LEARNING EQUIVALENT INPUT CHANNEL MAPPINGS AND GENERALIZED FEATURES FOR PATTERN TRANSFER

by

HOUTAN RAHMANIAN

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2013

Copyright © by HOUTAN RAHMANIAN 2013

All Rights Reserved

To Baha'i Students in Iran who are denied to access university-level education solely because of their beliefs.

Acknowledgements

First and foremost, I would like to greatly thank my supervising professor Dr. Manfred Huber for his continuous guidance, support, encouragement, and motivating during my graduate studies. I appreciate his invaluable advice during my Masters program, his vast knowledge in different fields, and his compassionate way to guide me in my research works. I cannot imagine finishing this thesis without his effective supervision.

Next, I would like to thank Dr. Gergely Zaruba for his help in the process of the development and testing of pressure sensing floor. I am also grateful to him for dedicating his time and energy to serve on my committee.

In addition, I wish to thank my professor Dr. Vassilis Athitsos for his productive courses I have taken, his interest in my research, and for taking time out of his busy schedule to serve on my committee.

I would also like to extend my highest appreciation to all my instructors in my undergraduate studies in the Baha'i Institute for Higher Education (BIHE). They voluntarily risked their own life to teach me in Iran, where I was deprived access to higher education only because of my beliefs. I will never forget their dedication to educate young people, and I hope I can do the same for the next generation of oppressed youth.

Last but not the least, I must express my deepest gratitude to my parents, Changiz and Nesa, without whom obtaining this degree would have been totally impossible for me. I would also appreciate my sisters and brother's support and

interest in my research, and their comments about it. My whole family has encouraged and inspired me during all my life. I am so fortunate to have them all.

July 17, 2013

Abstract

LEARNING EQUIVALENT INPUT CHANNEL MAPPINGS AND GENERALIZED FEATURES FOR PATTERN TRANSFER

HOUTAN RAHMANIAN, M.S.

The University of Texas at Arlington, 2013

Supervising Professor: Manfred Huber

In many real-world modeling applications, it is necessary to detect the origin of the patterns of the input data in addition to finding the patterns themselves. Having the input data generated by a systematically organized set of input channels (e.g. data from sensors, or variables) is very common in these applications. In addition, these input channels might also be of the same type. Therefore, the same pattern might be observed on different sets of input channels of the data, while it is caused by the same source in different localities.

Sparse coding is a very powerful method for learning high-level patterns (i.e. high-level features) from raw data input. It is capable of learning an overcomplete basis which has the capacity to capture robust and discriminative patterns within the data. However, like many other feature learning algorithms, it is unable to detect identical features or stimuli on different sets of input channels. In this work, we propose a novel method to build general features that can be applicable to different sets of channels. This succinct representational model will express the stimuli independent of the locality in which they appeared. Simultaneously, different equivalent

localities (equivalent channel sets) will be detected. As a result, when a feature is recognized on a channel set it can be transferred to the other equivalent channel sets. This enables the method to model and represent a pattern on localities where it has never been observed before.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction and Related Work

1.1 Introduction

There is an increasing need for data modeling and prediction in real-world applications. Data modeling and prediction is usually performed by extracting patterns or features from raw data inputs. In many of these applications, it is necessary to find the source and detect the patterns of the input data. On many occasions, the input data is generated by a systematically arranged set of input channels which might also be of common types (such as in the case of weather data, data from a traffic network, etc.). As a result, the source of the data can cause the same effect (i.e. the same pattern) on different input channels of the data.

In many cases, obtaining more data is expensive, difficult, time-consuming, or sometimes even impossible and the available data sets are often incomplete. Because of this data insufficiency, it would be impossible for the data to include all the existing patterns on every feasible set of input channels. Thus, using a method that requires less data is a great advantage. To address this in the context of ever increasing numbers of channels in such applications as sensor networks [1], and to be able to fully utilize the predictive power of patterns mined form data, a succinct transferable general model is needed to be able to represent these stimuli on different sets of channels.

Feature learning, in general, is the process of finding brief representations of stimuli or features in unlabeled data. These features will be found such that they can model as much variation in the raw data as possible and at the same time concisely

express the raw data. These features show how the data coming from the input channels are changing and relating together. Feature learning in raw input data and modeling the data using those patterns (instead of raw data) is a useful method for representing data and for improving its use in the learning process. When the features are found, data D can be transferred from raw data space into the feature space. If B is a linear feature basis matrix then

$$D' = B^T D$$

would be the data in feature space. The derived data D' can then be fed into arbitrary learning or data modeling algorithms. This will often improve the performance of the learning or modeling algorithm. Moreover, if domain knowledge is included in the process of feature learning, the performance of the learning method can be improved even further.

Many methods have been proposed for learning features from arbitrary raw input data. When the number of raw input channels increases, however, a limitation arises in all of the existing algorithms. They all lack the ability to generalize patterns across modalities. In other words, they fail to find a general pattern that is applicable to different channel mappings. Modality or locality of a feature is defined as a set of channels that usually perform together to form a feature. After all, data can have identical or very similar stimuli appearing on different channel sets. This fact is more common and easily comprehensible on sensory input data, where the same object or event causes the same pattern on different sets of sensory input channels. All of the existing methods, however, are inadequate at detectecting the same stimulus or feature if it appears on different channel sets.

In this thesis, we propose a method to generalize patterns over modalities or channel sets by finding the same stimulus or feature over different channel sets. For this purpose, the proposed method breaks down the features into the pattern itself and the locality or the modality mappings in which the pattern appears. In other words, to represent a stimulus, our method creates generalized features and a group of equivalent channel sets in which the generalized feature can appear. This results in a concise representational model which, in addition, provides a number of capabilities for transferring and predicting data. In particular, this model leads to a representation in which data can be transferable to other channel sets and which can be applied to any systematic data. Its main advantage is that a pattern can be detected or predicted on input channel sets where the feature has never appeared in the training data. For example, if feature f_1 is observed on channel sets set_1, set_2, set_3 and feature f_2 is only observed on set_1, then the method can also represent or predict f_2 on set_2 and set_3 in future data.

The proposed method uses the sparse coding algorithm to find local features. Then, it uses a hierarchical clustering algorithm to cluster these features into groups. Next, it generalizes the features and captures the corresponding channel mappings. The method also creates collections of channel sets. A collection of channel set contains channel sets which are mutually equivalent to each other. Later on, the method expands each generalized feature in all the channel sets that are equivalent to the ones in which the feature has been originally observed. To show the applicability of our framework, we applied it to two different experimental models: pressure sensor data, and human motion data.

This thesis is organized as follows: The remainder of this chapter covers the related work dealing with feature learning algorithms. Chapter 2 provides the background information, concepts, and formulation about feature learning with sparse

coding, feature-sign algorithm, agglomerative hierarchical clustering, and transfer learning. Chapter 3 discusses our framework, the proposed method, and its building components. Chapter 4 demonstrates the performance of the proposed method on two experimental models. Finally, Chapter 5 discusses the conclusion and the possible future work.

1.2 Related Work

Many different methods have been introduced for learning features from arbitrary raw input data. In this section, we briefly discuss the following algorithms which are closely related to the method used in this work: Principle component analysis, Independent component analysis, and Autoencoder.

Principle component analysis (PCA) [2] [3] is a common method for extracting features from data with high dimension. PCA's first use is to reduce dimensionality of highly correlated variables while keeping as much data variation as possible. PCA uses orthogonal transformation to convert correlated raw data into an uncorrelated feature basis of the data represented by the principal components of the data. Each of these principle components is a linear combination of input channels. The principle components are ordered so that the first few components explain the most variation in the raw input. Principal components can be calculated by applying singular value decomposition (SVD) on the matrix of the raw data or by applying eigenvalue decomposition on the matrix of data covariance [3]. The number of principal components (i.e. features) can never exceed the number of input channels. As a result, the PCA algorithm is unable to learn an overcomplete basis of features (i.e. having more features than the number of input channels which helps to achieve more stable, more robust, and more compact decomposition). On the other hand, since each principal component tries to cover as much as data variation as it can, principal components

are very likely to contain several patterns or stimuli. Thus, PCA is not a suitable method when the separation of the patterns is desired.

The Sparse PCA algorithm [4] is a modified version of PCA which aims to generate sparse principal components. Sparse PCA accomplishes that by adding a constraint or penalty for the number of nonzero coefficients. Therefore, Sparse PCA is more likely to extract separated patterns in principal components. The sparse principal components also make the resulting features easier to interpret for humans. Many different methods have been proposed to calculate Sparse PCA. Jolliffe et al. [4] present a LASSO-based method for calculating sparse coefficients. Hein et al. [5] propose an inverse power method for nonlinear eigenproblems to solve Sparse PCA. Zou et al. [6] formulate Sparse PCA as a regression-type problem with an elastic net regularization that can be solved. Again, all of these methods are unable to learn an overcomplete basis.

Independent component analysis (ICA) [7] [8] is another feature learning method. It assumes that data of input channels are a mixture of underlying independent sources that can not be measured directly (i.e. latent variables). As a matter of fact, ICA expects a data channel such as ch_1 to be a linear mixture of k independent sources $I_1, I_2, ..., I_k$:

$$ch_1 = a_1 I_1 + a_2 I_2 + ... + a_k I_k$$

The ICA algorithm splits the raw input signals into its independent building subcomponents ($I_{1...k}$). In other words, ICA separates the sources of the input. However, it fails to do so if the number of sources is larger than the number of input channels. That means ICA, like PCA and Sparse PCA, suffers from the limitation that it can not find more features than the dimensionality of the input data. This limitation of the number of features makes the features less specific. In this condition, the features

try to explain as much data as they can. As a result, a feature may represent the effect of several stimuli which is not desired in the case of interest in the separation of sources.

The autoencoder algorithm [9] is another feature learning method that uses multilayer neural networks to reconstruct input data. This process leads the hidden layer to learn features of the input data. Assume $x_1, x_2, ..., x_n$ are n input channels. Autoencoder assigns these inputs to both input and desired output of a multilayer neural network. Therefore, in the learning process (e.g. back-propagation algorithm) of the neural network, neurons of the hidden layer learn m features $f_1, f_2, ..., f_m$ so that they can reconstruct the input in the output layer. Figure 1.1 shows a simple autoencoder neural network.

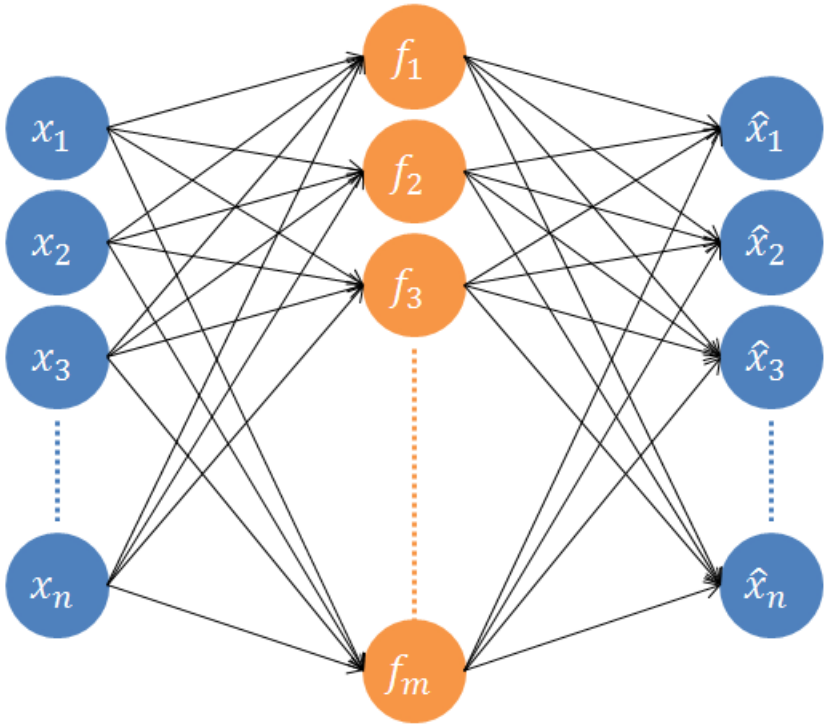


Figure 1.1: Concept of an autoencoder

Since there is no constraint on m, the autoencoder, unlike PCA and ICA, can learn an overcomplete basis of features. This provides the autoencoder the ability to have more specific features. Expanding on this, the sparse coding algorithm [10] is another feature learning algorithm which finds a basis that enables it to express the data using sparse coefficients. Like autoencoders, it can learn an overcomplete basis. Sparse coding is discussed in more detail in the next chapter.

When the number of raw features increases, however, a limitation arises in all of the discussed algorithms. They all lack the ability to generalize patterns across modalities. In other words, they fail to find a general pattern that is applicable to different channel mappings. Modality or locality of a feature is defined as a set of channels that usually perform together to form a feature. After all, data can have identical or very similar stimuli appearing on different channel sets. This fact is more common and easily comprehensible on sensory input data, where the same object or event causes the same pattern on different sets of sensory channels. All of the above methods however, are unable to detect the same stimulus or feature if it appears on different channel sets.

Chapter 2

Technical Background

In this chapter, we introduce the concept of transfer learning, feature learning using efficient sparse coding, the feature-sign search algorithm, and hierarchical clustering. These algorithms are later used in our proposed method.

2.1 Transfer Learning

In most data modeling or machine learning algorithms, it is usually assumed that training data completely represents the whole feature space. In fact, they also consider the test data or upcoming data to be from the same feature space and distribution. However, this assumption is not always true. In real-world applications, obtaining more training data is sometimes expensive, difficult, and even impractical. Therefore, these applications have to use training data which do not fully represent the feature space of their problems.

In these situation, transferring acquired knowledge from a similar or related domain to the current domain can improve the performance of the modeling or learning process. Transfer learning [11] is the process of transferring knowledge to new related domains to avoid expensive data collecting and, also using the transferred knowledge to boost the performance. This knowledge will be used along data of the new domain for learning as it is shown in Figure 2.1.

To evaluate the performance of the learning process, we will use the graphical representation of the performance increase (i.e. learning curves). The knowledge transfer ideally should cause the learning curve to have a higher start, to learn faster

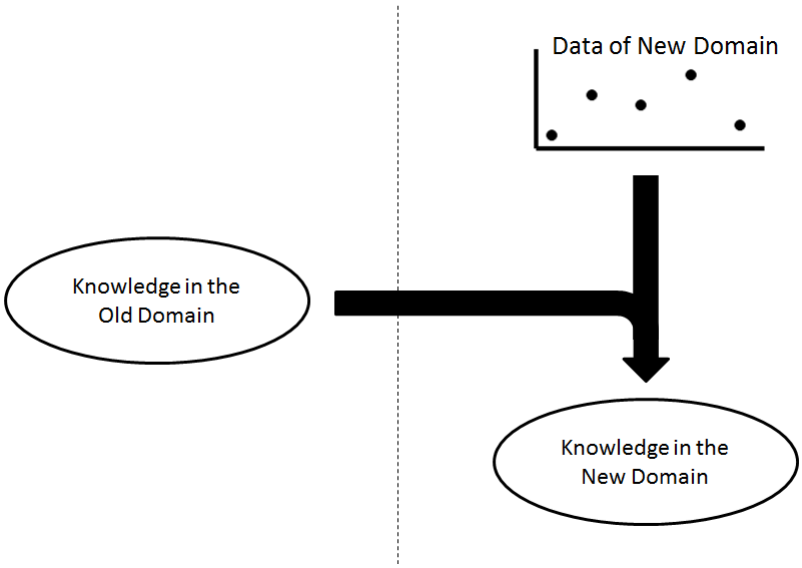


Figure 2.1: Transfer Learning

(i.e. higher slope), and to have a better final performance (i.e. higher asymptote) compared to a situation in which knowledge transfer has not been used[12]. Figure 2.2 displays two learning curves: one without knowledge transfer(orange), and the learning curve for the same task by using transfer learning (black).

2.2 Sparse Coding

Sparse coding [13] is a technique for learning high-level features from the given unlabeled training data. The goal of sparse coding is to discover features from the data which can be used by sparse coefficients to reconstruct the data. In other words, the sparse coding method tries to find the feature matrix $F \in \mathbb{R}^{l \times m}$ and the sparse coefficient matrix $S \in \mathbb{R}^{m \times n}$ such that the input data matrix $D \in \mathbb{R}^{l \times n}$ can be expressed as follow:

$$D \approx FS$$

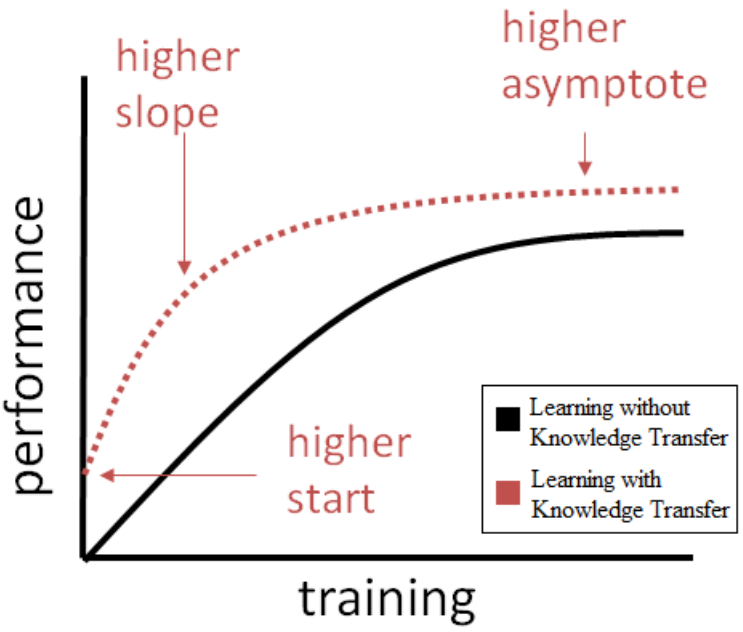


Figure 2.2: Learning curves with and without knowledge transfer

Unlike all the feature learning algorithms we discussed in the previous chapter, sparse coding can learn an overcomplete basis which means it can learn a basis with a larger number of features than the original input dimensions [13]. This fact enables the sparse coding algorithm to capture a large number of features in the data. This brings with it the ability to encode the same part of the data in different ways in order to obtain discriminative capabilities between different classes and, in particular, to obtain patterns that reflect the signature of particular events or causes.

As mentioned, sparse coding requires coefficients to be sparse. Different functions can be used as sparsity constraint functions to achieve this purpose. Below are some of these penalty functions:

$$\begin{cases} \|S\|_1 & L_1 \text{ norm penalty function} \\ (s^2+\epsilon)^{\frac{1}{2}} & epsilonL_1 \text{ penalty function} \\ log(1+s^2) & log \text{ penalty function} \end{cases}$$

Among these, L_1 norm is the most commonly used sparsity constraint function. The sparse coding's goals can be achieved by solving the following optimization problem:

$$argmin_{F,S}\|D - FS\|_2^2 + \alpha\|S\|_1$$

subject to

$$\forall j \qquad \sum_i F_{i,j}^2 \preceq constant$$

where α is the trade-off constant between sparsity of the coefficients and minimum residuals.

Lee et al. [10] proposed a method to iteratively solve this optimization problem by dividing it into two steps: optimizing the problem over S while keeping F locked, and optimizing over F while keeping S locked. This will reduce the non-convex optimization problem to two convex optimization problems. The proposed method iteratively solves these two convex optimizations.

The optimization problem over F is a least squares problem with quadratic constraint. It can be solved by a generic convex optimizer or by iterative gradient-based methods, but both of these methods have shown slow execution or slow convergence. However, in this method the optimization problem is efficiently solved using its Lagrange dual. The optimization problem over coefficients S, on the other hand, is an L_1-regularized least square problem. Lee et al. [10] suggest a new method to solve this optimization problem using an approach called feature-sign search algorithm. In this algorithm, the optimization for the objective of the residual and the sparsity

penalty , while iteratively keeping F and S constant, is proved to be strictly decreasing. Therefore, it is guaranteed that the optimization problem will converge to the global optimum in a finite number of iterations.

2.2.1 Learning Features: Lagrange Dual

Having the coefficient S constant, the optimization problem reduces to minimizing:

$$\|D - FS\|_2^2$$

subject to

$$\forall j \qquad \sum_i F_{i,j}^2 \preceq constant$$

This new optimization problem is a least square one with quadratic constraints, which can be solved by its Lagrange dual. The Lagrangian is first calculated as follows:

$$L(B, \vec{\lambda}) = trace((D - FS)^T(D - FS)) + \sum_{j=1}^{n} \lambda_j (\sum_{i=1}^{k} F_{i,j}^2 - constant)$$

where $\lambda_j >= 0$ is a dual variable. The Lagrange dual can then be computed by analytically minimizing over F.

$$Dual(\vec{\lambda}) = min_F L(B, \vec{\lambda})$$

$$= trace(D^T D - DS^T(SS^T + \Lambda)^{-1}(DS^T)^T - constant\Lambda)$$

where

$$\Lambda = diag(\vec{\lambda})$$

Next, the gradient and Hessian of the Lagrange dual are computed. The Lagrange dual can then be optimized, and the optimal features F can be analytically achieved in this manner:

$$F^T = (SS^T + \Lambda)^{-1}(XS^T)^T$$

Optimizing over a basis with this approach is significantly more efficient than the original optimization problem.

2.2.2 Learning Coefficients: Feature-Sign Search Algorithm

In the optimization process over coefficients S, while the feature F is fixed, the optimization reduces to smaller optimization problems over individual features. If $\vec{F_j}$ is a feature vector from the features matrix (basis), and $\vec{D}_i$ and $\vec{S_i}$ are respectively an input vector and a sparse coefficient vector, then the optimization problem looks like:

$$argmin_{\vec{S_i}} \|\vec{D}_i - \sum_j \vec{F}_j S_{i,j}\|^2 + \alpha \sum_j |S_{i,j}|$$

As the above formula shows, the sparsity constraint adds the absolute value of each coefficient $|S_{i,j}|$ to the cost function. If the sign of $S_{i,j}$ in the optimal point of the cost function was known ahead of time, the absolute term could be replaced by $S_{i,j}$ (for $S_{i,j} > 0$), $-S_{i,j}$ (for $S_{i,j} < 0$), or 0 (for $S_{i,j} = 0$). Then, the problem could be reduced to an unconstrained quadratic optimization problem.

The feature-sign search algorithm [10] guesses the sign of $S_{i,j}$ and substitutes the absolute value of coefficients with $S_{i,j}$, $-S_{i,j}$, or 0. The feature-sign search algorithm continuously improves its guess to reduce the cost function in a sequence of "feature-sign steps". Feature-sign search algorithm keeps a set of active coefficient (i.e. nonzero coefficient) and assumes that all the other coefficients are zero. Moreover, it saves

the matching sign for the active set. Then, it searches for optimal active set and matching signs in "feature-sign steps". To clearly explain how this algorithm works, the following optimization problem is substitute with the original one:

$$argmin_x f(x) \equiv \|y - Ax\|^2 + \gamma\|x\|_1$$

The approach of feature-sign algorithm is as follows: having an initial active set and their matching signs, it solves the unconstrained quadratic optimization problem. Then, it runs an efficient discrete line search between the newly calculated solution $\hat{x}_{new}$ and the old solution to update the active set and their matching signs. Algorithm 1 illustrates how the process of updating active set and their signs θ in the feature-sign search algorithm.

Although the initial active set and matching signs are not optimal at the beginning of the algorithm, it is guaranteed that the "feature-sign step" consistently decreases the objective value. In fact, It is proven that the feature-sign search algorithm converges to the global minimum of the cost function in a finite number of steps [10].

2.3 Hierarchical Clustering

Clustering is an unsupervised learning method. It categorizes objects into groups or clusters based on a defined similarity metric. The goal is to build clusters such that objects of a cluster are more similar to each other than to the objects of the other clusters. Many algorithms have been suggested for this purpose. K-means clustering is the most popular clustering algorithm. However, its result greatly depends on the number of clusters. Therefore, it usually assumes that the number of

Algorithm 1 The Feature-Sign Search Algorithm

Step 1:
$x := \vec{0}$, $\theta := \vec{0}$, *active set* $:= \{\}$

Step 2:
select $i = argmax_i |\frac{\partial \|y - Ax\|^2}{\partial x_i}|$ from zero coefficients of x
if $\frac{\partial \|y - Ax\|^2}{\partial x_i} > \gamma$ **then**
  $\theta_i := -1$, *active set* $:= \{i\} \cup$ *active set*
else if $\frac{\partial \|y - Ax\|^2}{\partial x_i} < -\gamma$ **then**
  $\theta_i := 1$, *active set* $:= \{i\} \cup$ *active set*
end if

Step 3:
$\hat{x}_{new} := (\hat{A}^T \hat{A})^{-1}(\hat{A}^T y - \gamma\hat{\theta}/2)$ (Analytical solution)
Execute the discrete line search between $\hat{x}$ and $\hat{x}_{new}$.
Examine the objective value at $\hat{x}_{new}$ and all the points where a sign changes and renew the $\hat{x}$ to the point that has the optimal objective value.

Step 4:
if $\frac{\partial \|y - Ax\|^2}{\partial x_j} + \gamma\theta_j = 0$ for $\forall x_j \neq 0$ **then**
  Go to Step 3.
else if $|\frac{\partial \|y - Ax\|^2}{\partial x_j}| \preceq \gamma$ for $\forall x_j = 0$ **then**
  Go to Step 2.
else
  Return x as solution.
end if

clusters, k, is already known in advance. Then, it tries to put objects in those k clusters such they fit into the best possible groups.

In contrast, hierarchical clustering [14] is a clustering algorithm that offers a very flexible and non-parametric technique. It does not require the number of clusters to be known in advance. The hierarchical clustering algorithm does not require prior knowledge about the domain of the data. It creates a hierarchical cluster tree. In fact, It generates a hierarchy or links between objects based on the similarity metric and the inconsistency that an object might cause be joining a cluster. For example,

Figure 2.3 displays a sample of such hierarchical cluster trees for clustering 30 random numbers based on their distance from each other.

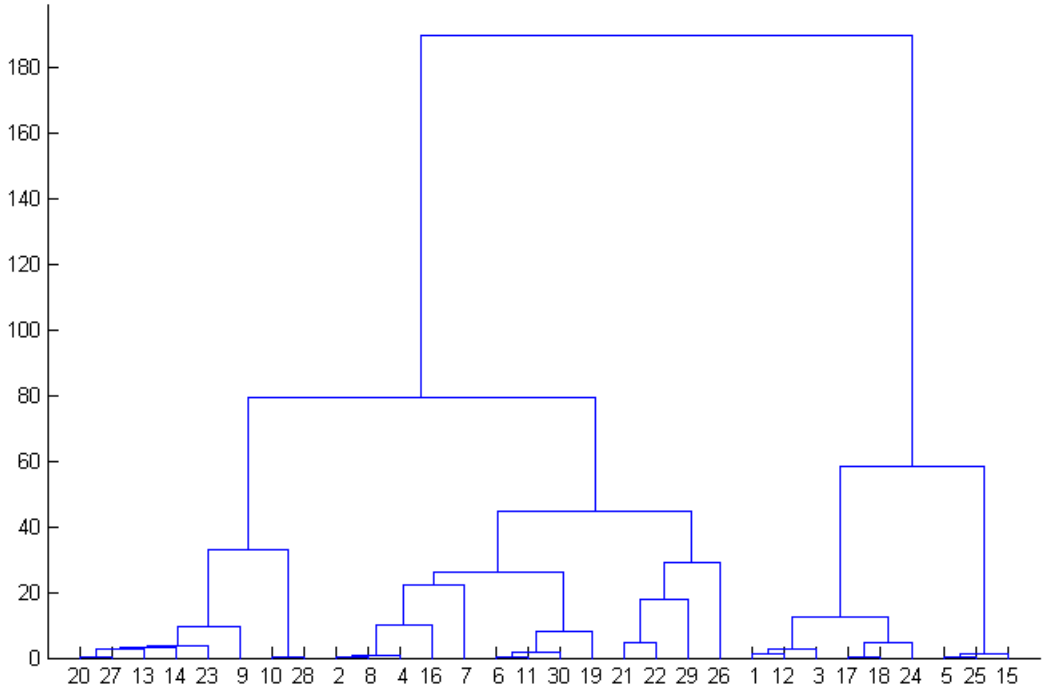


Figure 2.3: A sample hierarchical cluster tree in hierarchical clustering algorithm

When the hierarchical cluster tree is formed, the algorithm uses a criterion to cut off the tree and form the clusters. The algorithm needs to know how different two clusters are. In that way, the algorithm can stop linking those two clusters if they are more different than they are expected to be. A measure called inconsistency coefficient is defined for each link in hierarchical cluster tree. An inconsistency coefficient of a link is a comparison between the height of the link and the height of the links below it. If the height of the link is approximately equal to the height of links below it, it

indicates that there is no specific division between the objects joined in this hierarchy level. Otherwise, objects joined in this level are far apart each other to form a cluster.

Therefore, a common criterion for cutting off the hierarchical cluster tree is an inconsistency threshold. That means a node and all its subnodes will arrange a cluster when all the inconsistency coefficients of their links are less than a specified threshold.

2.3.1 Agglomerative Hierarchical Clustering

Agglomerative hierarchical clustering [7] is one type of hierarchical clustering algorithms. It uses a bottom-up approach to generate the hierarchical cluster tree. It initially puts each object in its own cluster. A similarity metric between two clusters is then defined. After that, at each level the algorithm merges the two most similar clusters together and forms a new cluster. Therefore, at each level the algorithm will contain one less cluster than at the previous level. It basically builds a hierarchy of clusters where in each level the variance within the clusters is slightly higher than those at the level before but where there are fewer clusters. Finally, the algorithm stops merging clusters when it reaches a defined criterion for the minimum similarity within a cluster. Algorithm 2 clarifies how the agglomerative hierarchical clustering algorithm works.

Algorithm 2 Agglomerative Hierarchical Clustering

Step 1:
Calculate the dissimilarity values between every two objects
Step 2:
Create a cluster from the two closest objects or clusters
Step 3:
Determine the dissimilarity value between new cluster and existing objects or clusters.
Step 4:
Return to **Step 2** unless the stopping criterion is met.

Chapter 3

Framework

In the proposed method, a feature learning component first learns the local features from the raw input data. Then, the feature learning component passes the learned local features to a feature generalizer component. The feature generalizer component breaks down the local features into the features and the locality in which they appear. A feature is a vector of numbers that can appear on arbitrary set of input channels. On the other hand, a locality is an ordered set of input channels that perform together. Different features can appear on a locality, and a feature can appear on different localities.

After breaking down the local features to the features and their localities, a clustering component in the feature generalizer clusters the features into groups. The feature generalizer later analyzes each cluster to generate a generalized feature.

During the generalization process, the feature generalizer uses the locality of clustered features to form channel mapping sets which represent where generalized features are applicable. These channel mapping sets will be sent to the equivalent channel mapping repository. The equivalent channel mapping repository is responsible for extracting equivalent channel mappings from the submitted sets and saving them in supersets. Extracting equivalent channel mappings enables us to transfer our knowledge from a channel mapping to its equivalent channel mappings. In this way, features which are learned in a specific locality are transferred to other equivalent localities.

The generalized features and channel mapping supersets can later be used for data reconstruction, modeling or prediction. In the following sections, we will explain how each component is architected and works. However, it is important to notice that the framework is designed to be flexible. One might use a different overcomplete feature learner component, but keep the feature generalizer the same. Also, a different classification algorithm might be leveraged in the feature generalizer component. Figure 3.1 shows the overall schema of our proposed framework.

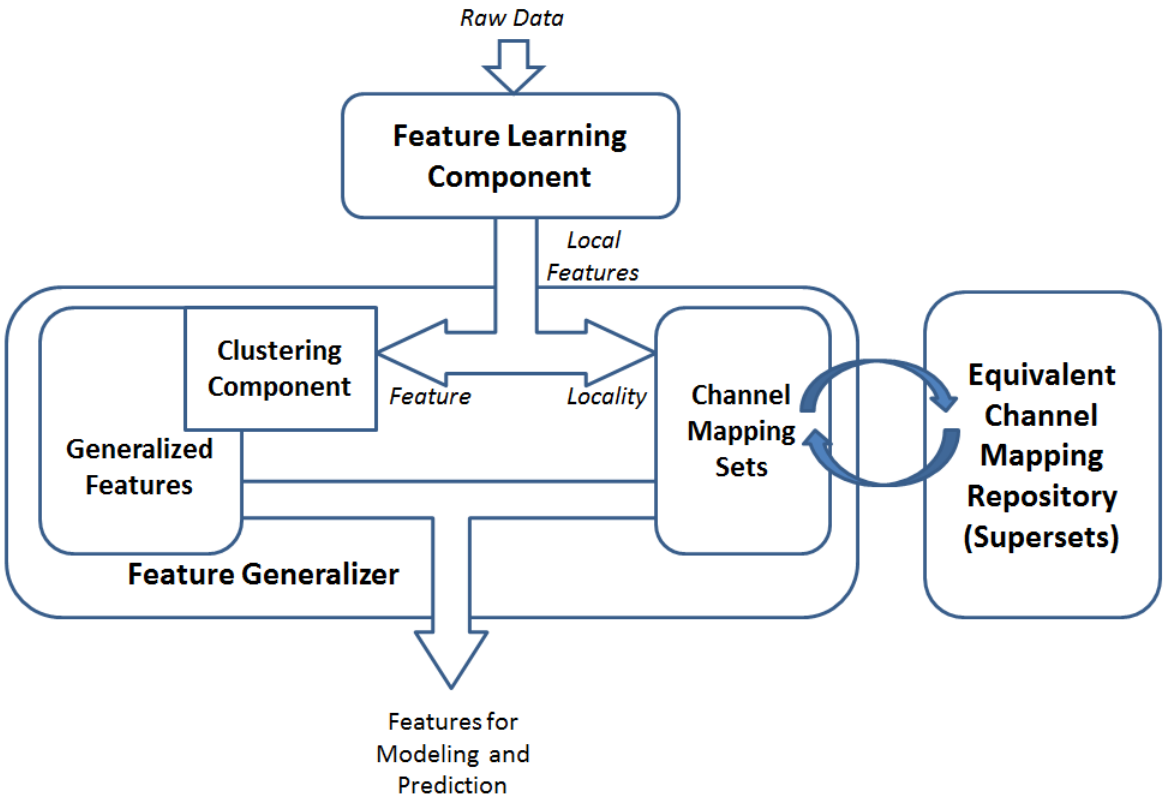


Figure 3.1: The overall schema of the proposed framework

3.1 Feature Learning Component

An efficient sparse coding algorithm [10] has been used here for local feature learning. The ability of the sparse coding algorithm to learn an overcomplete basis results in sparse features where, ideally, each explains a single source or stimulus. When the local features are learned, the channels in each local feature are divided into two sets: the Active Channel Set (ACS) and the Inactive Channel Set (ICS). This classification is performed based on the contribution of the channels to the feature vector size. $c_1, c_2, ..., c_n$, and $v_1, v_2, ..., v_n$ are respectively assumed to be the channels and the values assigned to them in feature f. If

$$1 - \frac{\sqrt{v_1^2 + ... + v_{i-1}^2 + v_{i+1}^2 + ...v_n^2}}{\sqrt{v_1^2 + ... + v_n^2}} > THR$$

where THR is the contribution threshold, then

$$c_i \in ACS$$

otherwise

$$c_i \in ICS$$

Since the members of the Inactive Channel Set (ICS) do not play a significant role in the feature, the algorithm then ignores their contribution to the feature. Therefore, a vector called active-channel vector can be constructed for each feature. An active-channel vector a_l is defined for active channels of each feature f_l. If $\vec{c_l} = \{c_{l_1}, c_{l_2}, ..., c_{l_m}\}$ are the members of the Active Channel Set (ACS) in feature

f_l , and $\vec{v_l} = \{v_{l_1}, v_{l_2}, ..., v_{l_m}\}$ are the values of the feature f_l in those channels, then the active-channel vector a_l is defined as follows:

$$a_l = [(c_{l_1}, v_{l_1}), (c_{l_2}, v_{l_2}), ..., (c_{l_m}, v_{l_m})]$$

The active-channel vector is a simple representation for sparse features. It should be noted that in this representation, features can have different numbers of non-zero elements (m). Thus, the length of the active-channel vectors are not necessarily equal.

3.2 Feature Generalizer Component

The feature generalizer component can be divided into its subcomponents. The feature generalizer component contains a clustering subcomponent that clusters active-channel vectors into groups. It also has a generalizing subcomponent that generates a general feature that is applicable to different channel mappings. The feature generalizer component as a whole also communicates with the equivalent channel mapping repository to create and use supersets of channel mapping to form generalized features.

3.2.1 Clustering Subcomponent

To compare each pair of active-channel vectors and collect them into clusters, a distance measure is introduced. The distance measure should be able to find possible similar features over arbitrary channels. Thus, the distance metric between two active-channel vectors is defined as the least square distance between all possible permutations of their channel mappings. In other words, if the a_i and a_j are two active-channel vectors, the distance between them is the minimum square distance

between all possible permutations of $\vec{v_i}$ and $\vec{v_j}$. For example, if $\vec{v_i}$ and $\vec{v_j}$ are respectively $\{4, 19, 10\}$ and $\{21, 2, 8\}$, the distance between a_i and a_j is 12. The channel mappings that result this distance are $\{c_{i_1}, c_{i_2}, c_{i_3}\}$ and $\{c_{i_2}, c_{i_1}, c_{i_3}\}$

Due to possible inequality in the length of active-channel vectors, dummy zero-valued channels are added to the shorter active-channel vector. As a result, active-channel vectors would always be of the same size at the time of comparison.

To cluster these active-channel vectors, each pair of features is evaluated for the distance measurement. Algorithm 3 illustrates how the distances between the features and their minimum distance channel mappings are computed.

Algorithm 3 Distance Measurement

Require: Active-Channel Vectors $a_0, a_1, ..., a_n$
  for all Pairs of Active-Channel Vectors (a_i, a_j) **do**
    Make $\vec{v_i}, \vec{v_j}$ the same size
    for all Permutations of $\vec{v_j}$ **do**
      Distance $=(v_{i_1} - v_{j_1})^2 + ... + (v_{i_m} - v_{j_m})^2$
      if Dist(i,j) > Distance **then**
        Dist(i,j) = Distance
        Mappings(i,j) = mappings of $\vec{c_i}$ and $\vec{c_j}$
      end if
    end for
  end for
  return Dist and Mappings:
  $Dist(i, j)$ = Distance between Active-Channel Vectors i, j
  $Mappings(i, j)$ = Channel mappings of Active-Channel Vectors i, j when the minimum distance is computed

Having obtained the distance values, the active-channel vectors will be organized into clusters. A hierarchical clustering [7] algorithm has been used for this purpose. This hierarchical clustering algorithm operates with an agglomerative (bottom-up) strategy. First, all the features are made to form their own clusters. Then two clusters are repeatedly selected to be joined together to form a new cluster until a

certain stop criterion is met. To evaluate the quality of the clusters, this framework reconstructs the data and uses the relative reconstruction error as a stop criterion for the clustering algorithm. Merging the clusters stops when the relative reconstruction error passes a certain threshold.

3.2.1.1 Generalizing Subcomponent

A general feature has to be clearly defined in order to explain the generalizing process. Each cluster describes a generalized feature. A generalized feature is represented with a value-vector

$$VV = [v_1, v_2, ..., v_n]$$

and a set of channel mappings

$$CM =$$

$$\{[c_{1_1}, c_{1_2}, ..., c_{1_n}], [c_{2_1}, c_{2_2}, ..., c_{2_n}], ..., [c_{m_1}, c_{m_2}, ..., c_{m_n}]\}$$

To represent a cluster in this form, we need to propose an approach to find the value-vector, VV, and the set of channel mapping, CM, for the general feature. For this purpose, a cluster prototype has been defined for each cluster. The cluster prototype of the cluster X is an active-channel vector a_i such that $a_i \in X$; where a_i has the minimum sum of distances to all other members of the cluster X. In fact, the value-vector VV of the general feature assigned to the cluster is set to the values of the cluster prototype. Also, the set of channel mappings CM in the general feature assigned to the cluster is initiated to the channel mappings of all the cluster members.

After that, sets of channel mappings for each cluster will be transmitted to the equivalent channel mapping repository unit for knowledge transfer purposes. The

Table 3.1: A sample of the process of superset generation in equivalent channel mapping repository

Transmitted Channel Mappings	Generated Supersets in the Repository
$\{[c_2, c_4, c_5, c_6, c_8], [c_5, c_{11}, c_8, c_7, c_4],$ $\boldsymbol{[c_2, c_7, c_5, c_4, c_6]}, [c_{14}, c_7, c_{11}, c_{10}, c_d],$ $[c_{11}, c_5, c_8, c_9, c_7], [c_{14}, c_{10}, c_{11}, c_{12}, c_8]\}$, $\{[c_4, c_8, c_{11}, c_9, c_8], [c_3, c_{11}, c_8, c_7, c_2],$ $[c_9, c_7, c_2, c_4, c_{12}], [c_{14}, c_7, c_{14}, c_{10}, c_d],$ $\boldsymbol{[c_2, c_7, c_5, c_4, c_6]}\}$, $\{[c_3, c_9, c_{10}, c_4, c_6], [c_6, c_{11}, c_1, c_7, c_2],$ $[c_8, c_7, c_2, c_4, c_{12}], [c_{10}, c_{11}, c_{14}, c_{13}, c_1],$ $[c_1, c_3, c_d, c_9, c_4], [c_1, c_2, c_3, c_4, c_5]$ $[c_{16}, c_{15}, c_{14}, c_{13}, c_{12}]\}$	$\{\boldsymbol{[c_2, c_7, c_5, c_4, c_6]}, [c_2, c_4, c_5, c_6, c_8],$ $[c_5, c_{11}, c_8, c_7, c_4], [c_{14}, c_7, c_{11}, c_{10}, c_d],$ $[c_{11}, c_5, c_8, c_9, c_7], [c_{14}, c_{10}, c_{11}, c_{12}, c_8],$ $[c_4, c_8, c_{11}, c_9, c_8], [c_3, c_{11}, c_8, c_7, c_2],$ $[c_9, c_7, c_2, c_4, c_{12}], [c_{14}, c_7, c_{14}, c_{10}, c_d]\}$, $\{[c_3, c_9, c_{10}, c_4, c_6], [c_6, c_{11}, c_1, c_7, c_2],$ $[c_8, c_7, c_2, c_4, c_{12}], [c_{10}, c_{11}, c_{14}, c_{13}, c_1],$ $[c_1, c_3, c_d, c_9, c_4], [c_1, c_2, c_3, c_4, c_5]$ $[c_{16}, c_{15}, c_{14}, c_{13}, c_{12}]\}$

channel mapping repository forms supersets of transmitted channel mappings by merging equivalent channel mapping sets. Two sets of channel mappings are equivalent when they share at least one exactly identical channel mapping. Table 3.1 shows sample transmitted channel mappings and generated supersets in the equivalent channel mapping repository (c_d stands for dummy channels).

The generated supersets in the equivalent channel mapping repository will be sent to the feature generalizer. These supersets represent the interchangeable channel mappings or localities. The feature generalizer uses these supersets to its advantage to transfer the extracted general patterns to similar localities. As a matter of fact, if one pattern is observed on one of the channel mappings of the superset SS, it theoretically can be observed on other channel mappings of the superset SS as well. Therefore, the feature generalizer uses the corresponding superset SS to replace the initial channel mappings in CM. These new channel mappings, CM_{new}, in addition to the extracted value vector VV will shape the generalized feature.

3.3 Evaluation Methodology

3.3.1 Simple Modeling Evaluation

We need an evaluation method to evaluate the performance of the proposed method. This evaluation method should be able to examine how good the extracted features are in representing the raw data, and how compact the extracted features are. In fact, good features are expected to be general and compressed, but not too compact that they can not represent the data.

In order to facilitate evaluation, we have defined two measures for examining the performance of the proposed model. The first one is a representational performance measure to evaluate the quality and the performance of the model in representing the data. The second one is a compactness measure or generality measure to evaluate how compressed the extracted features are.

The representational performance measure uses raw data reconstruction to execute the evaluation. Local features are first expressed by the value vector of the generalized feature (VV) and the existing channel mappings in that generalized feature (CM). Then, the coefficients $s_1, s_2, ..., s_n$ are found for the channel-remapped general features $f_1^{(r)}, f_2^{(r)}, ..., f_n^{(r)}$ using the feature-sign search algorithm [10]. These coefficients are used to reconstruct the original data using the channel-remapped general features. It should be noted that in this test, no knowledge transfer would be used because training data and test data are the same.

The generalized features can then be examined for their performance in the reconstructed data. The reconstruction error e can be computed as follows:

$$e = D - \sum_{i=1}^{n} f_i^{(r)} s_i$$

Moreover, the average reconstruction error per reading can also be computed as follows knowing the number of input training vectors and their dimensionality:

$$avgE = \frac{\sum_{i=1}^{p}\sum_{j=1}^{n}\frac{|d_{ij} - r_{ij}|}{\max_i d_{ij} - \min_i d_{ij}}}{p * n}$$

where d_{ij} is the original data, r_{ij} is the reconstructed data, p is the input row count (# of training vectors) and, n is the dimensionality of the training vectors. The same measure can evaluate the quality of other methods (e.g. the PCA algorithm) by performing on the data which is reconstructed by the features of a specific method.

This reconstruction error shows how good our developed model is in representing the data. As a matter of fact, it has been used for developing the stop criterion for the demonstrated hierarchical clustering. The clustering algorithm stops combining two clusters when the relative construction error passes a certain threshold:

$$\delta(e) = \frac{\Delta(e)}{e} \qquad > MaxRelError$$

Relative error can be used as stop criterion since it basically shows that there has been a large increase in reconstruction error caused by the combination of the last two clusters.

On the other hand, to evaluate the compactness of the features, we have defined a compactness measure. The number of elements that describe features has been defined as the compactness measure. Again, this measure can evaluate different methods. For example, the number of nonzero elements in the principal components can be simply counted and be used as compactness measure for the PCA algorithm.

However, calculating the number of elements that are used for expressing the features in the proposed method is more complicated. Each generalized feature, is

represented by a value-vector and a set of channel mappings. If the length of a value-vector of a feature is n and its channel mappings set has m members, then that feature can approximately be described by $n * (m + 1)$ elements: n elements to represent the value vector, and nm elements to describe the channel mappings. It is important to note that elements that are used for describing channels are simple integers, therefore they can be compressed. In fact, a channel mapping element uses less memory than a value-vector element. Since the number of the channel mappings' elements are dominant, the memory used for the extracted features is greatly affected.

For evaluating the performance of the proposed method, this work compares the proposed method with the PCA algorithm. To do so, principal components are used to reconstruct the data and compute the average reconstruction error. The number of features (principal components) used at each experiment is equal to the number of features used in our proposed method at the same experiment. However, it should be noted that the features in the proposed method can be applied to different localities, while the principal components are local and therefore non-transferable.

3.3.2 Knowledge Transfer Modeling Evaluation

Evaluating the performance of the model with knowledge transfer is similar to the simple modeling. However, the feature learning is executed on one dataset, and evaluation is performed on another dataset. The evaluation dataset potentially can have different features than the feature learning dataset. These different features are ideally the same pattern in a different locality.

In this evaluation, the local features will be extracted from the training dataset. The locality of features will then be separated from its value vector. Clusters will be formed on the features and channel mapping sets will be transferred to the equivalent channel mapping repository to shape supersets. Then, supersets will be transmitted

back to the generalizer to form generalize features in combination with clusters' value vectors. The generated general features will be used to reconstruct the testing dataset. The performance of this modeling will be evaluated by the same measures introduced for simple modeling.

Chapter 4

Experiments and Results

To demonstrate the applicability and ability of our framework to model and represent the data using the channel-remapped features, we conducted experiments for modeling with and without knowledge transfer. Experiments on modeling without knowledge transfer are aimed at evaluating how good the data can be represented by general features. Two experimental models have been proposed for this purpose: a pressure sensing floor, and human motion. Experiments on modeling with knowledge transfer, on the other hand, evaluate the performance of generalized features in the context of transfer learning by applying them on equivalent localities. This test is performed using a simulation on a larger pressure sensing floor.

4.1 Modeling without Knowledge Transfer

4.1.1 Pressure Sensing Floor

In order to show that our framework forms a generalized feature to represent similar features over different channels, we have applied it on an experiment which is known to have identical features over different channels. A 4×7 foot floor was built, and 16 sensors were placed under the floor. Figure 4.1 and Figure 4.2 respectively show the pressure sensing floor (SmartFloor) and how the sensors were placed under the floor.

Pressure sensor readings were captured from a single human subject walking on the prototype floor under observed conditions. It is known that the sensor readings should be formed from very similar features over different channels due to the nature of

Figure 4.1: Pressure sensing floor (SmartFloor)

the pressure sensory system which has identical sensor configurations (sensor spacing and type) in different regions of the floor. The captured data contains more than 90,000 input vectors. The local features were learned from the generated data using the sparse coding algorithm [10]. Figure 4.3 displays the learned features. It can be observed that there are very similar features over different sensory channels due to the location of the human subject on the floor.

These local features have been used to build generalized features. In this experiment, the algorithm built 5 general features from the 19 local features extracted by the sparse coding algorithm. Table 4.1 shows one of these general features.

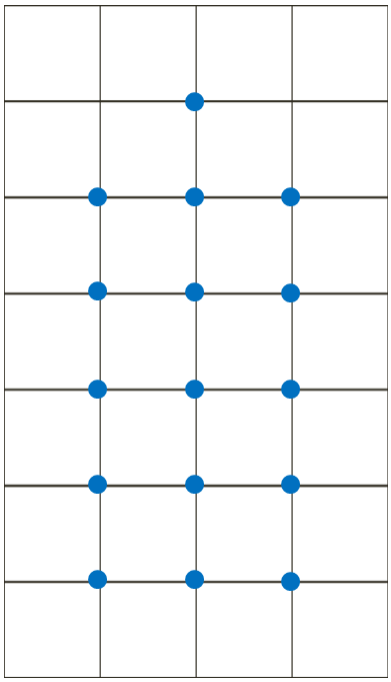


Figure 4.2: The placement of the pressure sensors under the floor

Once the general features have been created, they can be used to reconstruct the local features by remapping the general feature to the local channels. Figure 4.4 shows the reconstructed local features from the general features.

Then, the channel-remapped features were used to reconstruct the original sensory data. The sensory data were reconstructed with a 4.35% average reconstruction error. Once constructed, the model is also able to create or predict a local feature in a place p_1 where the feature has not been seen. This is possible by transferring consistent channel mappings from another feature which shared the place p_2 (which also appeared in the original feature) in its channel mappings and also appeared at p_1. Given this, the local features can be applied to the new place p_1 where it has not occurred in the dataset.

To put the average reconstruction error in context, we performed the PCA algorithm on the same training dataset. The number of features (principal components)

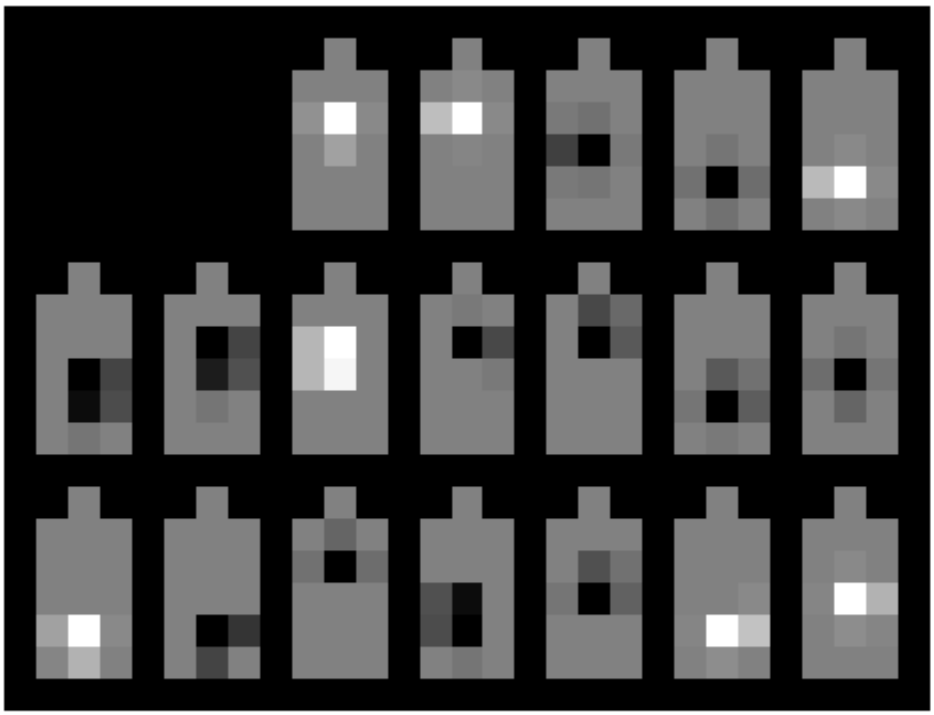


Figure 4.3: The learned local features of the pressure sensing floor

used here is equal to the number of features used in our proposed method. In fact, 5 principal components were used here to model data.

Our proposed method has described its general features with 100 nonzero elements. In contrast, PCA has used 80 nonzero elements to describe its features. These principal components were then used to reconstruct the data. The data was reconstructed with a 9.63% average reconstruction error. In comparison, our proposed method showed much better performance with the same number of features while the feature compression was almost equal.

Table 4.1: A sample generalized feature learned from the pressure sensing floor

A sample generalized feature	
$ValueVector$	[0.064078, 0.062151, 0.907050, 0.404215, 0.059600]
$ChannelMappings$	$\{[c_2, c_4, c_5, c_6, c_8]$, $[c_5, c_{11}, c_8, c_7, c_4]$, $[c_2, c_7, c_5, c_4, c_6]$, $[c_{14}, c_7, c_{11}, c_{10}, c_d]$, $[c_{11}, c_5, c_8, c_9, c_7]$, $[c_{14}, c_{10}, c_{11}, c_{12}, c_8]\}$

4.1.2 Human Motion

Humans are able to perform simple and primitive actions. They have learned to combine these primitives simultaneously and consecutively to build more complex actions to perform. The Human Motion Database [15][16] is a database containing a single human subject performing 250 simple common daily actions of human beings. This database contains some of the building blocks or primitives of human motion. The human motion data in this database is expressed through 22 human joints. Each joint angle is represented by separate rotations along the x, y, z coordinate axes. Therefore, the data is 66-dimensional. Due to relative symmetry in the human body, it is expected that there are nearly identical features in human motion data over different channels (i.e. over different sets of joints).

We have used over 700,000 input vectors from this database and have learned 300 local features from those input vectors using the sparse coding algorithm. Local features are then fed into our method for generalization and construction of their

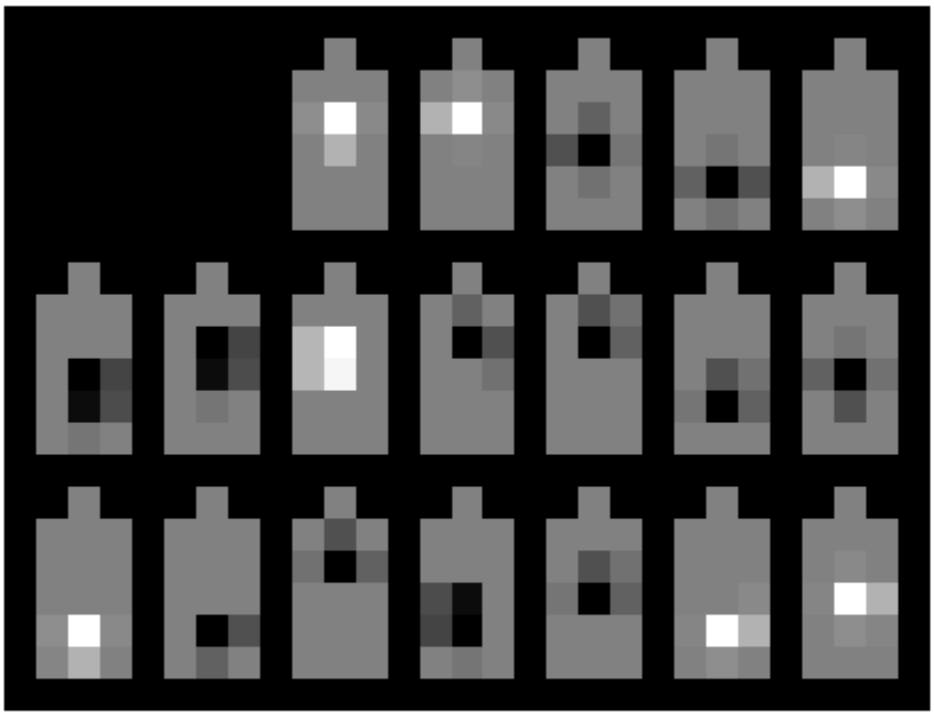
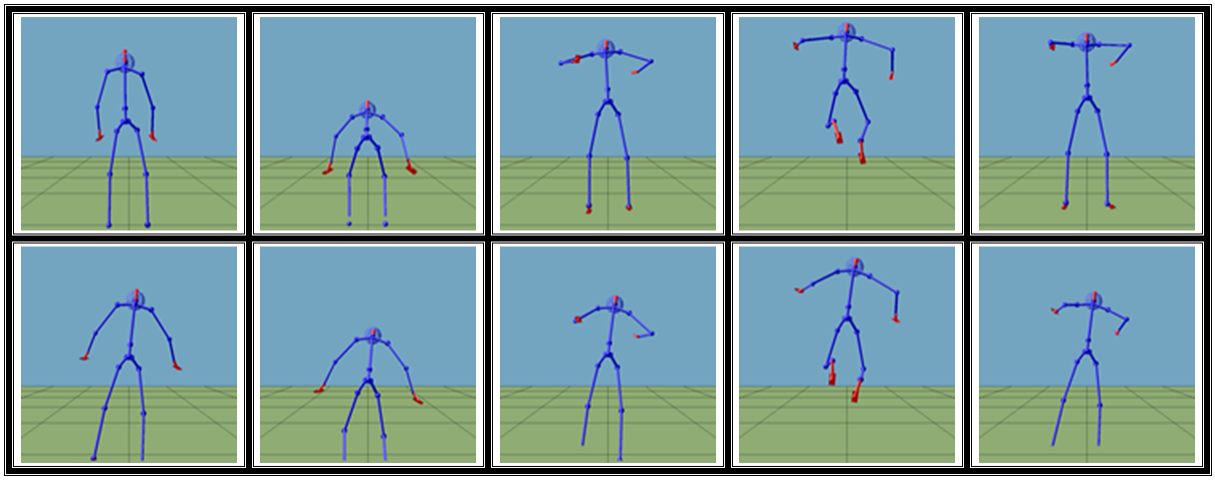


Figure 4.4: The features of the pressure sensing floor after channel-remapping from the generalized features

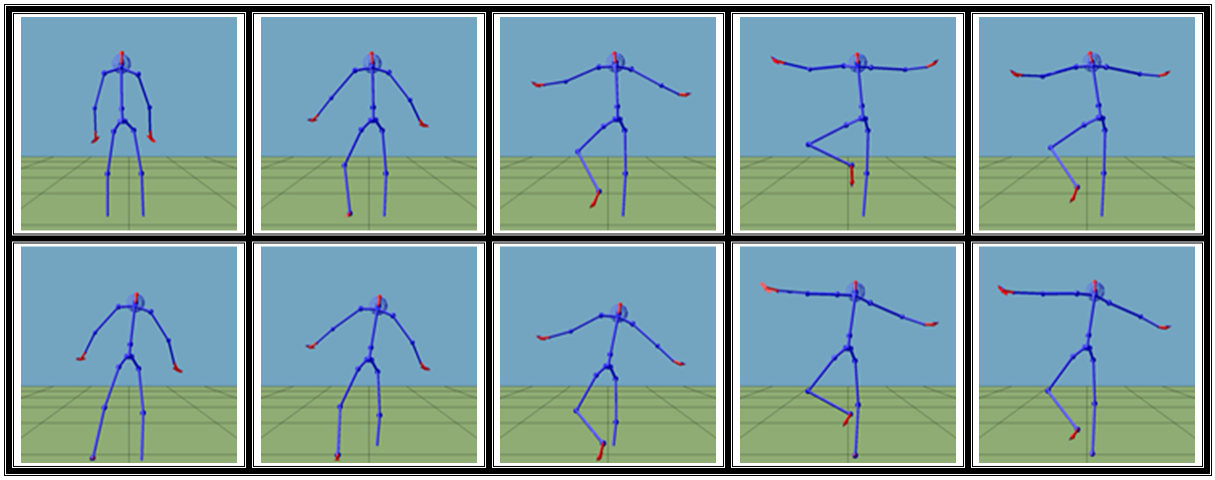
channel remapping sets. 65 generalized features were built from those 300 local features.

The generalized features were then used to regenerate local features. The sparsity optimization was solved and coefficients for the generalized features were computed. The original sensor data was then reconstructed using these channel-remapped features. Figure 4.5 shows a few frames from the original and reconstructed data for two human actions in the database.

Our method was able to reconstruct the sensory data with a 4.46% average reconstruction error. The method is also able to predict sensor data for the situations



(a) Original and reconstructed human motion data of jumping



(b) Original and reconstructed human motion data of balancing on one leg

Figure 4.5: The original data and the reconstructed human motion data for two actions in the database: (a) Balancing on one leg, (b) Jumping in place. The first row of each figure displays the original human motion data and the second row displays the reconstructed data using channel-remapped generalized feature

that were not part of the input data using remapping. For example, if throwing a ball with the right hand is in the training data then the model is able to find throwing the ball with the left hand - which does not exist in the training data - using remapping due to a relative symmetry of the human body which can be found through channel mappings arising in walking tasks.

For comparison purposes, we also executed the PCA algorithm on the human motion dataset. 65 principal components (features) were used here for describing the data, the same number of features used in our proposed method. Our proposed method has described its general features with 1936 elements while PCA has used 4290 nonzero elements to describe its own features. In fact, our method used more compressed features to represent the same data. Later, the calculated principal components from PCA algorithm were then used to reconstruct the human motion data. The data was reconstructed with a 2.92% average reconstruction error. In comparison, our proposed method represented the data in far more compressed features in a trade-off for a small performance reduction.

4.2 Modeling with Knowledge Transfer

4.2.1 Pressure Sensing Floor Simulator

In order to evaluate the performance of our method on modeling with knowledge transfer, a pressure sensing floor simulator is implemented. This simulator enables us to have a higher potential for transferring a pattern to equivalent localities. That is because the prototype of the pressure sensing floor is not large enough to have detectable equivalent channel mapping sets.The simulator replicates a 4×11 foot floor which has 30 sensors placed under it. Figure 4.6 displays what the sensor placement looks like.

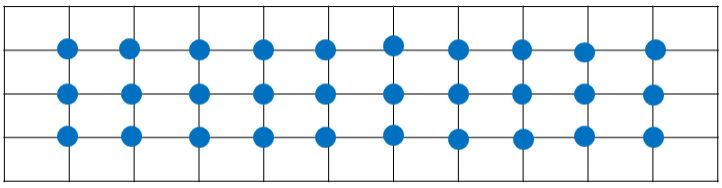


Figure 4.6: The placement of pressure sensors in the simulator

To capture data on this simulator, we ideally assumed that each point of pressure only affects the four sensors at the corners of the tile in which the pressure point is located. Assume p_p, x_p, and y_p are respectively the applied pressure to the pressure point, x-coordinate of the pressure point, and y-coordinate of it. Also assume that d_1,d_2,d_3,and d_4 are the pressure point's distance to the four sensor locations computed as follow:

$$d_i = \sqrt{(x_p - x_i)^2 + (y_p - y_i)^2}$$

Then, the pressure sensory reading r_i on sensors will be:

$$r_1 = \frac{(d_2 + d_3 + d_4)p_p}{d_1 + d_2 + d_3 + d_4}$$

$$r_2 = \frac{(d_1 + d_3 + d_4)p_p}{d_1 + d_2 + d_3 + d_4}$$

$$, ...$$

After that, we had two different source of pressure moving on the pressure sensing floor: First, a synthetic device like a 4-wheel vehicle, and second, an artificial human subject. These two sources give us two pressure profiles. The first profile and half of the second profile is used as training data to extract the features. Thus, the training data includes the pressure patterns of the artificial 4-wheel vehicle on the whole floor, but it does not have any pressure data (and therefore any pressure patterns) of the artificial human subject on half of the floor.

In this setting, our proposed method with knowledge transfer is expected to learn the equivalent localities or channel sets using the data provided by the first pressure profile. It is also expected to transfer the learned generalized features of the second pressure profile to new localities where is equivalent to the localities in which

it is observed. Therefore, a boost in performance is anticipated when our method uses the equivalent localities for knowledge transfer.

In the first step, feature learning component has extracted 16 local features from the given training data. Figure 4.7 shows these 16 local features.

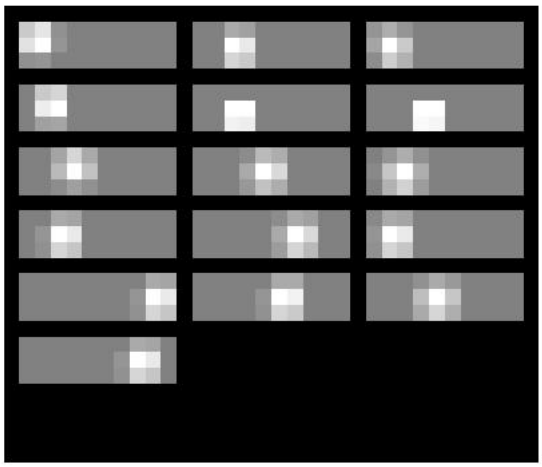


Figure 4.7: The 16 local features that is learned from training data

Then, after initial generalization, without knowledge transfer, 11 general features have been formed. The evaluation on testing data with these features resulted in 1.35% average reconstruction error. Finally, the generalization has been completed using equivalent channel mappings for knowledge transfer. The new generalized features were then applied to evaluate their performance on the testing set. The result was an average reconstruction error as low as 0.9%. This shows 33% decrease in average reconstruction error in comparison to the general features without knowledge transfer.

Like the other two experiments, our proposed method was compared with the PCA's performance on the data of this experiment. Again, the PCA algorithm has

used 11 principal components (features) here for describing the data, the same number of features used in our proposed method for this dataset. Our proposed method has described its generalized features with 214 elements when no transfer knowledge was used, and it has used 266 elements to describe its generalized features when it took the advantage of supersets for knowledge transfer purposes. On the other hand, the PCA algorithm has used 330 nonzero elements to describe its own features. Thus, our method applied more compressed features to represent the same data. After that, the calculated principal components from PCA algorithm were used to reconstruct the data. In fact, the data was reconstructed with a 8.79% average reconstruction error using principal components. As a result, one can see that our proposed method represented the data by more compressed features and had much better performance using the supersets to transfer its acquired knowledge.

As presented, by this method there is no need to have all the patterns on all possible localities. For instance, on the pressure sensing floor we can use data obtained in one context (i.e. cause by one source) to learn localities, then use that equivalent localities in addition to training data of a human subject walking on a limited number of places to form generalized features and model the way a person walks on the floor.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Learning features in complex data is an important task for modeling and learning tasks in real-world situations. The generalized feature and channel-remapping presented in this thesis provides a powerful method to construct a model for similar features over different sets of channels or localities. This method provides a succinct representational method to separate the stimuli from its locality and process it independent of where it happened. As a result, it provides the possibility to transfer stimuli to other channel sets where it has not been observed yet and thus to recognize previously unseen data patterns or to generate activities that have not previously occurred. This method learns the equivalent localities and moves the extracted pattern from one of these localities to the others. Therefore, the method enables us to model the extracted patterns on new localities for systematic data.

5.2 Future Work

Calculating the distances between two active-channel vectors by trying arbitrary mapping permutations of the active-channel vectors becomes computational expensive for large size active-channel vectors. Finding another way to calculate the minimum distances between two active-channel vectors would improve the performance of the proposed method.

In addition, applying the modeling method with knowledge transfer ability on real complex sensory data can ideally demonstrate the performance of the method

more clearly. If we have more patterns, more input channels, and complex data, modeling with knowledge transfer should theoretically show much better results in comparison with simple modeling methods.

Appendix A

Core Implementation Components

Generalizer without Knowledge Transfer

(SimpleGeneralizer.m)

```
clc;
clear all;
pars.beta = 10;
pars.noiseVar = 1;
pars.sigma = 1;
load('HFeature.mat');
load('HSensorReading.mat');
TB =B;
load('MFeature.mat');
load('MSensorReading.mat');
D = [MSensorRead; HSensorRead(1:534,:)];
B = [TB'; B']';
%% remove all zero pattern from the code
B =B(:,(sum(B) =0));
[dimension patternCount] = size(B);
contributionThreshold = 1e-3;
maxRelativeError = 0.01;
%% deactivate bases that are contributing less than threshold to the vector length
inactives = deactivate(B, contributionThreshold);
BP=abs(B);
BP(inactives==1)=0;
c = sum(BP =0,1)
mapping(patternCount,patternCount).ord1 = 0;
mapping(patternCount,patternCount).ord2 = 0;
distance = zeros(patternCount,patternCount);
[row col  ] = find(BP);
for i=1:patternCount
        for j=1:patternCount
                if(distance(j,i)==0)
                        %get the non-zero elements for column i
                        orderi = row(col==i);
                        ini = BP(orderi,i);
                        %get the non-zero elements for column j
                        orderj = row(col==j);
                        inj = BP(orderj,j);
                        % calculate the minimum distance between permutations
```

```
                % of two vectors
                [dist ord1 ord2] = MinDistance(ini, inj);
                distance(i,j)= dist;
                %save the permutation that gave us the minimum distance
                % if the first vector is shorter than the second
                nonExist = (ord1==-1);
                ord1(nonExist)=1;
                mapping(i,j).ord1 = orderi(ord1);
                % point to dummy dimension
                mapping(i,j).ord1(nonExist) = dimension+1;
                %save the permutation that gave us the minimum distance
                % if the second vector is shorter than the first
                nonExist = (ord2==-1);
                ord2(nonExist)=1;
                mapping(i,j).ord2 = orderj(ord2);
                %make the nonexisting element to point to a value more than
                %dimension which will be filled by 0
                % point to dummy dimension
                mapping(i,j).ord2(nonExist) = dimension+1;
        else
                %it is already computed (it is symmetric)
                distance(i,j)= distance(j,i);
                mapping(i,j).ord1 = mapping(j,i).ord2;
                mapping(i,j).ord2 = mapping(j,i).ord1;
        end
        fprintf('(% d , % d) \n ', i,j);
    end
end
%%
d = squareform(distance);
Z = linkage(d);
%show the dendrogram graph
dendrogram(Z);
save('temp2.mat');
%%
load('temp2.mat');
lastError = inf;
for i=size(Z,1):-1:1
    tempT = cluster(Z,'cutoff',Z(i,3),'criterion','distance');
    error = ReconstructionError( tempT, BP, distance, D, mapping, pars);
    if maxRelativeError >(lastError-error)/lastError && lastError >error
        distanceCutOff= Z(i,3);
        break;
    else
        lastError = error;
```

```
        end
    end
    T1 = cluster(Z,'cutoff',distanceCutOff,'criterion','distance')
    save('featureWithoutTransferKnowledge.mat');
```

Generalizer with Knowledge Transfer

(FullGeneralizer.m)

```
clear all;
load('featureWithoutTransferKnowledge.mat');
%% Look for identical channel mappings
active = (BP =0);
[dimension patternCount] = size(BP);
%% deactivate bases that are contributing less than threshold to the vector length
equal = zeros(patternCount);
for i=1:patternCount
        for j=1:patternCount
                if i =j
                        equal(i,j) = (sum(xor(BP(:,i),BP(:,j)))==0);
                end
        end
end
%% Bases from clustering
bases = BP;
clusterCount = numel(unique(T1));
clusterMeanIndex = zeros(clusterCount,1);
for i=1:clusterCount
        clusterMeanIndex(i,1) = ClusterMean((T1==i), distance);
end
bases(size(bases,1)+1, :) = 0 ;
one = ones(size(bases));
one(:,clusterMeanIndex) =0;
bases(one==1) = 0;
for i=1:size(bases,2)
        if clusterMeanIndex(T1(i,1))  = i
                clustMean = clusterMeanIndex(T1(i,1));
                bases(mapping(i,clustMean).ord1 , i) = bases(mapping(i,clustMean).ord2
,                 clustMean);
        end
```

```
end
%% Bases from knowledge transfer
[row col  ] = find(equal);
newBases = zeros(dimension+1,1);
for i=1:size(row)
        source = row(i,1);
        dest = col(i,1);
        destinations = find(T1 == T1(dest,1));
        for j=1:size(destinations)
                if destinations(j,1) =dest
                        temp = zeros(dimension+1,1);
                        temp(mapping( destinations(j,1), dest).ord1 , 1) = bases(
                        mapping( destinations(j,1), dest).ord2 , source);
                        newBases = [newBases, temp];
                end
        end
end
bases = [bases, newBases(:,2:size(newBases,2))];
bases = bases(1:size(bases,1)-1, :);
save('featureWithTransferKnowledge.mat');
```

Calculating the Reconstruction Error

(ReconsructionError.m)

```
function error = ReconstructionError( clusters, bases, distances, inputData,
dimMap, pars )
clusterCount = numel(unique(clusters));
clusterMeanIndex = zeros(clusterCount,1);
for i=1:clusterCount
        clusterMeanIndex(i,1) = ClusterMean((clusters==i), distances);
end
bases(size(bases,1)+1, :) = 0 ;%%adding dummy dimension
one = ones(size(bases));
one(:,clusterMeanIndex) =0;
bases(one==1) = 0;
for i=1:size(bases,2)
        if clusterMeanIndex(clusters(i,1))  = i
                clustMean = clusterMeanIndex(clusters(i,1));
                bases(dimMap(i,clustMean).ord1 , i) = bases(dimMap(i,clustMean).ord2
```

```
                , clustMean);
        end
end
bases = bases(1:size(bases,1)-1, :);
S= l1ls_featuresign(bases, inputData', pars.beta/pars.sigma*pars.noise_var);
DD = bases*S;
error = sum(sum( (DD-inputData').^2 ))/( size(DD,1)*size(DD,2) );
end
```

References

[1] Y. Ma, Y. Guo, X. Tian, and M. Ghanem, "Distributed clustering-based aggregation algorithm for spatial correlated sensor networks," *Sensors Journal, IEEE*, vol. 11, no. 3, pp. 641–648, 2011.

[2] I. Jolliffe, *Principal component analysis*, 2nd ed., ser. Springer Series in Statistics. New York City: Springer Publishing, August 2005.

[3] L. I. Smith, "A tutorial on principal components analysis," *Cornell University, USA*, vol. 51, p. 52, 2002.

[4] R. Jenatton, G. Obozinski, and F. Bach, "Structured sparse principal component analysis," *arXiv preprint arXiv:0909.1440*, 2009.

[5] M. Hein and T. Bhler, "An inverse power method for nonlinear eigenproblems with applications in 1-spectral clustering and sparse pca." *Advances in Neural Information Processing Systems (NIPS)*, pp. 847–855, 2010.

[6] H. Zou, T. Hastie, and R. Tibshirani, "Sparse principal component analysis," *Journal of Computational and Graphical Statistics*, vol. 15, no. 2, pp. 262–286, 2006.

[7] T. Hastie, R. Tibshirani, and J. Friedman, *"The Elements of Statistical Learning: Data Mining, Inference, and Prediction."*, 2nd ed. Springer-Verlag, February 2009.

[8] A. Hyvärinen and E. Oja, "Independent component analysis: algorithms and applications," *Neural networks*, vol. 13, no. 4, pp. 411–430, June 2000.

[9] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science Magazine*, vol. 313, no. 5786, pp. 504 – 507, 28th July 2006.

[10] H. Lee, A. Battle, R. Raina, and A. Ng, "Efficient sparse coding algorithms," *Advances in Neural Information Processing Systems (NIPS)*, vol. 19, pp. 801–808, 2007.

[11] S. J. Pan and Q. Yang, "A survey on transfer learning," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, no. 10, pp. 1345–1359, 2010.

[12] L. Torrey and J. Shavlik, "Transfer learning," *Handbook of Research on Machine Learning Applications. IGI Global*, vol. 3, pp. 17–35, 2009.

[13] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1?" *Vision research*, vol. 37, no. 23, pp. 3311–3325, 1997.

[14] F. Murtagh, "A survey of recent advances in hierarchical clustering algorithms," *The Computer Journal*, vol. 26, no. 4, pp. 354–359, 1983.

[15] "Human motion database," http://smile.uta.edu/hmd/, 2011, [Online; accessed 25-April-2013].

[16] G. Guerra-Filho and A. Biswas, "The human motion database: A cognitive and parametric sampling of human motion," *Image Vision Comput.*, vol. 30, no. 3, pp. 251–261, 2012.

Biographical Information

Houtan Rahmanian received his Bachelors degree from Baha'i Institute for Higher Education (BIHE), in 2008. He started his Masters in Computer Science at The University of Texas at Arlington in 2011. His current research interests are in the area of machine learning, data mining, and computer vision. He has joined Amazon.com, Inc. after graduation.