change. ρ is the typical learning rate and directly affects the size of the gradient step. The second parameter i_0 is used with the learning rate to help the algorithm converge to a steady state solution by reducing the learning rate inversely proportional to the iteration. When used in conjunction the learning rate for any give iterate is chosen to be $\min(\rho, \frac{\rho i_0}{i})$ where i its the current iteration. Typical values for these parameters are hard to establish because it depends heavily on the amount of data used to train the algorithm. For the testing done in the later sections learning rates of 0.25 were used with great success. The second parameter is not as important and as a general rule of thumb i_0 is set to be around 10% of the total number of iterations.

u is the parameter that controls the tradeoff between the classification accuracy and the distance from the dictionary atoms to the global data mean. A large value of u will push all the dictionary atoms to be very close to the global data mean while a small value of u will force the algorithm to learn a dictionary and model parameter that leads to much better classification. Typical values for u would be between 0.0001 and 0.05as the classification error is much more important.

v is another parameter that is very similar to u, it controls how large the value of the model parameter $\boldsymbol{w}$ can be by constantly pushing the vector to have an L_2 norm of close to zero. Larger values of v will push the model parameter to be closer to zero. Typical values for v are between 0.001 and 0.1.

s is another tuning parameter that controls the effect of the smoothness term on the learned dictionary. Larger values of s will force a much smoother dictionary. However, when this parameter is set too large this term will force the dictionary elements to be a simple line. Therefore, if the smoothing term is desired typical ranges for its values are between 0.001 and 0.2. If the smoothing term is not desired set the value of s to equal zero.

Next the parameters T and B which control the size of the dictionary. The dictionary size in the standard dictionary learning problem, which in the proposed method is $T + B$, is what determines how many basic elements the dictionary should find in the training data. A larger dictionary size means that the dictionary will contain more elements and be able to decompose more signals, when the other parameters are tuned correctly, but this also means that it will take require more data and time to train as the model becomes much more complex. In the proposed method T and B have the same effect as the dictionary size does in the standard model, except that the effect is broken down into two groups target and non-target. Everything else from the standard dictionary size model is carried over except now the model allows for finer control over how the dictionary atoms are split between target and non-target. Typical values for the dictionary size are very difficult to establish. Both parameters can be estimated by looking at the expected number of target and non-target signatures expected in the data.

Finally the batch size which controls how many data points are processed at the same time. This parameter comes from the stochastic gradient descent algorithm used in the proposed method where where the updates are computed for only a single data point or for a small batch of data points. In the true stochastic gradient descent algorithm the update is computed after each point is processed, however this can lead to very erratic changes in the dictionary and the model parameters. So to smooth out these erratic jumps a small number of data points are used to compute the updates, this lead the algorithm to converge quicker while also still maintaining the stochasticity of the training algorithm. Typical values for this parameter are between 1 and 500, this essentially tunes how smoothly the algorithm will converge but does not necessarily effect the final results.

Chapter 4

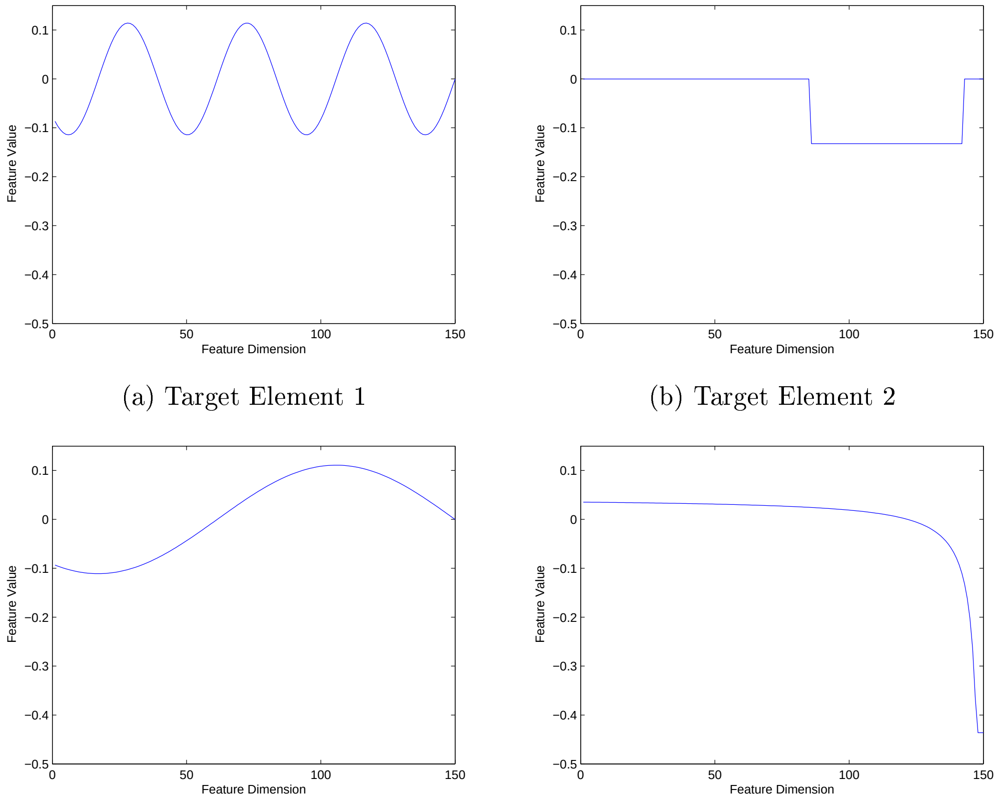
Experiments

In this section several experiments are conducted to look at the performance of the proposed algorithm. In particular three different types of data will be used throughout this chapter. The first is a set of synthetic data which can be used to demonstrate the effectiveness of the proposed method. The second type of data is wideband electromagnetic induction data from a metal detector used for landmine detection. The final data type is a collection of sonar images used for testing target detection and classification algorithms.

The proposed method will be compared with MT-eFUMI and TDDL. There are two main tests that are usually used for the testing of dictionary learning algorithms, reconstruction and classification. Typically, reconstruction error is used for the unsupervised methods as many of the supervised dictionary learning, such as the proposed method, will sacrifice some reconstruction ability to attain a dictionary that can better classify the data. As the main goal of this algorithm is increasing the classification performance that will be the main focal point for performance comparisons between the proposed algorithm and the two comparison algorithms.

4.1 Synthetic Data

Synthetic data is chosen here to demonstrate the benefits of the proposed method versus the two methods it was based on. The synthetic data gives the ability to test the limits of the proposed algorithm compared with the two other algorithms that it was based off of. Since the way the data is generated can easily be controlled it also provides the ability to test the limits of the algorithms and see how their performance deteriorates as the data becomes more difficult.



(a) Target Element 1

(b) Target Element 2

(c) Background Element 1

(d) Background Element 2

Figure 4.1: Actual pure dictionary atoms that are used to create the synthetic data.

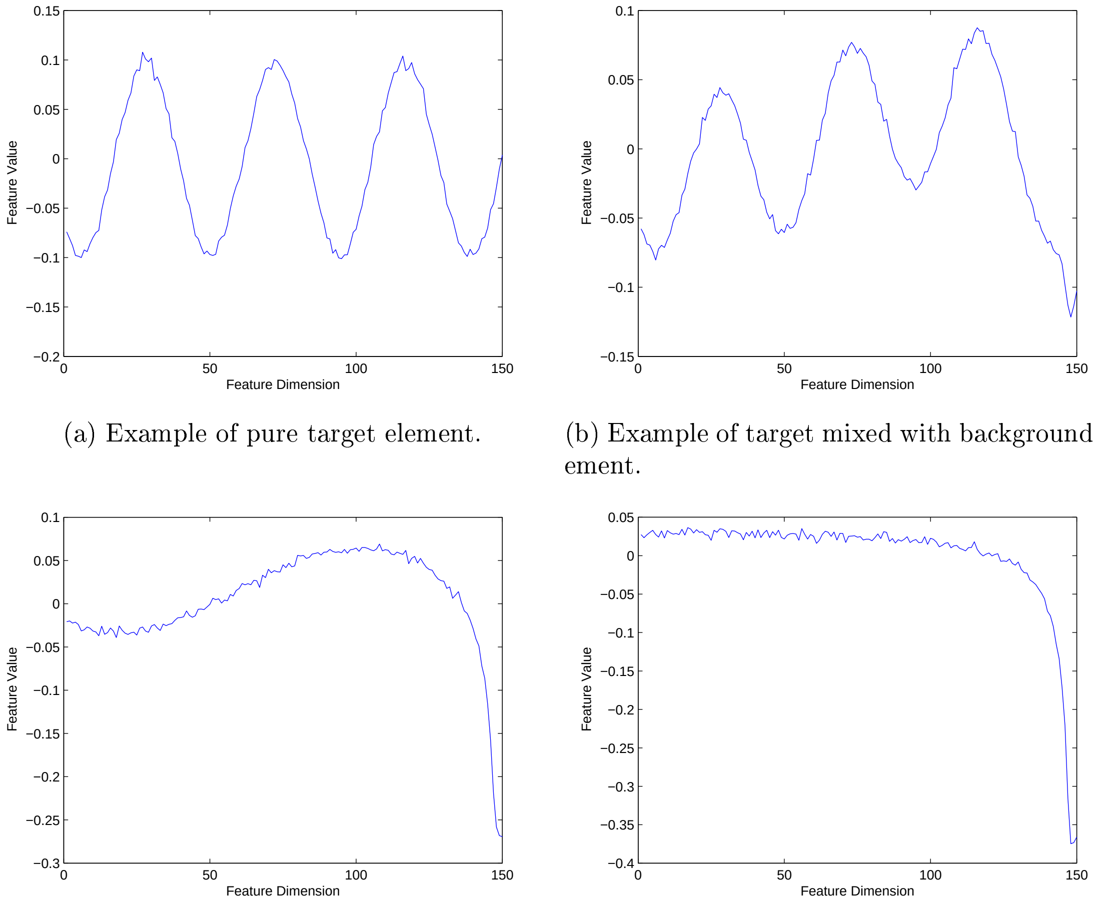
4.1.1 Data Set Description

The synthetic data that was generated for this experiment was created by using four distinct elements shown in Figure 4.1. The data itself is simulated by randomly combining these elements in different ways to generate a full set of data. To do this random sparse weights were created and then the data would be generated by multiplying the actual dictionary by the randomly created sparse weights and then adding noise. The full algorithm used to generate the data is shown below.

Algorithm 2 Algorithm for generating the synthetic data used in the experiments in this section.

```
Require: Actual dictionary $\boldsymbol{D}_a$, Signal to Noise Ratio $snr$, Number of Target Points $N_t$, Number of Non-Target Points $N_b$, and Target Proportion $P_t$
Get number of target, t, and non-target, b, dictionary elements
Generate $N_t$, t dimensional sparse random weight vectors
Scale target sparse weights and non-target sparse weights so that
    $$P_t = \frac{\sum(targetweights)}{\sum(non - targetweights)}$$
Generate $N_b$, b dimensional sparse random weight vectors
Concatenate sparse weights for target and non-target add zeros to non-target sparse weights to match size of the target weights
Compute clean data via $\boldsymbol{X} = \boldsymbol{D}_a\boldsymbol{A}$
Generate Gaussian noise
Add noise to match requested $snr$
return Data ($\boldsymbol{X}$) and actual sparse weights ($\boldsymbol{A}$)
```

This structure is ideal for testing dictionary learning algorithms because the learned dictionaries can be directly compared with the originals giving an excellent method for comparing different dictionary learning algorithms. A few examples of the training data are shown in Figure 4.2.



(a) Example of pure target element.

(b) Example of target mixed with background element.

(c) Example of multiple background elements being mixed.

(d) Example of a pure background element.

Figure 4.2: Examples of synthetic data created for testing proposed method.

4.1.2 Testing the Sparse Coder

The first step in using the synthetic data is to make sure that the sparse coder is functioning properly. This is done with the synthetic data since the dictionary and the sparse combinations of the dictionary that created the data are known and can be compared with directly.

The first test that is performed is a check to see if the sparse coder, MT-*e*FUMI in this case, can accurately pick the correct dictionary elements that were used to create the signal if it is given the actual dictionary. The results from this test are shown in Table 4.1. The results shown in the table are from 100,000 randomly generated signals from the synthetic dictionary shown in Figure 4.1, random sparse weights were created then the data points were created for the sparse weights and noise was added to the signals such that each sample had the same signal to noise ration (SNR). The noise is from a Gaussian distribution with unit variance and zero mean.

Table 4.1: Test results from the sparse coder testing, test was performed using 100,000 randomly generated sparse weights and the dictionary shown in Figure 4.1.

	Error in Sparse Weights **Sum Squared Error(SSE)**		
SNR of Data	∞	25dB	5dB
Average Error	6.6953×10^{-5}	6.7541×10^{-5}	4.9569×10^{-4}
Variance	9.8993×10^{-8}	9.7042×10^{-8}	2.0414×10^{-6}

This test shows that given a fixed dictionary and signals created from that dictionary the MT-*e*FUMI sparse coder can generate a very accurate estimate of the actual sparse weight that was used to generate the signal even in the presence of noise.

The next part of the proposed implementation that needs to be tested is the ability of the algorithm to estimate the probability that a given point contains a target signature. To test this another round of data was generated just as before, then the actual sparse weights and dictionary elements were used along with Equation 2.20 to estimate the probability of target. The results from this experiment are that the algorithm can correctly identify 100% of the data points that contain any portion of the target dictionary elements. This result is to be expected as this portion of the proposed algorithm is taken directly from the *e*FUMI algorithm.

The next tests to be performed on the new method using the synthetic data are test that determine how well it performs when presented with deteriorating data. In all of the following test the proposed method will be compared with the two algorithms that it was based on to shown the improvements it makes on them.

4.1.3 Breakdown Testing

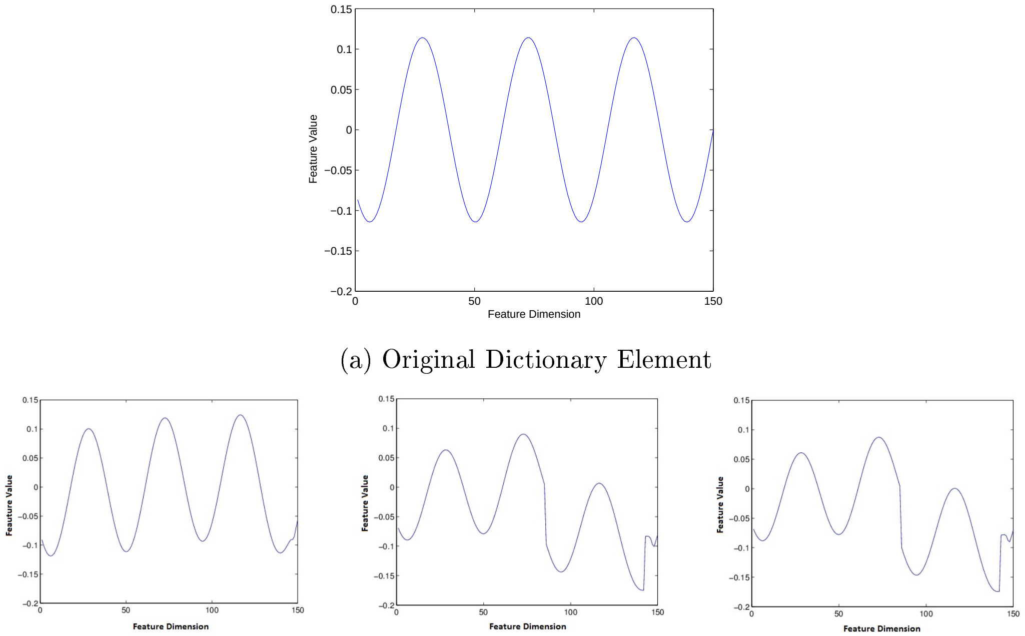
In this section experiment will be performed that test how well the proposed algorithm performs when the training data gets progressively worse. All of the results presented in this section were found by randomly generating data under the conditions described in each test five times then using random initialization for each of the five test for all of the algorithms. All the results shown in the following tables are then the average over these five trials along with the standard deviation over the five trials.

The first test will be to generate data and change the number of negative points in the positive bags to determine how well the algorithm can throw away non-target points in the positive bags. The second test will be to generate synthetic data with varying proportions of backgrounds to see if the algorithm can still distinguish the targets from the background.

For the remaining tests using the synthetic data 1000 data points were generated using the dictionary shown in Figure 4.1, each target point will consist of only one target class and any number of background elements. The data will be labeled so that exactly half is target, 500 target and 500 non-target, the number of actual targets in the target class is held fixed at 85%, unless otherwise stated. The target/non-target ratio is fixed at 85%, again unless otherwise stated.

The parameters used for the TD-eFUMI algorithm are $\lambda = 0.1$, $\rho = 0.5$, $i_0 = 5$, $T = 2$, $M = 2$, $\epsilon = 1$, $v = 0.001$, $u = 0.0001$, $\beta = 5$, and a batch size of 50. The smoothness term was not used in this data, as such the parameter s was set to zero. For the TDDL algorithm the parameters are $\lambda_1 = 0.0001$, $\lambda_2 = 0$, $\rho = 0.25$, $K = 4$, and a batch size of 50. For the MT-eFUMI algorithm the parameters used were mimicked from the TD-eFUMI algorithm. These values were determined through multiple tests of the algorithms.

In the first test the percentage of actual target points labeled as target is varied through the following values; 100%, 85%, 70%, 50%, 25%. The results will then be compared based on how well the algorithm can classify the positive bag according to the actual class labels.



(a) Original Dictionary Element

(b) 100% target

(c) 70% target

(d) 50% target

Figure 4.3: Visual breakdown of the learned dictionary with varying levels of actual targets present in the positive bags. Value under images shows percentage of target points in positive bags.

Figure 4.3 shows a dictionary element that was learned by the proposed algorithm under three of the different conditions tested for the amount of target points in the positive bag. The top image is the original dictionary element that was used to create the data. In the learned dictionary elements shown in the bottom of the figure it is clear that the number of target points does affect the dictionary element that is learned but it is also clear that the algorithm is still clearly able to learn the main structure of the original element.

Now moving on to the classification performance of the proposed method given the above conditions. The classification results are shown in Table 4.2. These results still mirror what was expected when looking at the dictionary elements that were learned. Even though the dictionary elements do degrade as their are fewer and fewer target points in the positive bags since the algorithm can still distinguish between the target and non-target points the classification is not affected until the background points far outweigh the target points.

Table 4.2: Classification performance for varying levels of target points in positive bags. The first value in each table cell is the average percentage of points correctly classified the plus minus symbol then denotes the standard deviation of the results.

	TD-*e*FUMI	**TDDL**	**MT-*e*FUMI**
100%	100 ± 0	100 ± 0	61.24 ± 3.62
85%	100 ± 0	100 ± 0	61.12 ± 1.86
70%	100 ± 0	100 ± 0	60.88 ± 1.63
50%	96.16 ± 5.28	91.18 ± 13.78	43.8 ± 6.88
25%	90.26 ± 7.07	76.26 ± 22.05	26.36 ± 9.63

This is the experiment where the multiple instance portion of the proposed method shines particularly well. Over the first three target bag configuration the performance of TDDL and the proposed TD-*e*FUMI algorithms mirror each other at 100% correct classification. However, when the number of non-target points labeled as targets are equal the proposed algorithm is able to maintain its classification performance above 90% even when only one quarter of the points labeled as target are actually target points. The TDDL algorithm does well when the ratio of target points to non-target points in the positive bag is equal as it still maintains a correct classification percentage of greater than 90% but at the next step the algorithm's performance falls off greatly when compared to TD-*e*FUMI.

With this experiment it is clear that when the data labels have a low level of uncertainty the proposed method will mirror the results of TDDL, which it is primarily based on. This is expected because if, in the proposed model, the probability estimation is assumed to be correct all of the time the proposed model essentially reduces to the original TDDL algorithm. But, when the data does begin to degrade by having more and more uncertain labels the proposed algorithm can differentiate between the points that are labeled target and are actually target while the TDDL algorithm cannot.

For the second part of the test the percentage of actual target points in the positive bags will be fixed at 85% and the proportion of target to background membership will be varied at the following levels; 95%, 85%, 70%, 50%, 25%. Again the results will then be compared based on how well the algorithm can classify the positive bag according to the actual class labels. For this test the weights that were used to create the data were created so that they summed to one so that the target proportion could be easily controlled.

Table 4.3: Classification performance for varying levels of target proportions. The first value in each table cell is the average percentage of points correctly classified the plus minus symbol then denotes the standard deviation of the results.

	TD-*e*FUMI	**TDDL**	**MT-*e*FUMI**
95%	100 ± 0	100 ± 0	59.20 ± 1.76
85%	100 ± 0	100 ± 0	62.54 ± 4.19
70%	100 ± 0	100 ± 0	64.1 ± 1.67
50%	94.96 ± 8.66	100 ± 0	48.3 ± 6.15
25%	81.76 ± 8.44	58.02 ± 10.27	45.34 ± 2.17

The classification results are shown in Table 4.3. The results are similar to those see in the previous test, initially both TDDL and the proposed method behave similarly. However there comes a point that TDDL performs poorly while the proposed method still manages decent performance. In this case TDDL can correctly classify 100% of the data until the ratio of target to non-target in the positive bag becomes less than 50%. The proposed algorithm correctly classifies 100% of the data until this ratio is 70% while at 50% it only manages 95% correct classification, slightly down from the TDDL algorithm. However at the last stage the proposed algorithm stays much closer to its original value while TDDL's classification performance falls much faster.

Here again the multiple instance learning that is introduced into the TDDL method by the proposed algorithm has allowed it to maintain decent classification performance even when the targets are mostly composed of non-target elements. In this experiment, the multiple instance learning helps to give a better estimation of the non-target elements since they have specific dictionary elements. While in the TDDL model the dictionary atoms are not assigned to be target or non-target and all atoms are shared. This difference allows

TD-eFUMI to get a much cleaner picture of what the background looks like, allowing for more possibility of the difference in the target elements to be seen and learned.

One final test using the synthetic data was to modify the original dictionary that was used to generate the data so that one of the background elements would look similar to the target elements. In this test, the same dictionary as before is used except that the fourth element is now modified so that it is a linear combination of the previous fourth element and the first target dictionary element. Then a mixing term is swept through several values so that the background element was eventually the same as the target element. Again with this test the classification performance of the algorithm are the only result that is being compared. The results for this last test are shown in Table 4.4.

The percentages shown at the top of Table 4.4 indicate how mixed the second target dictionary is. For example a value of 100% indicates that the second target dictionary atom is pure, i.e. it looks identical to the one shown in Figure 4.1b. When the value is 50% it indicates that the second dictionary atom in the dictionary used to make the data is a linear combination of dictionary atoms one and four with a 50-50 distribution. Then lastly a value of 0% indicates that the second target atom in the actual dictionary looks identical to actual background element four.

Table 4.4: Classification performance for varying levels of similarity between one target dictionary element and one non-target dictionary element. The first column shows the percentage of actual background element in the described true dictionary element. The first value in each table cell is the average percentage of points correctly classified the plus minus symbol then denotes the standard deviation of the results.

	TD-eFUMI	**TDDL**	**MT-eFUMI**
100%	100 ± 0	100 ± 0	60.70 ± 1.45
75%	100 ± 0	100 ± 0	74.88 ± 14.37
50%	98.90 ± 2.46	90.52 ± 13.00	71.22 ± 20.96
25%	93.48 ± 11.05	89.22 ± 14.85	54.46 ± 20.29
0%	78.12 ± 1.18	69.28 ± 1.11	49.42 ± 6.87

In this test both TDDL and TD-eFUMI do slightly worse than in previous tests, but his test is also much more difficult, as then only manage to maintain 100% correct classification for the first two stages of the test. Again in this test though, the proposed method is able to hold on longer than TDDL to the better classification performance.

In this experiment the most likely reason that the proposed algorithm does better when the actual target and non-target atoms are mixed is that the dictionary atoms have assigned classes for each of the learned dictionary atoms. This allows the proposed method to learn a much cleaner background than the TDDL method, as stated before, thus making the proposed method more suitable for detecting smaller differences between a point that contains both target and non-target elements.

These results have shown that the proposed algorithm can correctly identify target points from non-target points even when both are labeled as target.

4.2 Metal Detector Data

The first set of real-world data that the proposed method will be implemented on is data collected from a handheld wideband electromagnetic induction system (WEMI).

4.2.1 Data Set Description

The sensor utilizes WEMI [47] to detect metal objects underground. In particular, the sensor used to collect the data used for testing records a response over twenty one frequencies. The sensor is swept over a lane that contains several mine-like objects and other items. The dataset that is being used here consists of six of these lanes collected from the same location. The content of the lanes are shown in Table 4.5. The table breaks the objects into five types, metal targets, low-metal targets, non-metal targets, metallic clutter and non-metallic clutter as these are the main target types in the data.

The frequency response that is measured by the receiver is then a twenty one dimensional

Table 4.5: Object description for metal detector dataset. Abbreviations, MT: Metal Target, LMT: Low-Metal Target, NMT: Non-Metal Target, CL: Clutter.

	MT	**LMT**	**NMT**	**CL**
Lane 1	4	7	0	6
Lane 2	4	10	0	4
Lane 3	4	7	0	8
Lane 4	6	6	3	0
Lane 5	7	5	5	0
Lane 6	6	6	2	3
Totals	31	41	10	11

complex vector for each data point. In order to make the vector real and still keep the imaginary part of the vector intact, the imaginary portion of the vector is moved to the tail end of the real portion to create a new forty two dimensional real vector.

For this test the proposed algorithm and the two comparison algorithms will be run as classifiers on this data. The classifiers therefore need specific points in the data to train on. For this a prescreener is used to find points of interest in the data for the classifiers to train on. The prescreener that the algorithms will use is the Joint Orthogonal Matching Pursuits (JOMP) algorithm described in [1]. This algorithm was briefly described in Section 2 but as a review, it is another dictionary based algorithm that used a fixed dictionary that models what metallic objects look like in the data. The data is then compared to the dictionary and if they are similar to any of the target elements the point is assigned a high confidence. If the data is closest to the background element the confidence is then set to zero. The number of alarms from each lane and the totals are shown in Table 4.6.

Then to generate the actual alarms, mean shift [48] was used to cluster the confidence maps to find the locations where the JOMP algorithm said there were targets. From each of these points a fixed radius is drawn around the point and all data within the radius is used to generate the alarms.

Table 4.6: Distribution of alarms found by the JOMP prescreener over the six testing lanes.

	Target Alarms	**False Alarms**	**Total Alarms**
Lane 1	20	29	49
Lane 2	19	22	41
Lane 3	18	20	38
Lane 4	14	17	31
Lane 5	19	13	32
Lane 6	19	8	27
Totals	109	109	218

4.2.2 Classification

For testing lane based cross validation is used, which means that six fold cross validation is used. In this way the alarms from one lane are held out from the training data to be used as the test set, while the alarms from the remaining five lanes are combined to generate the training data.

For classification each classifier was trained in a binary fashion to be able to classify points as either target or non-target. The best way to then compare the algorithms is to then look at the receiver operator characteristic (ROC) curve for each of the algorithms. The parameters used for the TD-*e*FUMI algorithm are $\lambda = 0.12$, $\rho = 0.4$, $i_0 = 3$, $T = 6$, $B = 2$, $\epsilon = 1$, $v = 0.001$, $u = 0.0001$, $\beta = 5$, and a batch size of 500. These values were determined through multiple tests of the algorithm.

The smoothness term was used in the testing of the TD-*e*FUMI algorithm on this data, the parameter s was set to 0.1. The reason that the term was used here was that the non-target data for the WEMI data has a very large amount of noise, masking the structure in many cases. Therefore the algorithm would introduce some of this noise into the dictionary elements instead of maintaining smooth dictionary elements. Upon adding this term, the proposed method was again able to learn smooth dictionary elements and perform well in the classification tasks.

The parameters used for the TDDL algorithm are $\lambda_1 = 0.001$, $\lambda_2 = 0$, $\rho = 0.25$, $K = 7$,

Table 4.7: Similarity table for each of the six folds in the cross validation training. The Dictionary elements are sorted so that the most similar elements are being compared. The actual measure shown in the table is the mean squared error between each of the dictionaries.

	Fold 1	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**
Fold 1	0	0.0529	0.1535	0.1907	0.2008	0.2695
Fold 2		0	0.1333	0.1903	0.2059	0.2245
Fold 3			0	0.0617	0.1543	0.1182
Fold 4				0	0.1294	0.1125
Fold 5					0	0.1830
Fold 6						0

and a batch size of 50. The parameters used for the MT-*e*FUMI algorithm are again a subset of the TD-*e*FUMI parameters.

The differences between the dictionaries learned by the proposed method during the cross validation training are shown in Table 4.7. All of the dictionaries are sorted so the elements are matched with their closest match in the other dictionaries. The process for sorting the dictionary elements is a greedy process. The process to sort the dictionaries to match one another involves picking an initial dictionary to sort the others to match. Then for each of the other dictionaries the first element in the initial dictionary is compared to all of the dictionary elements of the unsorted dictionaries, the dictionary element from each unsorted dictionary that best matches, according to euclidean distance, is moved to match the position of the initial dictionary element. The element is then removed from the unsorted dictionaries and this process is repeated until all of the dictionary elements have been sorted.

The mean squared error is computed between all of the matched dictionary elements in the two dictionaries being compared. The mean squared error leaves eight individual errors which are then summed together to generate the final similarity measure, lower values are better. The lower left half of the table is left empty as it is just a mirror image of the top right.

Table 4.7 shows that the dictionaries learned across the separate folds are fairly consistent which would seem to indicate that the dictionary learning aspect of the proposed algorithm is generalizable to data collected in similar formats.

The first test to look at is whether or not the proposed classifier can improve upon the prescreeners detection. The two ROC curves for both JOMP and the TD-eFUMI classifier are shown in Figure 4.4.

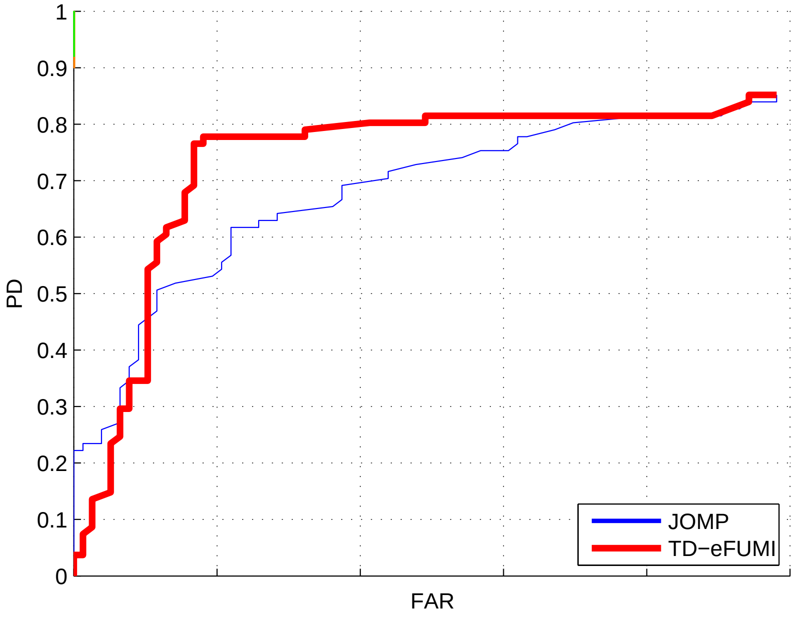


Figure 4.4: ROC curve comparing the classification performance of TD-eFUMI with the prescreener that was used to generate alarms.

In the ROC curve it can be seen that the JOMP prescreener starts off with a detection rate of around 20% and then slowly increases until it reaches its maximum detection rate of just over 80%. The classifier is then ran and should improve upon the results of the prescreener. When looking at the ROC curve for TD-eFUMI, the ROC curve reaches up to just shy of 80% very quickly. This test shows that the proposed classifier can improve the performance of our detector, since we can move the peak detection rate much further to the left on the ROC curve.

Now moving on to a direct comparison of the three dictionary learning methods that

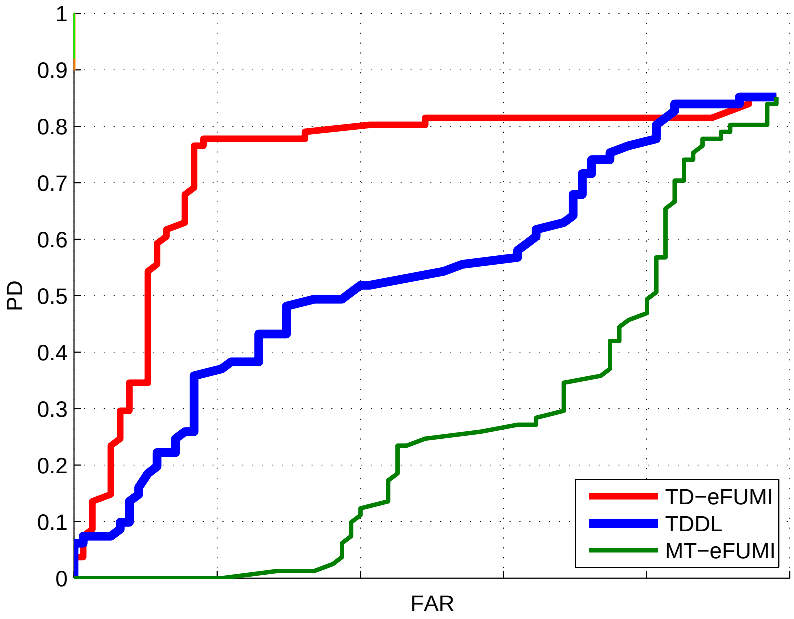


Figure 4.5: ROC curve for TD-*e*FUMI and other comparison algorithms using the metal detector data.

have been used and discussed, as comparison. The ROC curves for all of these algorithms are now shown in Figure 4.5. In this figure it can be seen that the proposed classifier does much better than either of the two classifiers.

MT-*e*FUMI is the worst performer of the group of algorithms, the reasoning for this could be that MT-*e*FUMI by itself is not a classification algorithm. Therefore, MT-*e*FUMI needs a method to use its dictionary to decide if the current point is a target point.

The next step up on the performance list is the TDDL algorithm this algorithm does do much better than the MT-*e*FUMI algorithm. Again this algorithm does include a method to directly use the dictionary for classification purposes. Having this classifier built directly into the algorithm obviously is an advantage as TDDL has outperformed MT-*e*FUMI in every test so far.

However, both TDDL and MT-*e*FUMI fail in comparison to the proposed algorithm on this dataset. This is because the proposed method takes the advantages of both algorithms

Table 4.8: Distribution of alarms returned by RX detector on sonar data set.

Object Name	**Number of Alarms**
Non-Target	2223
Block	125
Cone	56
Sphere	66
Torus	33
Pipe	146
Cylinder	667

and combines them, it has the built-in classifier from the TDDL algorithm and it takes the multiple instance learning portion from MT-*e*FUMI to be able to handle uncertainties in the labeling of training data.

4.3 Sonar Data

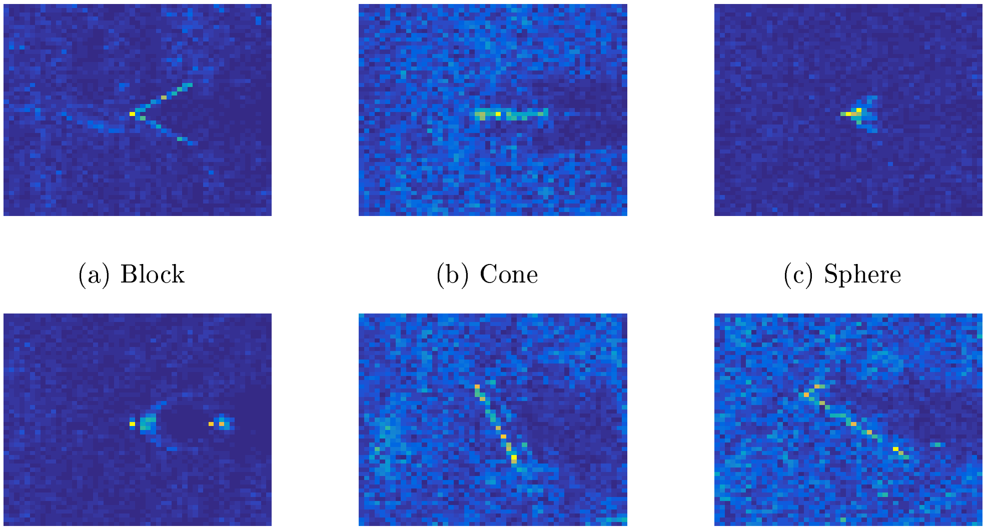
The second real data set to be used will be from a synthetic aperture sonar sensor. For this test TD-*e*FUMI is compared with TDDL and MT-*e*FUMI to demonstrate the benefits of combining the two methods has over the two individual methods.

4.3.1 Data Set Description

The data used in the following two experiments consists of 143 sonar images of various seafloor environments and objects added to the scenes. In this set there are 13 different target configurations and 11 different underwater scenes. In each of the images there appear only 4 targets consisting of a subset of block, cone, sphere, torus, pipe, or cylinder.

The images are originally 1800 $\times$ 3984 but are cut down to 1800 $\times$ 3310 to remove artifacts from the sonar module. An RX detector [49] was implemented and ran on the entire image data set generating 3315 alarms. The distribution of these alarms are shown in Table 4.8. Figure 4.6 shows a selection of the alarms returned from the RX detector.

In Figure 4.6, in the examples of the alarms it is possible to see the difficulty in correctly classifying all of the targets. Many of these targets are very similar to others in the data.



(a) Block

(b) Cone

(c) Sphere

(d) Torus

(e) Pipe

(f) Cylinder

Figure 4.6: Example alarms from the RX detector output on the sonar images

.

For the dictionary learning algorithms the image was quantized by re-sampling the image and keeping every fifth pixel in both the horizontal and vertical dimensions. From these re-sampled images a patch of size 51 × 51 was extracted from around the alarm location, this patch was then vectorized to form the new data vector.

4.3.2 Classification Results

Moving on to the classification on the sonar data. For this test the data set is split into two sections to be used for cross validation testing, two fold cross validation. One set was used for training while the other was used to test, then the roles of the two sets were reversed and the results were then combined to create the full results presented in the rest of this section. Each classifier was trained in a one versus all manner, so that seven separate dictionaries were trained for each fold in the cross validation.

The parameters used for the TD-eFUMI algorithm are $\lambda = 0.1$, $\rho = 0.5$, $i_0 = 7$, $T = 3$, $B = 6$, $\epsilon = 1$, $v = 0.001$, $u = 0.0001$, $\beta = 5$, and a batch size of 25. The smoothness term is not used for this data. For the TDDL algorithm $\lambda_1 = 0.001$, $\lambda_2 = 0$, $\rho = 0.25$, $K = 9$, and a batch size of 50. The setting for MT-eFUMI algorithm were again taken directly from the TD-eFUMI settings. These values were determined through multiple tests of the algorithm.

An example of one of these dictionaries for each of the three methods are shown in Figures 4.7, 4.8, 4.9.

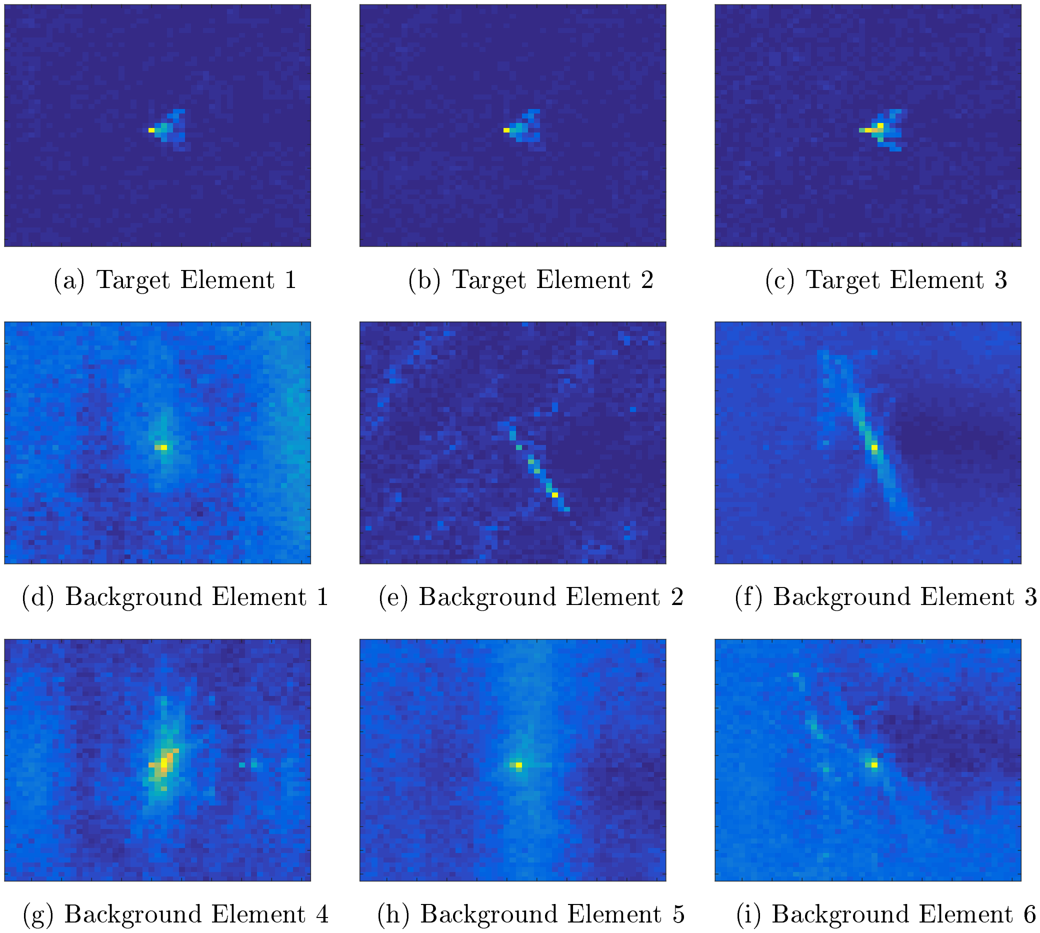
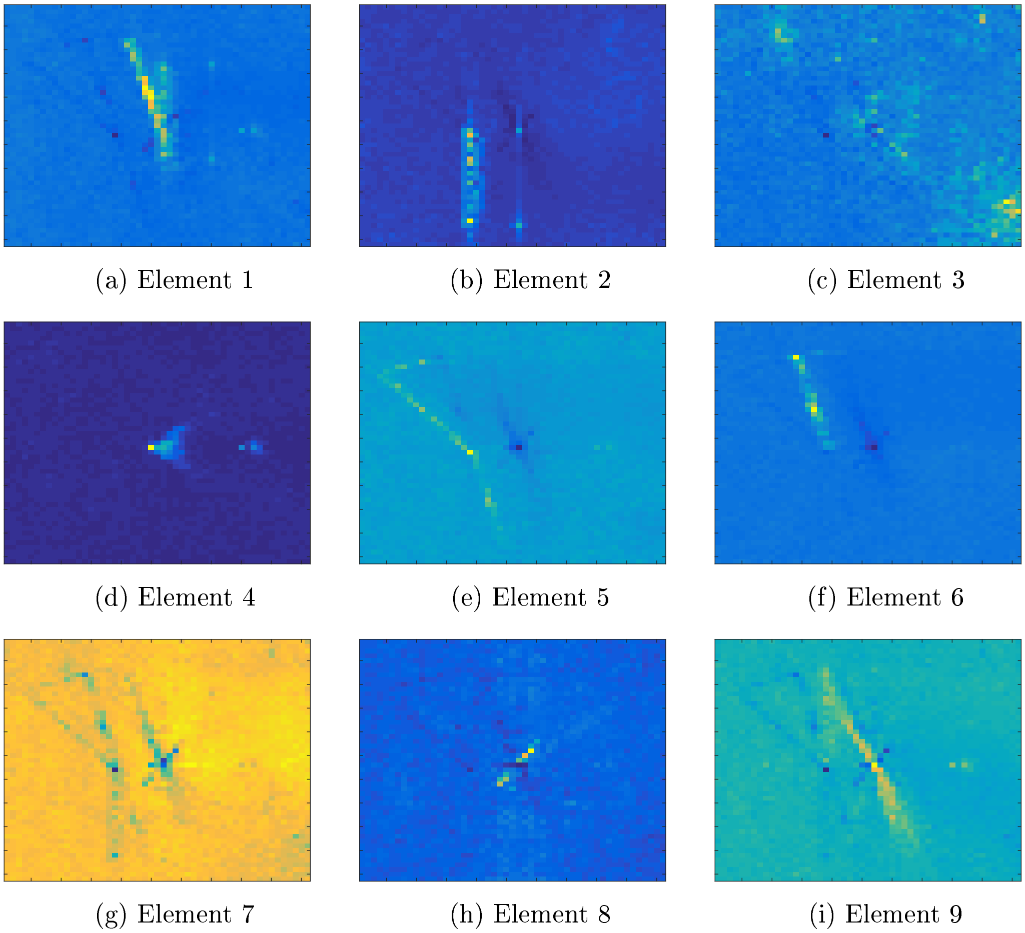

(a) Target Element 1
(b) Target Element 2
(c) Target Element 3
(d) Background Element 1
(e) Background Element 2
(f) Background Element 3
(g) Background Element 4
(h) Background Element 5
(i) Background Element 6

Figure 4.7: Example dictionary learned by TD-eFUMI on the sonar data. Dictionary is for the case when the sphere is the target class.


(a) Element 1

(b) Element 2

(c) Element 3

(d) Element 4

(e) Element 5

(f) Element 6

(g) Element 7

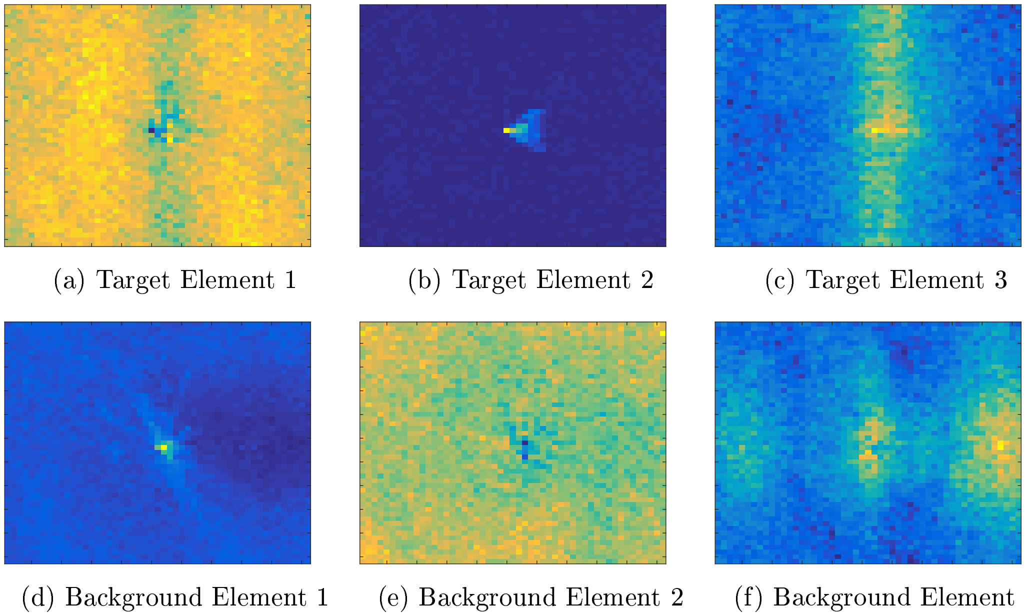
(h) Element 8

(i) Element 9

Figure 4.8: Example dictionary learned by TDDL on the sonar data. Dictionary is for the case when the sphere is the target class.

The dictionary size chosen for this test was nine total elements, three of which were designated as target while the remaining six were for the non-target class. The dictionary shown here shows that the data for the the sphere class is very consistent because the three target dictionary elements are nearly identical. The non-target dictionary elements often show a single high intensity value at the center of the patch this is probably due to the fact that nearly all of the alarms returned by the RX detector are centered on the highest intensity points of the targets.

Another interesting observation about the non-target dictionary is that many of the ele-



(a) Target Element 1

(b) Target Element 2

(c) Target Element 3

(d) Background Element 1

(e) Background Element 2

(f) Background Element 3

Figure 4.9: Example dictionary learned by MT-eFUMI on the sonar data. Dictionary is for the case when the sphere is the target class. There are only six dictionary elements in this case because in the implementation of MT-eFUMI used here the algorithm is able to discard dictionary elements it deems an unnecessary.

ments contain lines through the center of the patches. It is likely that these lines correspond to the different orientations of the other targets in the data.

The proposed algorithm is compared to a kNN classifier that also takes advantage of environmental context as described in the technical report in [50]. The results from this classifier are shown in Table 4.9, this method achieves an overall accuracy of 86.48%.

The results from the proposed method are shown in Table 4.10, The overall accuracy of the proposed method is 87.13%.

The two methods perform nearly identically as far as the overall performance goes, however the kNN classifier also uses much more information about the environment which then allows it have different classifiers based on where the targets are located. The proposed method achieves the same results without using this additional information. This contextual information leads to approximately a 10% increase in performance for the kNN classifier.

Table 4.9: Confusion matrix for the previous *k*NN classifier for the sonar alarms. Total classification accuracy is 86.48%.

Class	**Non-Target**	**Block**	**Cone**	**Sphere**	**Torus**	**Pipe**	**Cylinder**
Non-Target	**94.49**	0.44	0.58	0	0.488	0	4.40
Block	1.6	**48.80**	0.80	2.40	1.60	0	44.80
Cone	19.64	0	**73.21**	1.79	0	0	5.34
Sphere	0	0	0	**95.46**	0	0	4.55
Torus	33.33	0	0	0	**66.67**	0	0
Pipe	6.45	0	0	0	0	**0**	93.55
Cylinder	11.37	8.18	0	1.11	0.14	2.64	**76.56**

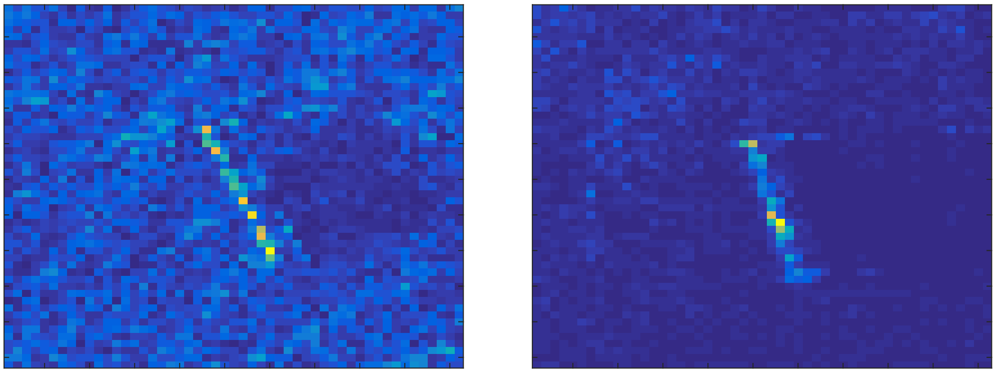
Table 4.10: Confusion matrix for the proposed TD-*e*FUMI classifier. Total Classification accuracy is 87.13%.

Class	**Non-Target**	**Block**	**Cone**	**Sphere**	**Torus**	**Pipe**	**Cylinder**
Non-Target	**98.75**	0	0.46	0	0	0	0.79
Block	16.13	**16.13**	1.61	3.23	0	7.26	55.65
Cone	20	0	**56.36**	0	0	0	23.64
Sphere	13.64	1.52	0	**83.33**	0	0	1.52
Torus	0	0	22.58	45.16	**32.26**	0	0
Pipe	2.22	0	0	0	0	**91.11**	6.67
Cylinder	15.26	0.73	0.73	5.52	0	14.34	**63.42**

The proposed method has the advantage that it no longer needs the contextual information to achieve better results.

Now looking into the individual class classification results. The first class to look at is the non-target class where the kNN classifier achieved its second highest accuracy at 94.49% and the proposed TD-eFUMI classifier achieved it's highest accuracy at 98.75%. This also happens to be the largest and therefor most important class to get correct in the data set. The proposed algorithm does much better than the kNN classifier in this case probably doe to the more even distribution of the data set in this case. Since the non-target class is roughly two-thirds of the total dataset the proposed algorithm is much more capable of learning a dictionary to classify this class very well while sacrificing some ability to classify the remaining points.

Another class that the proposed methods stands out in is the Pipe class where the kNN classifier could not correctly identify any and the proposed method correctly classifies 91.11%. This results is somewhat surprising due to the similarities between the Pipe and the Cylinder classes. An example that shows the similarity between some of these targets is shown in Figure 4.10



(a) Pipe alarm

(b) Cylinder alarm

Figure 4.10: Example of two alarms, one pipe and one cylinder, that are difficult to distinguish.

Table 4.11: Confusion matrix for the MT-eFUMI algorithm. Total classification accuracy is 25.34%.

Class	**Non-Target**	**Block**	**Cone**	**Sphere**	**Torus**	**Pipe**	**Cylinder**
Non-Target	**29.10**	36.95	23.0	2.49	5.08	2.82	0.55
Block	17.74	**15.32**	0	66.94	0	0	0
Cone	1.82	38.18	**21.82**	14.55	18.18	0	5.45
Sphere	0	9.09	0	**86.36**	3.03	0	1.52
Torus	0	0	0	30.30	**69.70**	0	0
Pipe	55.81	44.19	0	0	0	**0**	0
Cylinder	22.39	28.81	4.77	28.81	8.99	1.28	**4.95**

Another matter than cannot be easily measured is that several of the alarms returned by the RX detector are not necessarily centered over the targets. The proposed method, as seen on the synthetic data test, can throw these points out as they are not similar to the other target points. If the patches were guaranteed to be centered over the actual target center the proposed method's performance should increase again, however that is not the case here as the classifier is designed to run on the output of the detector.

Next the proposed method is compared to the two other algorithms used for the previous experiments, TDDL and MT-eFUMI. The results for both of these are shown in the confusion matrices in Tables 4.12 and 4.11. The overall accuracies are 84.66% for TDDL and only 25.34% for MT-eFUMI.

The first thing to notice here is that MT-eFUMI does not perform very well by itself on this data. This would indicate that the multiple instance does not really add any benefit to this data. However, it can also be observed that the proposed algorithm does slightly better than the original TDDL algorithm. This indicates that adding eFUMI into the TDDL algorithm actually helped the classification performance of the algorithm. These two statements seem to contradict each other. It seems that the multiple instance learning when not a classification approach is not good effective enough for classification tasks. The real benefit from the proposed algorithm is that the multiple instance learning is combined with a classification model that has been proven to work quite well. So it seems that it is the

Table 4.12: Confusion matrix for the TDDL algorithm. Total classification accuracy is 84.66%.

Class	Non-Target	Block	Cone	Sphere	Torus	Pipe	Cylinder
Non-Target	**99.08**	0.0	0	0	0.14	0.0	0.69
Block	17.74	**3.23**	0	3.23	0	0	75.81
Cone	65.45	0	**7.27**	0	0	0	27.27
Sphere	10.61	13.64	0	**71.21**	0	0	4.55
Torus	9.09	3.03	0	3.03	**69.70**	3.03	12.12
Pipe	6.98	0	0	0	0	**60.47**	32.56
Cylinder	32.29	0.92	0	1.47	0	7.16	**58.17**

combination of the learned classification model along with the multiple instance model that actually improve the classification results.

Chapter 5

Summary and concluding remarks

This thesis has reviewed previous literature covering a variety of dictionary learning algorithms and methods. The basics of dictionary learning as a simple reconstruction algorithm along with various methods of sparse coding and dictionary updating. From the simple dictionary learning models more advanced methods of supervised learning were developed that could also learn a classification model alongside the dictionary. Another method of dictionary learning method that was discussed were the functions of multiple instances algorithms which used multiple instance learning in order to learn effective dictionaries when given data with uncertain labels.

The Task Driven Extended Functions of Multiple Instances algorithm was proposed as a modification of the Task-Driven Dictionary Learning model. The proposed algorithm incorporates principles from multiple instance learning to add to the classification model from Task-Driven Dictionary Learning to help the supervised dictionary learning algorithm become more equipped to handle data with uncertain labels.

The performance of the proposed algorithm has been thoroughly examined on steadily worsening data and compared this with the Task-Driven Dictionary Learning model. The proposed algorithm matches the performance of Task-Driven Dictionary Learning whenever the data labels have low levels of uncertainty. Then when the uncertainty of the data increases the proposed method is able to maintain good classification performance while the Task-Driven Dictionary Learning method cannot.

The performance of the proposed algorithm was also tested on two separate real world datasets and its classification performance was compared with that of the Task Driven Dictionary Learning and Multiple Target Extended Functions of Multiple Instances algorithms. These tests showed that the proposed method could improve upon the Task-Driven Dictionary Learning and Multiple Target Extended Functions of Multiple Instances for multiple class classification on sonar images and as a classifier for potential targets in wideband electromagnetic induction data.

A couple of issues that arose during the testing was that the algorithm had no way built in to handle data classes that were unbalanced, this is a common problem though so perhaps not that surprising. Also if there are only small differences between the target and non-target classes the algorithm may struggle to find a way to separate these classes.

Future work related to this thesis could be to extend the propose model to be able to better handle multiple class classification. The original Task-Driven Dictionary Learning model was able to do multiple class classification via a regression model, adding the regression model to work with the proposed method would open much wider application possibilities for the algorithm.

Appendix A

Glossary of Symbols

Symbol	**Meaning**
$\boldsymbol{A}$	Matrix of all sparse weight vectors (Dimensions $K \times N$)
B	Bag indicator
$\boldsymbol{D}$	Dictionary (Dimensions $L \times K$)
E	Stopping criteria
F	Objective function
I	Maximum number of iterations
K	Total number of dictionary elements
L	Length of feature vector $\boldsymbol{x}$
M	Number of non-target dictionary elements
N	Number of data points
P	Probability function
T	Number of target dictionary elements
$\boldsymbol{X}$	Matrix of all data points (Dimensions $L \times N$)
$\boldsymbol{d}$	Dictionary element

$\boldsymbol{d}^{(T)}$	Target Dictionary element
$\boldsymbol{d}^{(NT)}$	Non-Target Dictionary element
$\boldsymbol{e}_m$	Error term for non-target dictionary elements
$\boldsymbol{e}_t$	Error term for target dictionary elements
i	Iteration counter (Can also be a subscript indicator)
j	Subscript for j^{th} bag
k	Subscript for k^{th} dictionary element
l	Subscript for l^{th} dimension of the data
m	Subscript for m^{th} non-target dictionary element
m_0	Any specific non-target dictionary index
n	Subscript for n^{th} data point
s	Parameter for smoothing dictionary elements
t	Subscript for t^{th} target dictionary element
t_0	Any specific target dictionary index
u	Parameter for controlling tradeoff between error term and constraints in the objective function
v	Parameter for regularizing $\boldsymbol{w}$
$\boldsymbol{w}$	Model parameter vector (Dimensions $K \times 1$)
$\boldsymbol{x}$	Input data point (Dimensions $L \times 1$)
z	Latent class indicator variable (0 or 1)
z'	Estimated class label (-1 or 1)
$\boldsymbol{\alpha}$	Sparse weight vector (Dimensions $K \times 1$)
$\boldsymbol{\alpha}^*$	Optimal sparse weight vector (Dimensions $K \times 1$)
β	Probability scaling parameter
δ	Importance factor term
ϵ	Tuning parameter for δ function

λ	Sparsity Promoting parameter
$\boldsymbol{\mu}_0$	Global data mean (Dimensions $L \times 1$)
ρ	Learning rate parameter

Appendix B

Full Derivation Details

In this appendix the full details of the derivations are presented in order to make the methods section more readable. With the readability in mind the notation for a target dictionary element will be shorted from $\boldsymbol{d}_t^{(T)}$ to $\boldsymbol{d}_t$ and the notation for the non-target dictionary element will be similar, $\boldsymbol{d}_m^{(NT)}$ becomes $\boldsymbol{d}_m$.

B.1 Expectation of $\mathcal{F}$

Below are the steps needed to derived the expectation of the main objective function. The main steps here are to evaluate the expectation operator and then to simplify the resulting expression. The regularization terms are not included in this section because they are not a part of the expectation.

$$\mathcal{F}(\boldsymbol{w}, \boldsymbol{x}_n, \boldsymbol{D}) = \mathbb{E}_z \left[\frac{(1-u)\delta}{2} \left(z'_n - \boldsymbol{w}^\intercal \boldsymbol{\alpha}_n^* \right)^2 \right] \tag{B.1}$$

$$= \frac{(1-u)\delta}{2} (P(z=1)(1 - \boldsymbol{w}^\intercal \boldsymbol{\alpha}_n^*)^2 + P(z=0)(-1 - \boldsymbol{w}^\intercal \boldsymbol{\alpha}_n^*)^2) \tag{B.2}$$

$$\begin{aligned} = \frac{(1-u)\delta}{2} [&P(z=1)(1 - 2\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n + (\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n)^2) \\ &+ (P(z=0)(1 + 2\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n + (\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n)^2))] \end{aligned} \tag{B.3}$$

$$\begin{aligned} = \frac{(1-u)\delta}{2} [&P(z=1) - 2P(z=1)\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n + P(z=1)(\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n)^2 \\ &+ P(z=0) + 2P(z=0)\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n + P(z=0)(\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n)^2] \end{aligned} \tag{B.4}$$

$$\begin{aligned} = \frac{(1-u)\delta}{2} [&P(z=1) + P(z=0) - 2P(z=1)\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n \\ &+ 2P(z=0)\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n + P(z=1)(\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n)^2 + P(z=0)(\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n)^2] \end{aligned} \tag{B.5}$$

$$= \frac{(1-u)\delta}{2} [1 + 2(P(z=0) - P(z=1))\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n + (\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n)^2] \tag{B.6}$$

$$= \frac{(1-u)\delta}{2} [1 + 2(1 - 2P(z=1))\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n + (\boldsymbol{w}^\intercal \boldsymbol{\alpha}_n)^2] \tag{B.7}$$

B.2 Closed Forms of α

Below are the steps required to compute the closed form solution of Equation 3.5 for both the target and non-target sparse weights. The process followed for both of these expressions is to find the point where Equation 3.5 is at a minimum. In order to do this we can find the gradient of Equation 3.5 and set this value equal to zero, since it is assumed to be convex, and solve for α_{nt_0} and α_{nm_0}.

$$\begin{multline}\frac{\partial}{\partial \alpha_{t_0}} \left(\frac{P(z=1)}{2} \left\| \boldsymbol{x}_n - \sum_{t}^{T} \alpha_{nt} \boldsymbol{d}_t - \sum_{m}^{M} \alpha_{nm} \boldsymbol{d}_m \right\|_2^2 \right. \\ \left. + \frac{P(z=0)}{2} \left\| \boldsymbol{x}_n - \sum_{m}^{M} \alpha_{nm} \boldsymbol{d}_m \right\|_2^2 + \lambda \|\boldsymbol{\alpha}_n\|_1 \right) \tag{B.8}\end{multline}$$

$$= \frac{\partial}{\partial \alpha_{t_0}} \left(\frac{P(z=1)}{2} \left\| \boldsymbol{x}_n - \sum_{t}^{T} \alpha_{nt} \boldsymbol{d}_t - \sum_{m}^{M} \alpha_{nm} \boldsymbol{d}_m \right\|_2^2 + \lambda \|\boldsymbol{\alpha}_n\|_1 \right) \tag{B.9}$$

$$= P(z=1) \boldsymbol{d}_{t_0}^{\intercal} \left(\boldsymbol{x}_n - \sum_{t}^{T} \alpha_{nt} \boldsymbol{d}_t - \sum_{m}^{M} \alpha_{nm} \boldsymbol{d}_m \right) + \lambda \mathrm{sign}(\alpha_{nt_0}). \tag{B.10}$$

Now setting equal to zero and solving for α_{nt_0},

$$0 = P(z=1) \boldsymbol{d}_{t_0}^{\intercal} \left(\boldsymbol{x}_n - \sum_{t}^{T} \alpha_{nt} \boldsymbol{d}_t - \sum_{m}^{M} \alpha_{nm} \boldsymbol{d}_m \right) + \lambda \mathrm{sign}(\alpha_{nt_0}) \tag{B.11}$$

$$P(z=1) \boldsymbol{d}_{t_0}^{\intercal} \alpha_{nt_0} \boldsymbol{d}_{t_0} = P(z=1) \boldsymbol{d}_{t_0}^{\intercal} \left(\boldsymbol{x}_n - \sum_{t, t \neq t_0}^{T} \alpha_{nt} \boldsymbol{d}_t - \sum_{m}^{M} \alpha_{nm} \boldsymbol{d}_m \right) + \lambda \mathrm{sign}(\alpha_{nt_0}) \tag{B.12}$$

$$\begin{multline}\alpha_{nt_0} = \left(P(z=1)\ \boldsymbol{d}_{t_0}^{\intercal} \boldsymbol{d}_{t_0}\right)^{-1} \Bigg(P(z=1) \boldsymbol{d}_{t_0}^{\intercal} \\ \left(\boldsymbol{x}_n - \sum_{t, t \neq t_0}^{T} \alpha_{nt} \boldsymbol{d}_t - \sum_{m}^{M} \alpha_{nm} \boldsymbol{d}_m \right) + \lambda \mathrm{sign}(\alpha_{nt_0}) \Bigg) \tag{B.13}\end{multline}$$

And following the same steps for α_{m_0}

$$\begin{aligned}\frac{\partial}{\partial \alpha_{m_0}} \Bigg(\frac{P(z=1)}{2} \left\| \boldsymbol{x}_n - \sum_{t}^{T} \alpha_{nt}\boldsymbol{d}_t - \sum_{m}^{M} \alpha_{nm}\boldsymbol{d}_m \right\|_2^2 \\ + \frac{P(z=0)}{2} \left\| \boldsymbol{x}_n - \sum_{m}^{M} \alpha_{nm}\boldsymbol{d}_m \right\|_2^2 + \lambda \|\boldsymbol{\alpha}_n\|_1 \Bigg)\end{aligned} \tag{B.14}$$

$$\begin{aligned}= P(z=1)\boldsymbol{d}_{m0}^{\mathsf{T}} \left(\boldsymbol{x}_n - \sum_{t}^{T} \alpha_{nt}\boldsymbol{d}_t - \sum_{m}^{M} \alpha_{nm}\boldsymbol{d}_m \right) \\ + P(z=0)\boldsymbol{d}_{m0}^{\mathsf{T}} \left(\boldsymbol{x}_n - \sum_{m}^{M} \alpha_{nm}\boldsymbol{d}_m \right) + \lambda \mathrm{sign}(\alpha_{m_0})\end{aligned} \tag{B.15}$$

$$\begin{aligned}= P(z=1)\boldsymbol{d}_{m0}^{\mathsf{T}}\boldsymbol{x}_n - P(z=1)\boldsymbol{d}_{m0}^{\mathsf{T}} \sum_{t}^{T} \alpha_{nt}\boldsymbol{d}_t - P(z=1)\boldsymbol{d}_{m0}^{\mathsf{T}} \sum_{m}^{M} \alpha_{nm}\boldsymbol{d}_m \\ + P(z=0)\boldsymbol{d}_{m0}^{\mathsf{T}}\boldsymbol{x}_n - P(z=0)\boldsymbol{d}_{m0}^{\mathsf{T}} \sum_{m}^{M} \alpha_{nm}\boldsymbol{d}_m + \lambda \mathrm{sign}(\alpha_{m_0})\end{aligned} \tag{B.16}$$

$$\begin{aligned}= P(z=1)\boldsymbol{d}_{m0}^{\mathsf{T}}\boldsymbol{x}_n + P(z=0)\boldsymbol{d}_{m0}^{\mathsf{T}}\boldsymbol{x}_n - P(z=1)\boldsymbol{d}_{m0}^{\mathsf{T}} \sum_{t}^{T} \alpha_{nt}\boldsymbol{d}_t \\ - P(z=1)\boldsymbol{d}_{m0}^{\mathsf{T}} \sum_{m}^{M} \alpha_{nm}\boldsymbol{d}_m - P(z=0)\boldsymbol{d}_{m0}^{\mathsf{T}} \sum_{m}^{M} \alpha_{nm}\boldsymbol{d}_m + \lambda \mathrm{sign}(\alpha_{m_0})\end{aligned} \tag{B.17}$$

$$= \boldsymbol{d}_{m_0}^{\mathsf{T}}\boldsymbol{x}_n - \boldsymbol{d}_{m_0}^{\mathsf{T}} \sum \alpha_{nm}\boldsymbol{d}_m - P(z=1)\boldsymbol{d}_{m_0}^{\mathsf{T}} \sum_{t}^{T} \alpha_{nt}\boldsymbol{d}_t + \lambda \mathrm{sign}(\alpha_{m_0}) \tag{B.18}$$

Now setting equal to zero and solving for α_{m_0} again

$$0 = \boldsymbol{d}_{m_0}^{\mathsf{T}} \boldsymbol{x}_n - \boldsymbol{d}_{m_0}^{\mathsf{T}} \sum_{m}^{M} \alpha_{nm} \boldsymbol{d}_m - P(z=1) \boldsymbol{d}_{m_0}^{\mathsf{T}} \sum_{t}^{T} \alpha_{nt} \boldsymbol{d}_t + \lambda \text{sign}(\alpha_{m_0}) \tag{B.19}$$

$$\boldsymbol{d}_{m_0}^{\mathsf{T}} \alpha_{nm_0} \boldsymbol{d}_{m_0} = \boldsymbol{d}_{m_0}^{\mathsf{T}} \boldsymbol{x}_n - \boldsymbol{d}_{m_0}^{\mathsf{T}} \sum_{m, m \neq m_0}^{M} \alpha_{nm} \boldsymbol{d}_m - P(z=1) \boldsymbol{d}_{m_0}^{\mathsf{T}} \sum_{t}^{T} \alpha_{nt} \boldsymbol{d}_t + \lambda \text{sign}(\alpha_{m_0}) \tag{B.20}$$

$$\begin{aligned} \alpha_{nm_0} = \left(\boldsymbol{d}_{m_0}^{\mathsf{T}} \boldsymbol{d}_{m_0}\right)^{-1} \Bigg(& \boldsymbol{d}_{m_0}^{\mathsf{T}} \boldsymbol{x}_n - \boldsymbol{d}_{m_0}^{\mathsf{T}} \sum_{m, m \neq m_0}^{M} \alpha_{nm} \boldsymbol{d}_m \\ & - P(z=1) \boldsymbol{d}_{m_0}^{\mathsf{T}} \sum_{t}^{T} \alpha_{nt} \boldsymbol{d}_t + \lambda \text{sign}(\alpha_{m_0}) \Bigg) \end{aligned} \tag{B.21}$$

B.3 Gradient of $\mathcal{F}$

Now that we have expressions for the subfunctions of the main objective function in Equation 3.3, the gradient of the objective function can be derived. Below is this derivation, while the derivation is quite long chain rule is used for the majority of the derivation. The smoothness term is not included here because it is derived and explained in the main text of Section 3.3

$$
\begin{aligned}
&\nabla_{\boldsymbol{d}_k}\left(\frac{(1-u)\delta}{2}[1+2(1-2P(z=1))\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n+(\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n)^2]\right.\\
&\qquad\left.+\frac{u}{2}\left(\sum_{t=1}^{T}\|\boldsymbol{d}_t-\boldsymbol{\mu}_0\|_2^2+\sum_{k=1}^{k}\|\boldsymbol{d}_k-\boldsymbol{\mu}_0\|_2^2\right)+\frac{v}{2}\|\boldsymbol{w}\|_2^2\right)
\end{aligned}
\tag{B.22}
$$

$$
\begin{aligned}
&=\nabla_{\boldsymbol{d}_k}\left(\frac{(1-u)\delta}{2}[1+2(1-2P(z=1))\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n+(\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n)^2]\right.\\
&\qquad\left.+\frac{u}{2}\left(\sum_{t=1}^{T}\|\boldsymbol{d}_t-\boldsymbol{\mu}_0\|_2^2+\sum_{k=1}^{k}\|\boldsymbol{d}_k-\boldsymbol{\mu}_0\|_2^2\right)\right)
\end{aligned}
\tag{B.23}
$$

$$
=\frac{(1-u)\delta}{2}\nabla_{\boldsymbol{d}_k}\left(1+2(1-2P(z=1))\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n+(\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n)^2\right)+u(\boldsymbol{d}_k-\boldsymbol{\mu}_0)
\tag{B.24}
$$

$$
=\frac{(1-u)\delta}{2}\left(2(1-2P(z=1))\nabla_{\boldsymbol{d}_k}\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n+\nabla_{\boldsymbol{d}_k}(\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n)^2\right)+u(\boldsymbol{d}_k-\boldsymbol{\mu}_0)
\tag{B.25}
$$

$$
=\frac{(1-u)\delta}{2}\left(2(1-2P(z=1))\nabla_{\alpha_{nk}}(\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n)\nabla_{\boldsymbol{d}_t}\alpha_{nk}+\nabla_{\alpha_{nk}}(\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n)^2\nabla_{\boldsymbol{d}_k}\alpha_{nk}\right)+u(\boldsymbol{d}_k-\boldsymbol{\mu}_0)
\tag{B.26}
$$

$$
=\frac{(1-u)\delta}{2}\left(2(1-2P(z=1))w_k\nabla_{\boldsymbol{d}_k}\alpha_{nk}+2(\boldsymbol{w}^{\intercal}\boldsymbol{\alpha}_n)w_k\nabla_{\boldsymbol{d}_k}\alpha_{nk}\right)+u(\boldsymbol{d}_k-\boldsymbol{\mu}_0)
\tag{B.27}
$$

To finish the derivation the gradient of both α_{nt} and α_{nm} need to be derived. For α_{nt}

$$\begin{aligned}\nabla_{d_t}\alpha_{nt_0} = \nabla_{d_t}\left(P(z=1)\ \boldsymbol{d}_{t_0}^{\intercal}\boldsymbol{d}_{t_0}\right)^{-1}\Bigg(P(z=1)\boldsymbol{d}_{t_0}^{\intercal} \\ \left(\boldsymbol{x}_n - \sum_{t,t\neq t_0}^{T}\alpha_{nt}\boldsymbol{d}_t - \sum_{m}^{M}\alpha_{nm}\boldsymbol{d}_m\right) + \lambda\text{sign}(\alpha_{nt_0})\Bigg)\end{aligned} \tag{B.28}$$

$$\text{Let } \boldsymbol{e}_t = \left(\boldsymbol{x}_n - \sum_{t,t\neq t_0}^{T}\alpha_{nt}\boldsymbol{d}_t - \sum_{m}^{M}\alpha_{nm}\boldsymbol{d}_m\right) \text{ be the residual error}$$

$$= \nabla_{d_t}\left(P(z=1)\boldsymbol{d}_{t_0}^{\intercal}\boldsymbol{d}_{t_0}\right)^{-1}\left(P(z=1)\boldsymbol{d}_{t_0}^{\intercal}\boldsymbol{e}_t + \lambda\text{sign}(\alpha_{nt_0})\right) \tag{B.29}$$

$$\begin{aligned}&= \left(\nabla_{d_t}\left(P(z=1)\boldsymbol{d}_{t_0}^{\intercal}\boldsymbol{d}_{t_0}\right)^{-1}\right)\left(P(z=1)\boldsymbol{d}_{t_0}^{\intercal}\boldsymbol{e}_t + \lambda\text{sign}(\alpha_{nt_0})\right) \\ &\quad + \left(P(z=1)\boldsymbol{d}_{t_0}^{\intercal}\boldsymbol{d}_{t_0}\right)^{-1}\nabla_{d_t}\left(P(z=1)\boldsymbol{d}_{t_0}^{\intercal}\boldsymbol{e}_t + \lambda\text{sign}(\alpha_{nt_0})\right)\end{aligned} \tag{B.30}$$

$$= \frac{-2\boldsymbol{d}_{t_0}}{P(z=1)(\boldsymbol{d}_{t_0}^{\intercal}\boldsymbol{d}_{t_0})^2}\left(P(z=1)\boldsymbol{d}_{t_0}^{\intercal}\boldsymbol{e}_t + \lambda\text{sign}(\alpha_{nt_0})\right) + \left(P(z=1)\boldsymbol{d}_{t_0}^{\intercal}\boldsymbol{d}_{t_0}\right)^{-1}\boldsymbol{e}_t, \tag{B.31}$$

and for α_{nm}

$$\begin{aligned}\nabla_{\boldsymbol{d}_{m_0}} \alpha_{nm_0} = \nabla_{\boldsymbol{d}_{m_0}} &\left(\left(\boldsymbol{d}_{m_0}^{\intercal} \boldsymbol{d}_{m_0}\right)^{-1} \left(\boldsymbol{d}_{m_0}^{\intercal} \boldsymbol{x}_n - \boldsymbol{d}_{m_0}^{\intercal} \sum_{m,m\neq m_0}^{M} \alpha_{nm} \boldsymbol{d}_m \right.\right. \\ &\left.\left. \quad - \; P(z=1) \boldsymbol{d}_{m_0}^{\intercal} \sum_{t}^{T} \alpha_{nt} \boldsymbol{d}_t + \lambda \text{sign}(\alpha_{m_0}) \right)\right)\end{aligned} \tag{B.32}$$

$$\begin{aligned}= \nabla_{\boldsymbol{d}_{m_0}} &\left(\left(\boldsymbol{d}_{m_0}^{\intercal} \boldsymbol{d}_{m_0}\right)^{-1} \left(\boldsymbol{d}_{m_0}^{\intercal} \left(\boldsymbol{x}_n - \sum_{m,m\neq m_0}^{M} \alpha_{nm} \boldsymbol{d}_m \right.\right.\right. \\ &\left.\left.\left. \quad - \; P(z=1) \sum_{t}^{T} \alpha_{nt} \boldsymbol{d}_t \right) + \lambda \text{sign}(\alpha_{m_0}) \right)\right)\end{aligned} \tag{B.33}$$

Let $\boldsymbol{e}_m = \left(\boldsymbol{x}_n - \sum_{m,m\neq m_0}^{M} \alpha_{nm} \boldsymbol{d}_m - P(z=1) \sum_{t}^{T} \alpha_{nt} \boldsymbol{d}_t \right)$ be the residual error

$$= \nabla_{d_m} \left(\boldsymbol{d}_{m_0}^{\intercal} \boldsymbol{d}_{m_0}\right)^{-1} \left(P(z=1) \boldsymbol{d}_{m_0}^{\intercal} \boldsymbol{e}_m + \lambda \text{sign}(\alpha_{nm_0}) \right) \tag{B.34}$$

$$\begin{aligned}=& \left(\nabla_{\boldsymbol{d}_{m_0}} (\boldsymbol{d}_{m_0}^{\intercal} \boldsymbol{d}_{m_0})^{-1} \right) \left(P(z=1) \boldsymbol{d}_{m_0}^{\intercal} \boldsymbol{e}_m + \lambda \text{sign}(\alpha_{nm_0}) \right) \\ &\qquad + (\boldsymbol{d}_{m_0}^{\intercal} \boldsymbol{d}_{m_0})^{-1} \left(\nabla_{\boldsymbol{d}_{m_0}} P(z=1) \boldsymbol{d}_{m_0}^{\intercal} \boldsymbol{e}_m + \lambda \text{sign}(\alpha_{nm_0}) \right)\end{aligned} \tag{B.35}$$

$$= \frac{-2\boldsymbol{d}_{m_0}}{(\boldsymbol{d}_{m_0}^{\intercal} \boldsymbol{d}_{m_0})^2} \left(P(z=1) \boldsymbol{d}_{m_0}^{\intercal} \boldsymbol{e}_m + \lambda \text{sign}(\alpha_{nm_0}) \right) + (\boldsymbol{d}_{m_0}^{\intercal} \boldsymbol{d}_{m_0})^{-1} \boldsymbol{e}_m \tag{B.36}$$

Appendix C

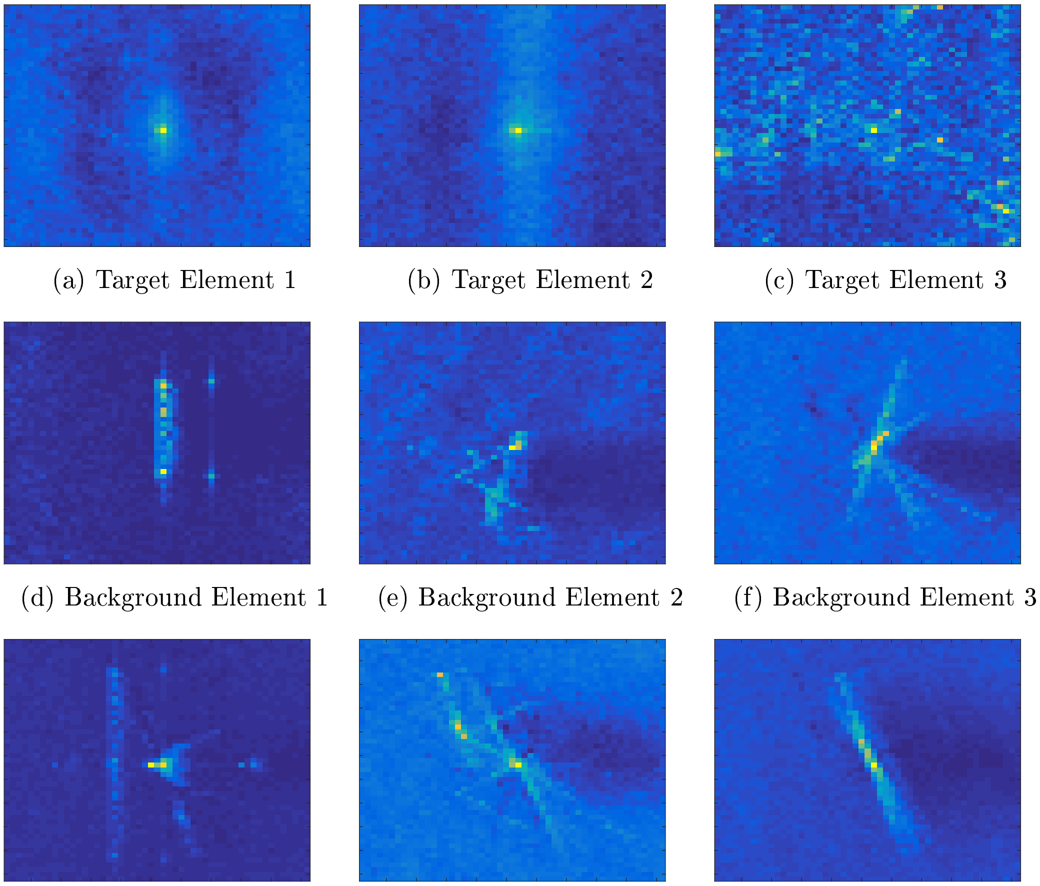
Additional Results

This appendix contains the complete set of results that were mentioned throughout the rest of the thesis.

C.1 Sonar Data

Below are the remaining six dictionaries that were created in the one versus all classification scheme that was used for the Sonar classification results.

C.1.1 TD-*e*FUMI



(a) Target Element 1

(b) Target Element 2

(c) Target Element 3

(d) Background Element 1

(e) Background Element 2

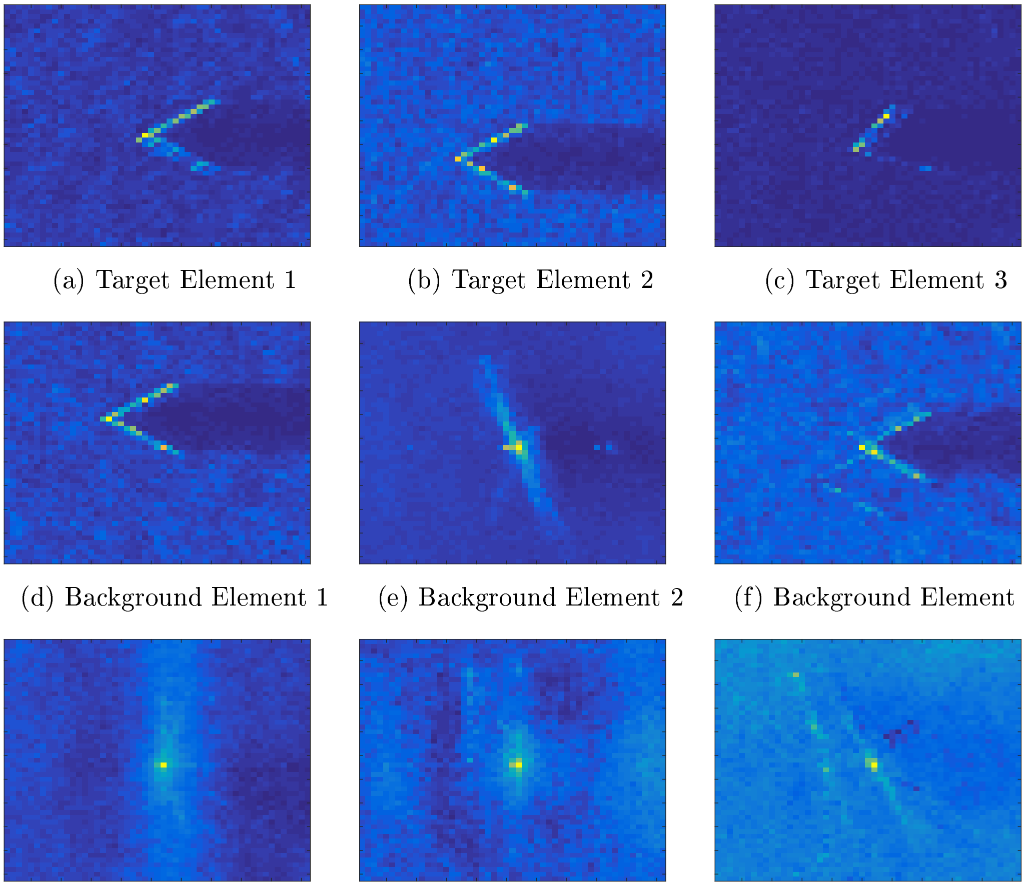
(f) Background Element 3

(g) Background Element 4

(h) Background Element 5

(i) Background Element 6

Figure C.1: TD-*e*FUMI dictionary for one versus all classification when the non-target class is the target class.



(a) Target Element 1

(b) Target Element 2

(c) Target Element 3

(d) Background Element 1

(e) Background Element 2

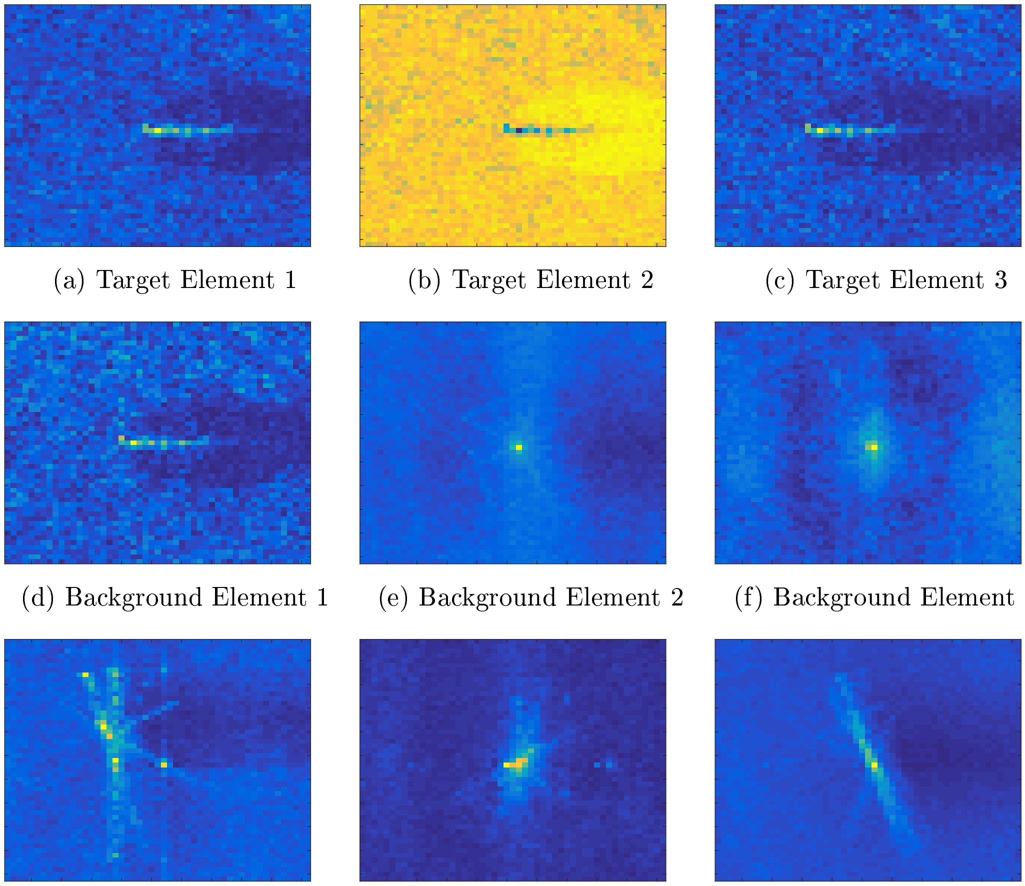
(f) Background Element 3

(g) Background Element 4

(h) Background Element 5

(i) Background Element 6

Figure C.2: TD-*e*FUMI dictionary for one versus all classification when the block class is the target class.



(a) Target Element 1 (b) Target Element 2 (c) Target Element 3

(d) Background Element 1 (e) Background Element 2 (f) Background Element 3

(g) Background Element 4 (h) Background Element 5 (i) Background Element 6

Figure C.3: TD-*e*FUMI dictionary for one versus all classification when the cone class is the target class.

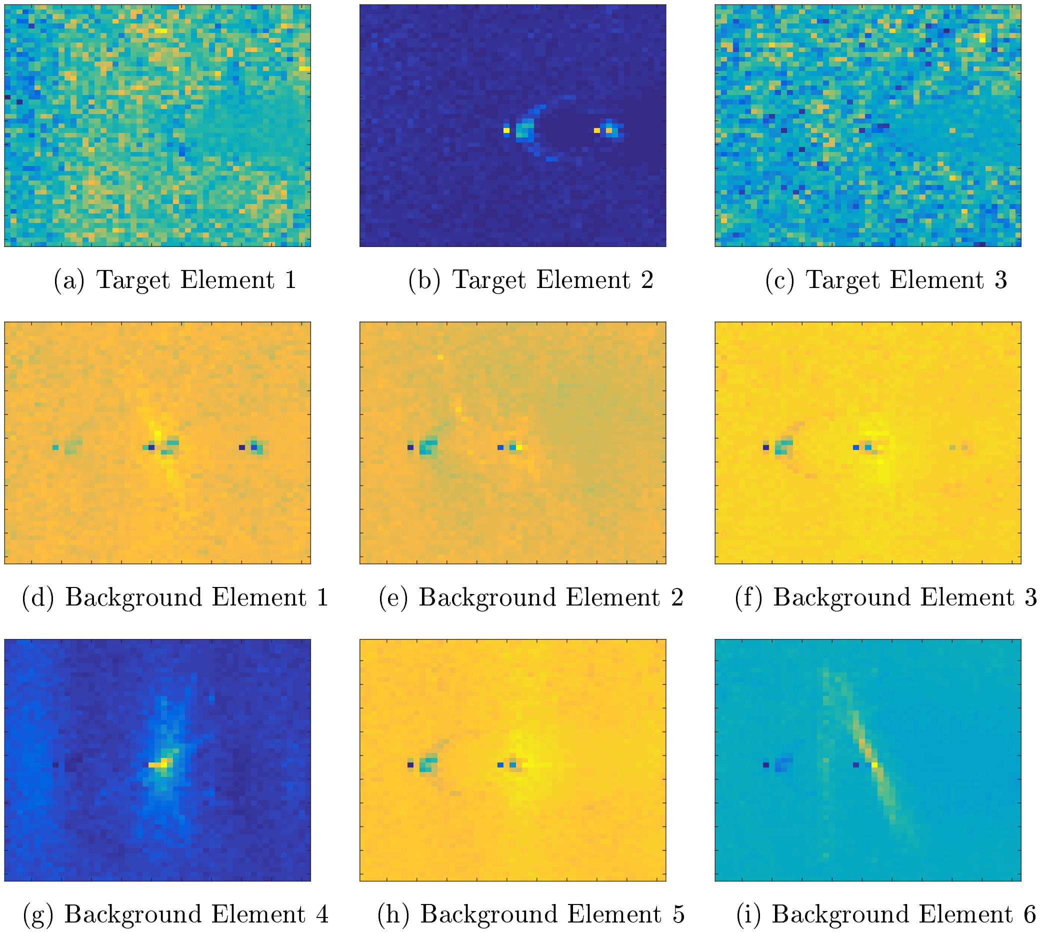
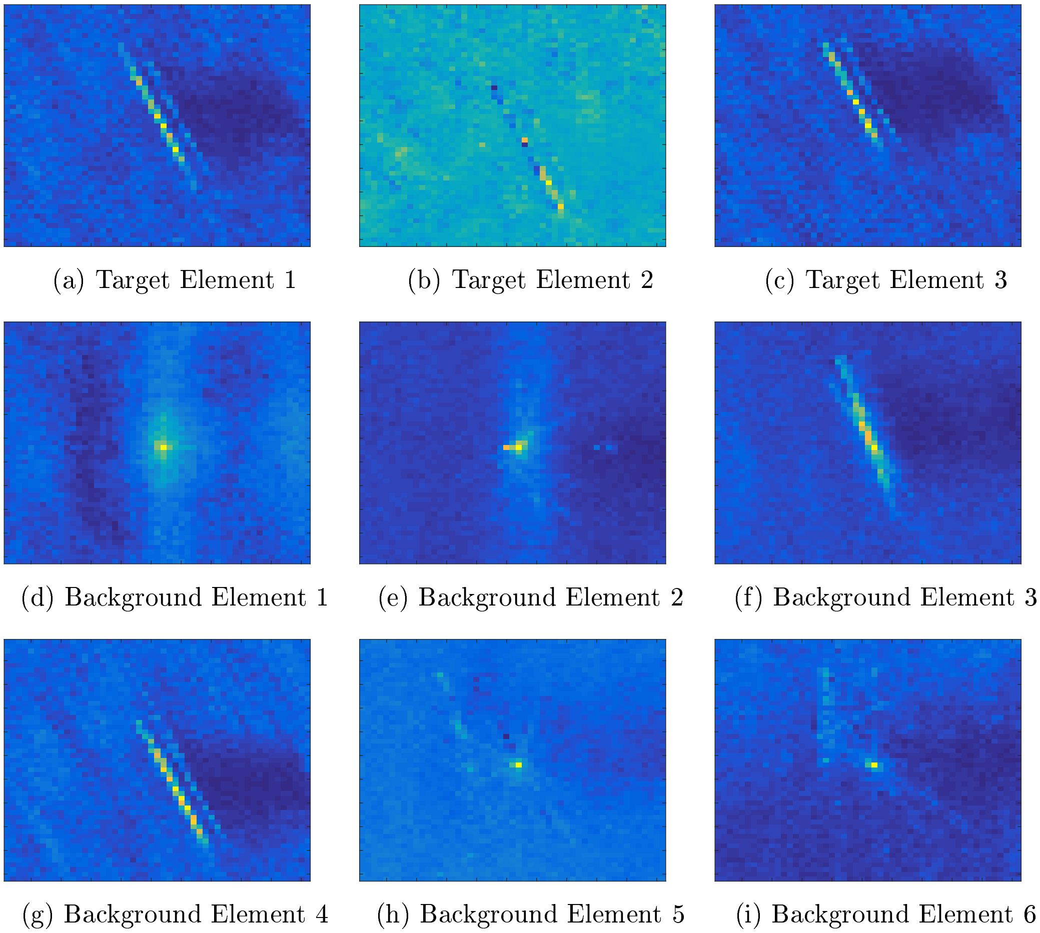


Figure C.4: TD-*e*FUMI dictionary for one versus all classification when the Torus class is the target class.



(a) Target Element 1

(b) Target Element 2

(c) Target Element 3

(d) Background Element 1

(e) Background Element 2

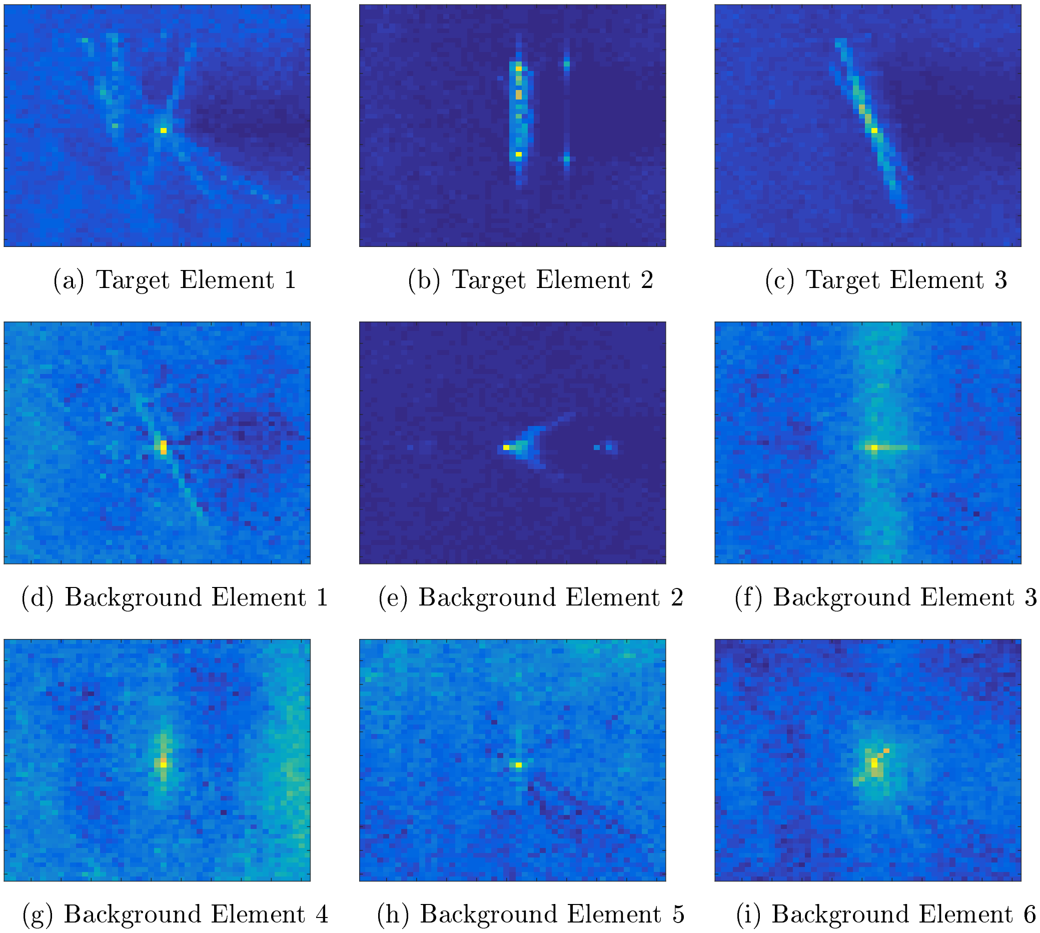
(f) Background Element 3

(g) Background Element 4

(h) Background Element 5

(i) Background Element 6

Figure C.5: TD-*e*FUMI dictionary for one versus all classification when the Pipe class is the target class.



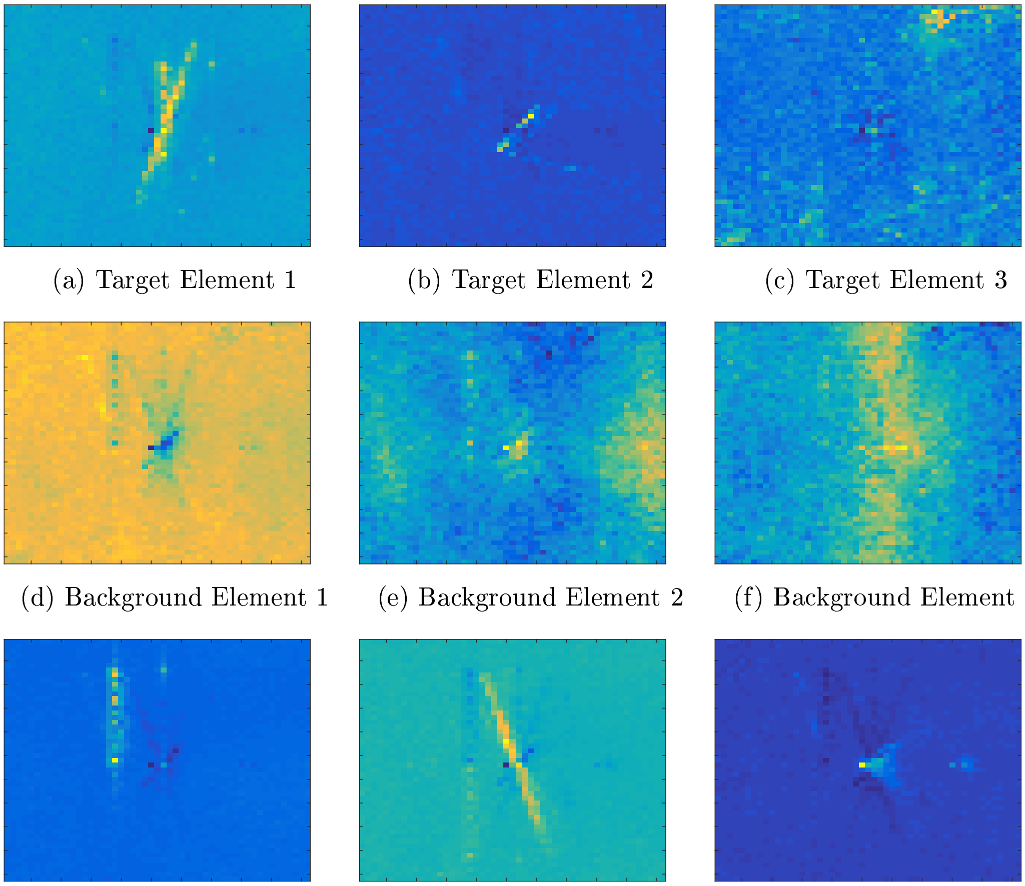
(a) Target Element 1 (b) Target Element 2 (c) Target Element 3

(d) Background Element 1 (e) Background Element 2 (f) Background Element 3

(g) Background Element 4 (h) Background Element 5 (i) Background Element 6

Figure C.6: TD-*e*FUMI dictionary for one versus all classification when the Cylinder class is the target class.

C.1.2 TDDL



(a) Target Element 1

(b) Target Element 2

(c) Target Element 3

(d) Background Element 1

(e) Background Element 2

(f) Background Element 3

(g) Background Element 4

(h) Background Element 5

(i) Background Element 6

Figure C.7: TDDL dictionary for one versus all classification when the non-target class is the target class.

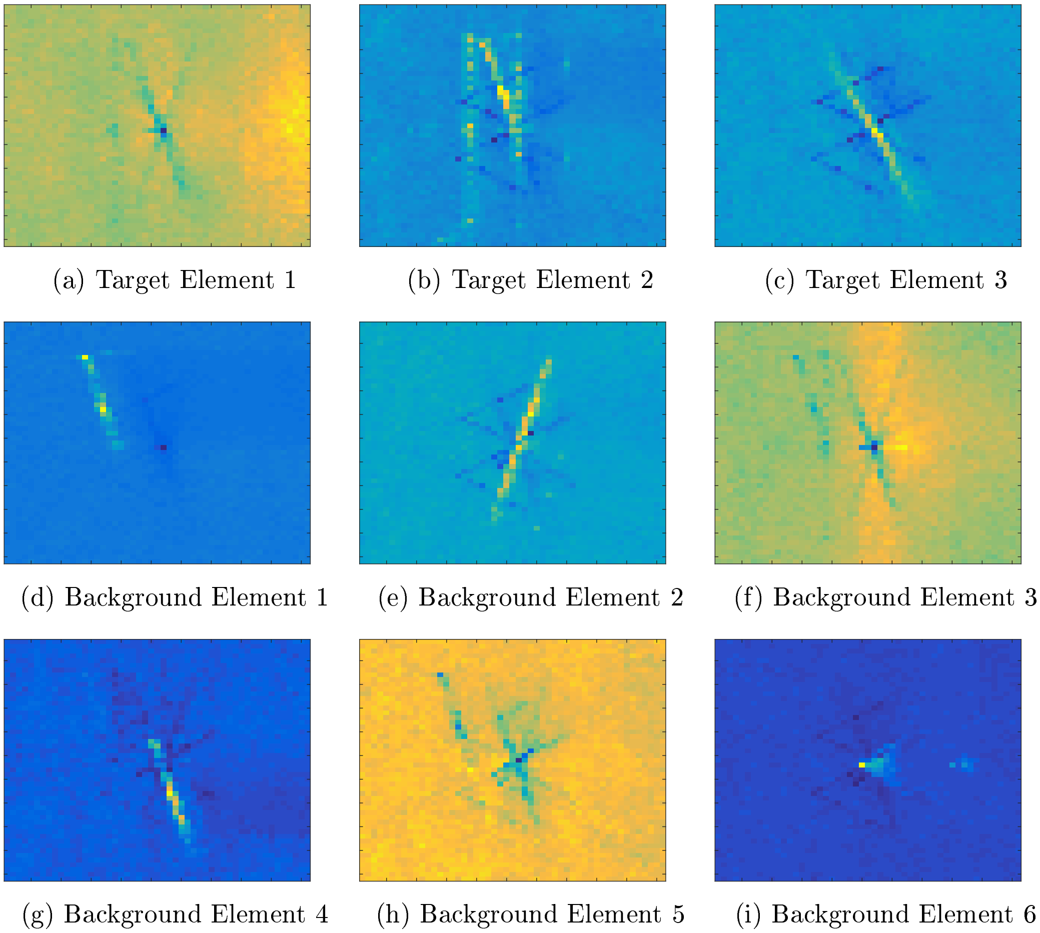
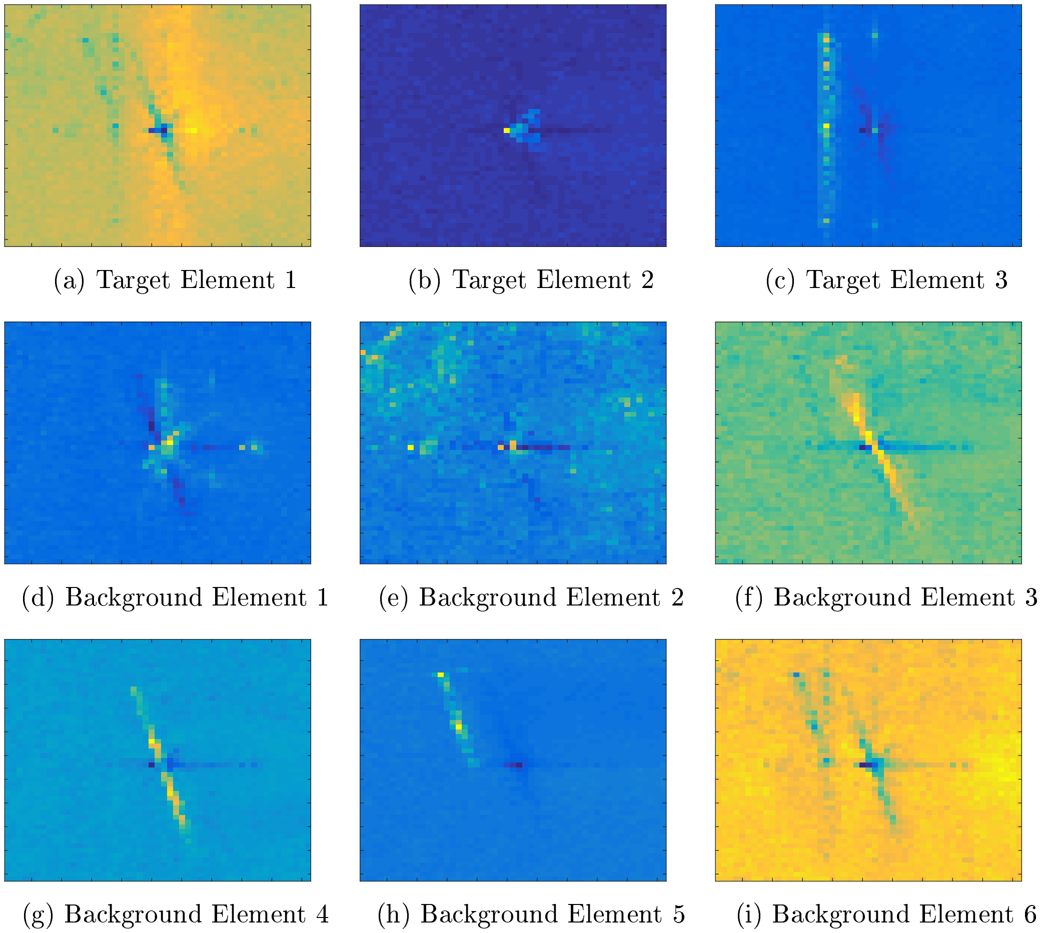


Figure C.8: TDDL dictionary for one versus all classification when the block class is the target class.



(a) Target Element 1

(b) Target Element 2

(c) Target Element 3

(d) Background Element 1

(e) Background Element 2

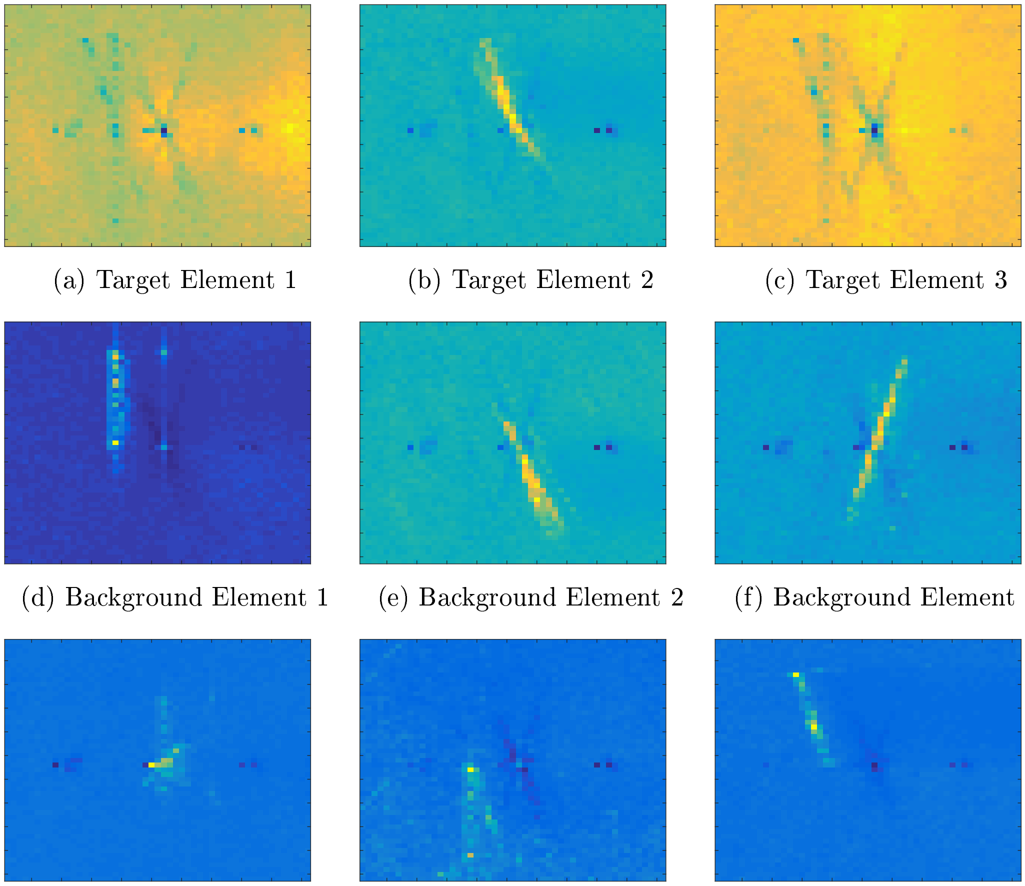
(f) Background Element 3

(g) Background Element 4

(h) Background Element 5

(i) Background Element 6

Figure C.9: TDDL dictionary for one versus all classification when the cone class is the target class.



(a) Target Element 1

(b) Target Element 2

(c) Target Element 3

(d) Background Element 1

(e) Background Element 2

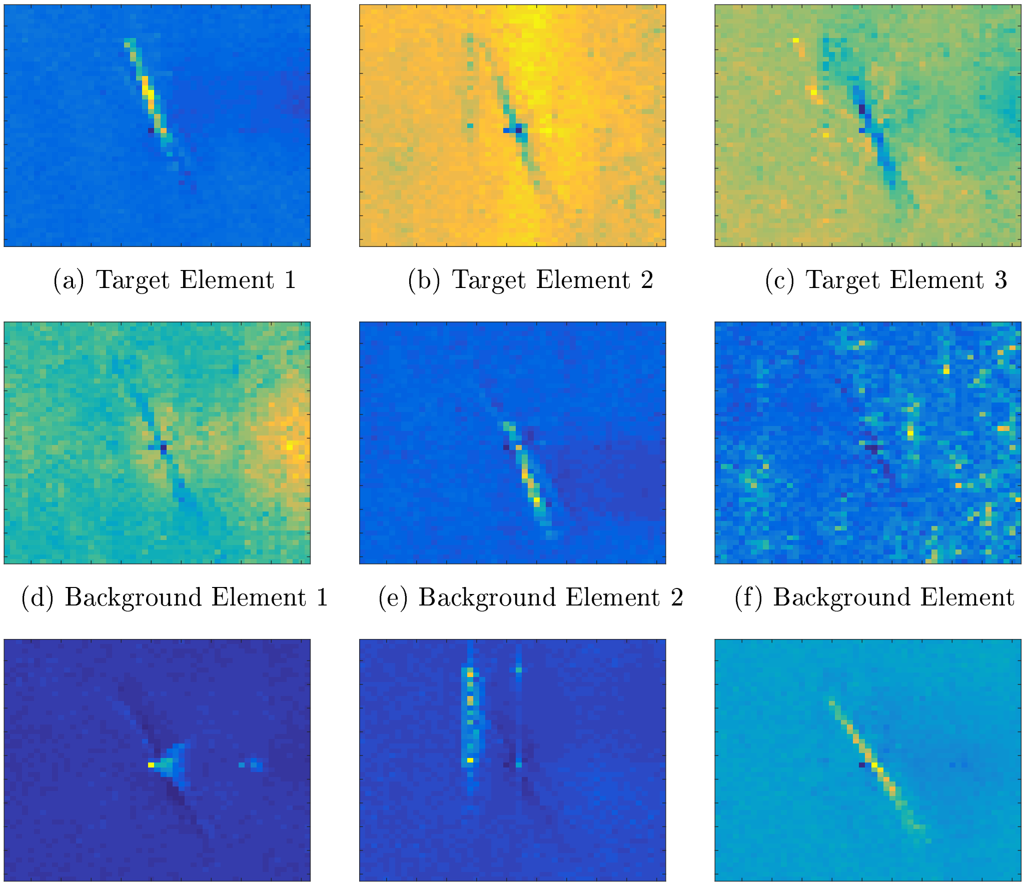
(f) Background Element 3

(g) Background Element 4

(h) Background Element 5

(i) Background Element 6

Figure C.10: TDDL dictionary for one versus all classification when the Torus class is the target class.



(a) Target Element 1

(b) Target Element 2

(c) Target Element 3

(d) Background Element 1

(e) Background Element 2

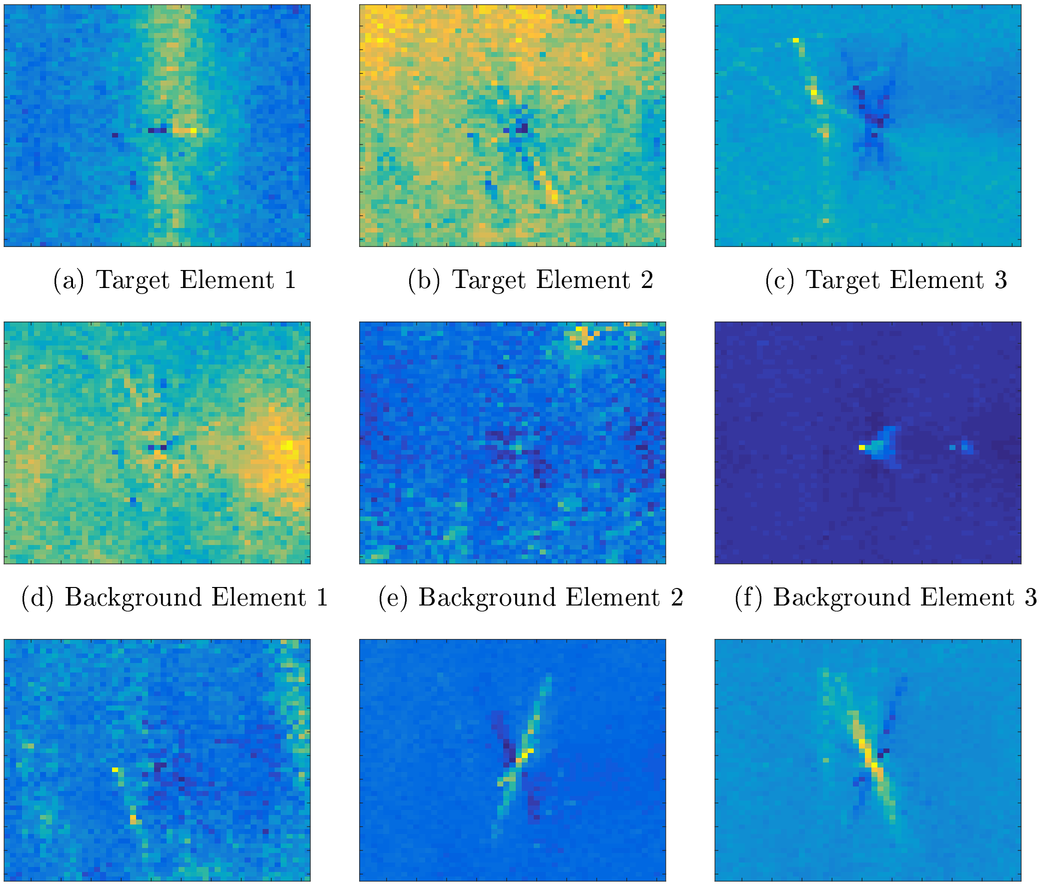
(f) Background Element 3

(g) Background Element 4

(h) Background Element 5

(i) Background Element 6

Figure C.11: TDDL dictionary for one versus all classification when the Pipe class is the target class.



(a) Target Element 1

(b) Target Element 2

(c) Target Element 3

(d) Background Element 1

(e) Background Element 2

(f) Background Element 3

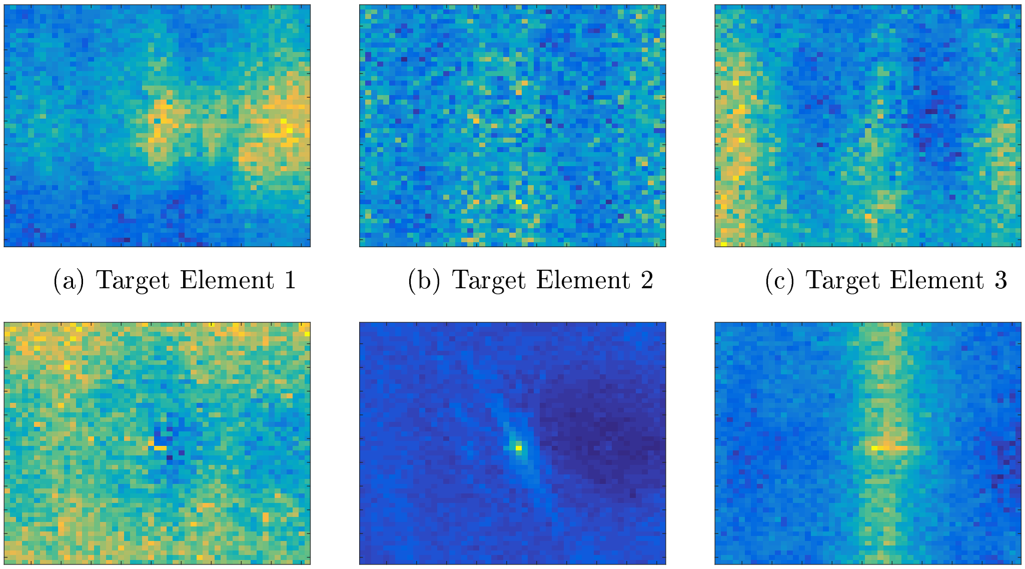
(g) Background Element 4

(h) Background Element 5

(i) Background Element 6

Figure C.12: TDDL dictionary for one versus all classification when the Cylinder class is the target class.

C.1.3 MT-*e*FUMI



(a) Target Element 1

(b) Target Element 2

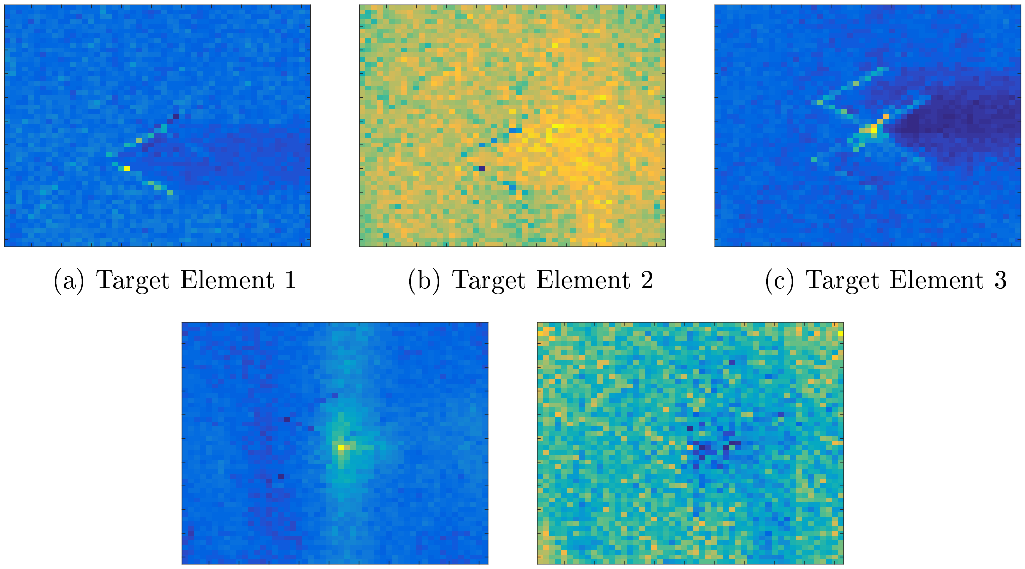
(c) Target Element 3

(d) Background Element 1

(e) Background Element 2

(f) Background Element 3

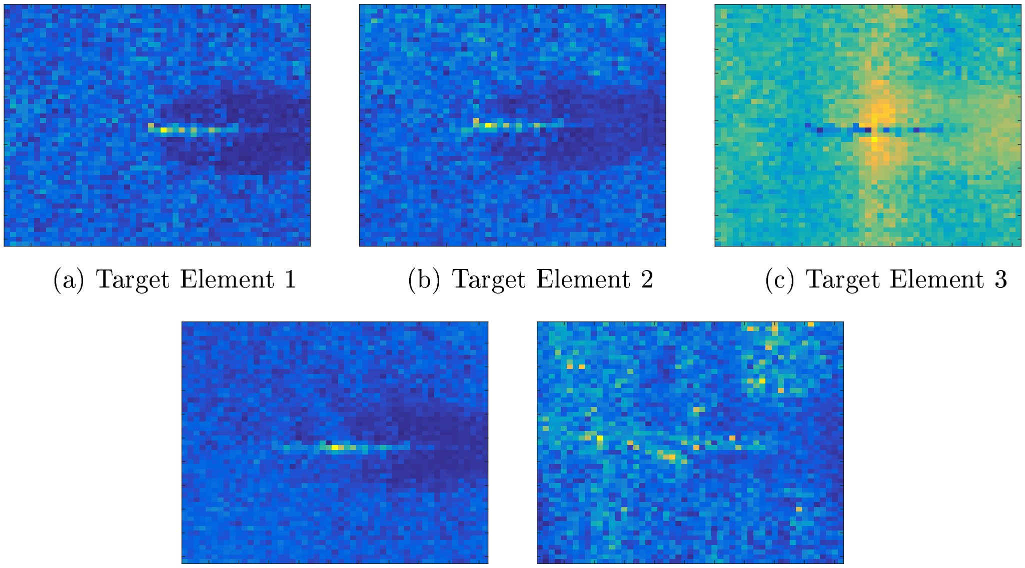
Figure C.13: MT-*e*FUMI dictionary for one versus all classification when the non-target class is the target class.



(a) Target Element 1 (b) Target Element 2 (c) Target Element 3

(d) Background Element 1 (e) Background Element 2

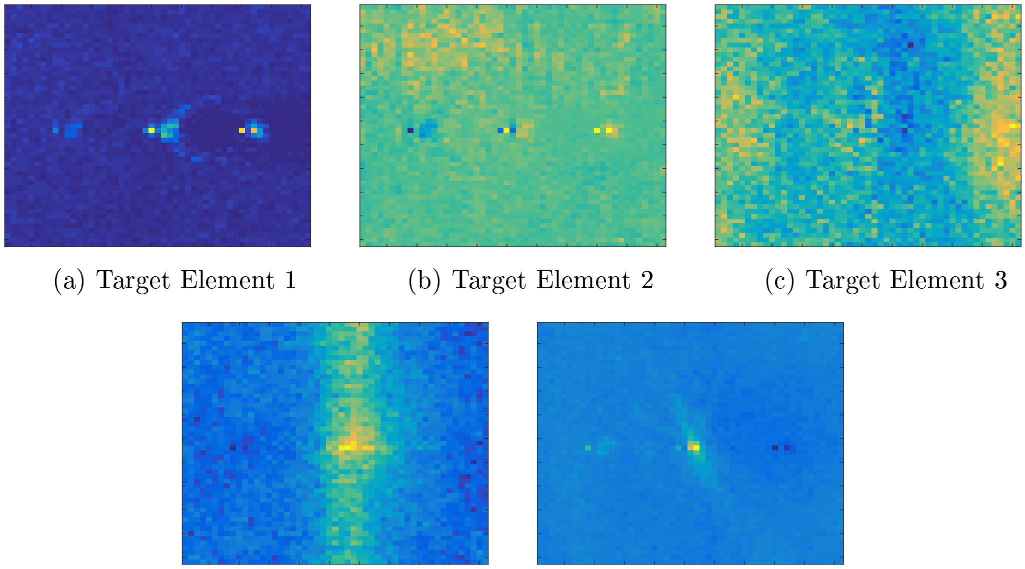
Figure C.14: MT-*e*FUMI dictionary for one versus all classification when the block class is the target class.



(a) Target Element 1 (b) Target Element 2 (c) Target Element 3

(d) Background Element 1 (e) Background Element 2

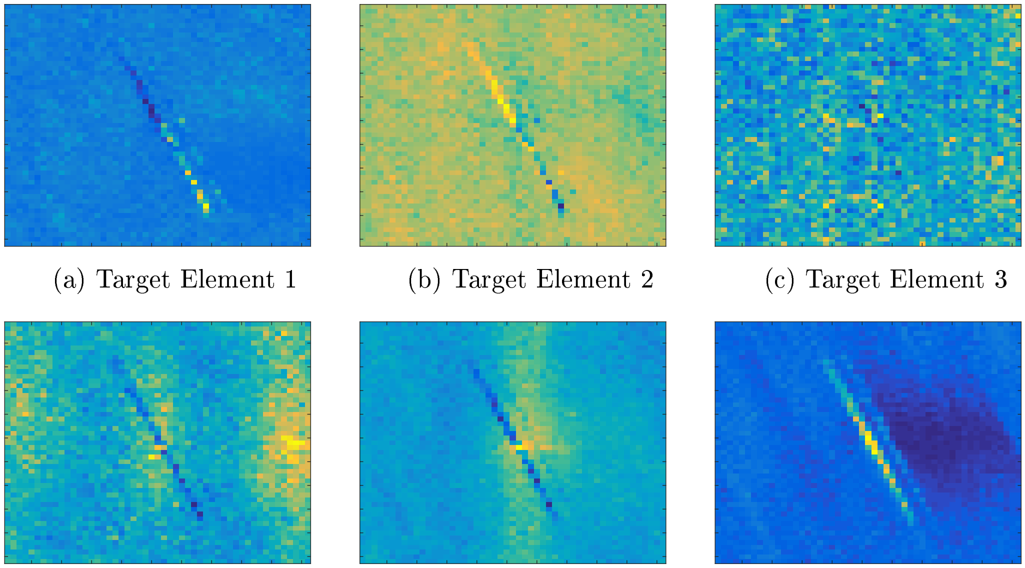
Figure C.15: MT-*e*FUMI dictionary for one versus all classification when the cone class is the target class.



(a) Target Element 1 (b) Target Element 2 (c) Target Element 3

(d) Background Element 1 (e) Background Element 2

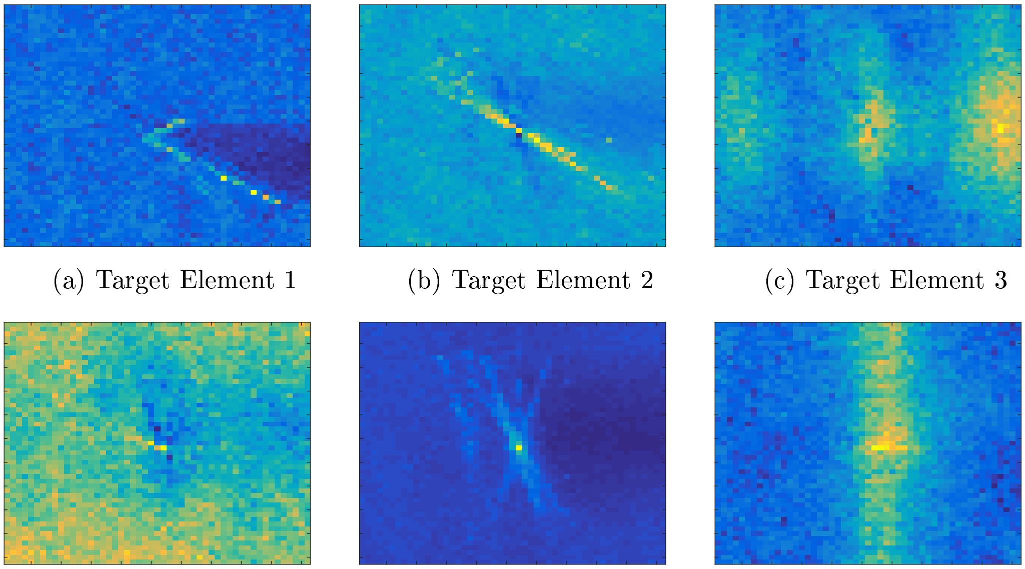
Figure C.16: MT-*e*FUMI dictionary for one versus all classification when the Torus class is the target class.



(a) Target Element 1 (b) Target Element 2 (c) Target Element 3

(d) Background Element 1 (e) Background Element 2 (f) Background Element 3

Figure C.17: MT-*e*FUMI dictionary for one versus all classification when the Pipe class is the target class.



(a) Target Element 1

(b) Target Element 2

(c) Target Element 3

(d) Background Element 1

(e) Background Element 2

(f) Background Element 3

Figure C.18: MT-*e*FUMI dictionary for one versus all classification when the Cylinder class is the target class.

Bibliography

[1] Sean Goldberg, Taylor Glenn, Joseph N. Wilson, and Paul D. Gader. Landmine detection using two-tapped joint orthogonal matching pursuits. *Proc. SPIE*, 8357:83570B–83570B–8, 2012.

[2] I Tosic and P. Frossard. Dictionary learning. *Signal Processing Magazine, IEEE*, 28(2):27–38, March 2011.

[3] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 689–696, New York, NY, USA, 2009. ACM.

[4] V. Patel, Yonggang Shi, P.M. Thompson, and A.W. Toga. K-svd for hardi denoising. In *Biomedical Imaging: From Nano to Macro, 2011 IEEE International Symposium on*, pages 1805–1808, March 2011.

[5] J.J. Fuchs. Recovery of exact sparse representations in the presence of bounded noise. *Information Theory, IEEE Transactions on*, 51(10):3601–3608, Oct 2005.

[6] J.-J. Fuchs. Recovery of exact sparse representations in the presence of noise. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on*, volume 2, pages ii–533–6 vol.2, May 2004.

[7] J. Mairal, F. Bach, and J. Ponce. Task-driven dictionary learning. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(4):791–804, April 2012.

[8] A. Shrivastava, J.K. Pillai, V.M. Patel, and R. Chellappa. Learning discriminative dictionaries with partially labeled data. In *Image Processing (ICIP), 2012 19th IEEE International Conference on*, pages 3113–3116, Sept 2012.

[9] Yi-Chen Chen, V.M. Patel, R. Chellappa, and P.J. Phillips. Ambiguously labeled learning using dictionaries. *Information Forensics and Security, IEEE Transactions on*, 9(12):2076–2088, Dec 2014.

[10] Jaume Amores. Multiple instance classification: Review, taxonomy and comparative study. *Artificial Intelligence*, 201(0):81 – 105, 2013.

[11] A. Zare and C. Jiao. Extended functions of multiple instances for target characterization. In *6th IEEE Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, June 2014.

[12] Alina Zare, Matthew Cook, Brendan Alvey, and Dominic K. Ho. Multiple instance dictionary learning for subsurface object detection using handheld emi. *Proc. SPIE*, 9454:94540G–94540G–8, 2015.

[13] Jianchao Yang, Jiangping Wang, and T. Huang. Learning the sparse representation for classification. In *Multimedia and Expo (ICME), 2011 IEEE International Conference on*, pages 1–6, July 2011.

[14] Zhong Zhao and Guocan Feng. A dictionary-based algorithm for dimensionality reduction and data reconstruction. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 1556–1561, Aug 2014.

[15] K.K. Herrity, A.C. Gilbert, and J.A. Tropp. Sparse approximation via iterative thresholding. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 3, pages III–III, May 2006.

[16] Y.C. Pati, R. Rezaiifar, and P.S. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 40–44 vol.1, Nov 1993.

[17] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67:301–320, 2005.

[18] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics*, 32(2):407–499, 2004.

[19] K. Engan, S.O. Aase, and J. Hakon Husoy. Method of optimal directions for frame design. In *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on*, volume 5, pages 2443–2446 vol.5, 1999.

[20] M. Aharon, M. Elad, and A. Bruckstein. k -svd: An algorithm for designing overcomplete dictionaries for sparse representation. *Signal Processing, IEEE Transactions on*, 54(11):4311–4322, Nov 2006.

[21] G.H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5):403–420, 1970.

[22] F. Pourkamali Anaraki and S.M. Hughes. Compressive k-svd. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 5469–5473, May 2013.

[23] Zhuolin Jiang, Zhe Lin, and L.S. Davis. Learning a discriminative dictionary for sparse coding via label consistent k-svd. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1697–1704, June 2011.

[24] Zhuolin Jiang, Zhe Lin, and L.S. Davis. Label consistent k-svd: Learning a discriminative dictionary for recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(11):2651–2664, Nov 2013.

[25] Bin Shan, Wei Hao, and Rui Zhao. Infrared image de-noising based on k-svd overcomplete dictionaries learning. In *Image and Signal Processing (CISP), 2012 5th International Congress on*, pages 316–320, Oct 2012.

[26] Léon Bottou. Stochastic learning. In Olivier Bousquet and Ulrike von Luxburg, editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag, Berlin, 2004.

[27] Lang Chen and Jianjun Wang. Dictionary learning with weighted stochastic gradient descent. In *Computational Problem-Solving (ICCP), 2012 International Conference on*, pages 9–12, Oct 2012.

[28] K. Labusch, E. Barth, and T. Martinetz. Robust and fast learning of sparse codes with stochastic gradient descent. *Selected Topics in Signal Processing, IEEE Journal of*, 5(5):1048–1060, Sept 2011.

[29] Chih jen Lin. Projected gradient methods for non-negative matrix factorization. Technical report, Neural Computation, 2007.

[30] G. Monaci and P. Vandergheynst. Learning structured dictionaries for image representation. In *Image Processing, 2004. ICIP '04. 2004 International Conference on*, volume 4, pages 2351–2354 Vol. 4, Oct 2004.

[31] M. Nazzal and H. Ozkaramanli. Improved single image super-resolution using sparsity and structured dictionary learning in wavelet domain. In *Signal Processing and Communications Applications Conference (SIU), 2013 21st*, pages 1–4, April 2013.

[32] Xuan Zhang, Xiaowen Dong, and P. Frossard. Learning of structured graph dictionaries. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 3373–3376, March 2012.

[33] Ze-Min Cai and Jian-Huang Lai. Ibp-svd: A practical method for learning adaptive dictionaries for image de-noising. In *Wavelet Analysis and Pattern Recognition, 2007. ICWAPR '07. International Conference on*, volume 2, pages 641–646, Nov 2007.

[34] A. Cherian, S. Sra, and N. Papanikolopoulos. Denoising sparse noise via online dictionary learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 2060–2063, May 2011.

[35] S. Zubair and Wenwu Wang. Audio classification based on sparse coefficients. In *Sensor Signal Processing for Defence (SSPD 2011)*, pages 1–5, Sept 2011.

[36] Jiang Chen, Mu Zhichun, Zhang Baoqing, and Zhang Jin. Ear recognition via sparse representation over learned dictionary. In *Control and Decision Conference (CCDC), 2013 25th Chinese*, pages 1487–1491, May 2013.

[37] Karin Schnass and P. Vandergheynst. Dictionary learning based dimensionality reduction for classification. In *Communications, Control and Signal Processing, 2008. ISCCSP 2008. 3rd International Symposium on*, pages 780–785, March 2008.

[38] Yifeng Li and A. Ngom. Supervised dictionary learning via non-negative matrix factorization for classification. In *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*, volume 1, pages 439–443, Dec 2012.

[39] Julien Mairal, Francis Bach, Jean Ponce, Guillermo Sapiro, Andrew Zisserman, Thème Cog, Julien Mairal, Francis Bach, Jean Ponce, Guillermo Sapiro, Andrew Zisserman, Équipes projets Willow, and Ecole Normale Supérieure. Supervised dictionary learning, 2008.

[40] M.J. Gangeh, A. Ghodsi, and M.S. Kamel. Kernelized supervised dictionary learning. *Signal Processing, IEEE Transactions on*, 61(19):4753–4767, Oct 2013.

[41] Xiao-Chen Lian, Zhiwei Li, Changhu Wang, Bao-Liang Lu, and Lei Zhang. Probabilistic models for supervised dictionary learning. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2305–2312, June 2010.

[42] B.G. Vijay Kumar and I. Patras. Supervised dictionary learning for action localization. In *Automatic Face and Gesture Recognition (FG), 2013 10th IEEE International Conference and Workshops on*, pages 1–8, April 2013.

[43] M.J. Gangeh, P. Fewzee, A. Ghodsi, M.S. Kamel, and F. Karray. Multiview supervised dictionary learning in speech emotion recognition. *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, 22(6):1056–1068, June 2014.

[44] Hua Wang, Feiping Nie, Weidong Cai, and Heng Huang. Semi-supervised robust dictionary learning via efficient l-norms minimization. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1145–1152, Dec 2013.

[45] A. Zare and P. Gader. Pattern recognition using functions of multiple instances. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 1092–1095, Aug 2010.

[46] A. Zare, P. Gader, J. Bolton, S. Yuksel, T. Dubroca, R. Close, and R. Hummel. Subpixel target spectra estimation and detection using functions of multiple instances.

In *Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), 2011 3rd Workshop on*, pages 1–4, June 2011.

[47] W.R. Scott. Broadband electromagnetic induction sensor for detecting buried landmines. In *Geoscience and Remote Sensing Symposium, 2007. IGARSS 2007. IEEE International*, pages 22–25, July 2007.

[48] Yizong Cheng. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790–799, Aug 1995.

[49] Irving S. Reed and Xiaoli Yu. Adaptive multiple-band cfar detection of an optical pattern with unknown spectral distribution. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 38(10):1760–1770, Oct 1990.

[50] X. Du, A. Seethepalli, H. Sun, and A. Zare. Final report: Environmentally-adaptive target recognition systems. Technical report, TigerSense: The Machine Learning and Sensing Laboratory University of Missouri, Columbia, 2015.