

Robust Dictionary Learning for Image Processing

by

Samarth Mishra

Roll No: 130050008

under the guidance

of

Prof. Suyash Awate

Bachelors' of Technology Thesis (II)



Department of Computer Science and Engineering

Indian Institute of Technology, Bombay

Declaration

I declare that this written submission represents my ideas in my own words and where other's ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Date: 23th November, 2016

Samarth Mishra

Place: IIT Bombay, Mumbai

Roll No: 130260018

Acknowledgements

I would first like to thank Prof. Suyash Awate, my thesis advisor, for his continuous support and guidance, through the course of this project. I would thank Ravi, my teammate for this project, who has been an excellent colleague and a friend. I would also like to thank Pradyot and Aditya, for their help during the intial phase of the project, and Nishanth, for helping us get a hang of his previous implementation.

Contents

1	Introduction	1
2	Background	3
2.1	The Dictionary Learning Problem	3
2.2	Non negative sparse coding (NNSC)	5
2.3	Manifold Geometry	6
2.3.1	Manifolds and Riemannian Manifolds	6
2.3.2	Distances on Riemannian Manifolds	7
3	Dictionary Learning on Hyperspheres	8
3.1	Nonlinear Sparse Coding	8
3.2	Manifold Dictionary Learning for images	10
3.2.1	Dictionary learning on the unit hypersphere	10
3.2.2	Denoising algorithm	11
3.2.3	Experiments	12
4	Structured sparsity and semi-supervision	16
4.1	Some background on norms	16
4.2	The optimal dictionary size	18
4.3	Using supervision data	18
4.4	An outline of the algorithm	19
4.5	Understanding Structured Sparsity	21

4.6	Experiments-1	21
4.6.1	Structured sparsity	21
4.6.2	Semi-supervised learning	23
4.7	Experiments on data on unit sphere	24
4.7.1	Structured sparsity	24
4.7.2	Semi-supervised learning	24
4.8	Semi-supervised dictionary learning on MNIST dataset	26
5	Kernel Dictionary Learning	30
5.1	Formulation	30
5.2	The kernel trick	31
5.3	Formulating the cost function using kernels	32
6	Graph Regularized Sparse Codes	34
6.1	Formulating graph regularized sparse coding	34
6.2	Modifications to graphSC implemented	36
7	Experiments with graph regularization	37
7.1	Performance of gaussianNN regularization	37
7.1.1	Gaussian noise	37
7.1.2	Salt and Pepper noise	42
7.2	Performance of kNN regularization	46
7.3	Performance with varying p	50
7.4	Performance with different kernels	53
8	Discussion and Future Work	55

Chapter 1

Introduction

Dictionary Learning is a technique used to learn discriminative sparse representations of complex data. The essence of this technique is similar to principal components. The aim is to learn a set of basis elements, such that a linear combination of a small number of these elements can be used to represent all given data points. A set of these basis elements is called a dictionary. A single element of a dictionary is referred to as an atom. This linear decomposition of data into atoms has found its use in many state of the art image processing applications.

One such application is the use in denoising images. Dictionary atoms can learn to represent only the important parts of an image, ignoring the noise, and thus reconstruction of the image from these atoms results in a denoised image.

Dictionary learning has also been very successful in classification tasks. Learning classifiers on weight vectors of data points (i.e., vector containing weights of different atoms required to reconstruct the data point), rather than on the original data helps in better discrimination, and hence, better classification.

Variants of dictionary learning techniques which exploit the underlying substructure in the data, have been found to boost this performance in classification tasks. State of the art results have been found for classification on image datasets like textured images and medical images [6].

Also, some other variants, which use the idea of differential learning for different classes, have been shown to perform well in classification of facial images, handwritten digits, etc [5]. These base classification on the idea that data points in different classes will use different atoms in their sparse representations and that classes can be identified by the set of atoms used in the linear combination.

Incorporating new ideas into traditional dictionary learning has been yielding some good results in terms of performance boosts over the state of the art, and this hence, has become a hot topic for research.

Chapter 2

Background

2.1 The Dictionary Learning Problem

Consider a set of d dimensional data points

$$\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2 \dots \mathbf{X}_n]$$

. Thus, \mathbf{X} is a $d \times n$ dimensional matrix where data points lie along columns of the matrix.

The goal of the dictionary learning task is to learn a set of m atoms $\mathbf{A} = [\mathbf{A}_1, \mathbf{A}_2 \dots \mathbf{A}_m]$, and corresponding weights $\mathbf{W} = [\mathbf{W}_1, \mathbf{W}_2 \dots \mathbf{W}_n]$ such that $\mathbf{X} \approx \mathbf{A}\mathbf{W}$. The i^{th} column of the weight matrix corresponds to the weights of the atoms in the linear combination to reconstruct the data point \mathbf{X}_i .

$$\mathbf{X}_i \approx w_{1i}\mathbf{A}_1 + w_{2i}\mathbf{A}_2 + \dots + w_{mi}\mathbf{A}_m$$

where

$$\mathbf{W}_i = [w_{1i}, w_{2i} \dots w_{mi}]^T$$

We would like to learn a dictionary that minimises the difference between the actual and the reconstructed data, but under certain constraints of sparsity on the weights. Hence the problem is to learn \mathbf{A} and \mathbf{W} such that

$$\mathbf{A}, \mathbf{W} = \arg \min_{\mathbf{A}, \mathbf{W}} \sum_{i=1}^n \|\mathbf{X}_i - \mathbf{A} \mathbf{W}_i\|_2^2 + \lambda f(\mathbf{W})$$

where $f(\mathbf{W})$ is the sparsity term, f being a function that has a smaller value for more and more sparse representations.

Different techniques can and have been used to minimise the objective function, the simplest being gradient descent. Gradient descent algorithms that have been used in practice for the above optimisation function usually update atoms and weights alternately, to convergence [3] [6].

In the above objective function, the difference between the actual and the reconstructed data has been measured in terms of the *Frobenius norm* of the matrix $\mathbf{X} - \mathbf{A} \mathbf{W}$. $f(\mathbf{W})$ is a sparsity function on the matrix \mathbf{W} . This term is also known as the regulariser. Different kinds of norms can be used for the difference term and the sparsity term, and these lead to different results. The l_0 norm, which counts the number of non-zero terms, is ideal for the sparsity term. But this leads to non-differentiability of the objective function, and hence, turns the problem into an NP-hard combinatorial optimization problem, which is undesirable. Hence common regularisers use l_1 or l_2 norm (which is the same as Frobenius norm). A more detailed analysis of sparsity functions and different kinds of norms can be found in Section 4.1.

Although the number of atoms to use, m in the above algorithm, is a hyperparameter, there has been some recent work [5], which uses a kind of sparsity term to induce what is called *structured sparsity*. Training using this sparsity term can lead to elimination of atoms from an overfull dictionary, i.e., the weights corresponding to these atoms become negligible after training, and it is almost as if they aren't used in the linear combination to reconstruct data points. Thus, this algorithm can "learn" the parameter m , if initialised with a large enough dictionary.

2.2 Non negative sparse coding (NNSC)

Non negative sparse coding introduces certain additional constraints to the dictionary learning problem. The constraint added is that coefficients in the weight matrix \mathbf{W} should all be non-negative. It is argued that negative coefficients lead to features cancelling each other out, and hence, building up data points from different features(atoms) should use positive weights for these features.

In [3], the author has used the following cost function for dictionary learning.

$$C(\mathbf{A}, \mathbf{W}) = \frac{1}{2} \|\mathbf{X} - \mathbf{AW}\|_2^2 + \lambda \sum_{ij} f(W_{ij})$$

For sparsity, the function used is $f(s) = |s|$, which makes the objective function

$$C(\mathbf{A}, \mathbf{W}) = \frac{1}{2} \|\mathbf{X} - \mathbf{AW}\|_2^2 + \lambda \sum_{ij} W_{ij}$$

since, the constraint of non-negative weights is imposed.

A gradient descent algorithm, as discussed in the previous section, can be used to minimise the above objective function (starting with proper initialisation). Though, in [3], the author has used a more efficient direct update for the weight matrix. Using the l_1 norm was primarily motivated by the fact that this makes the objective function quadratic in \mathbf{S} , allowing for the single step update:

$$\mathbf{W}^{t+1} = \mathbf{W}^t .* (\mathbf{A}^T \mathbf{X}) ./ (\mathbf{A}^T \mathbf{AW}^t + \lambda)$$

where, \mathbf{W}^t is the matrix value at iteration step t , and \mathbf{A}^T means the transpose of matrix \mathbf{A} . The operators $.*$ and $./$ signify element wise operations on the two operand matrices. Algorithm 1 outlines the algorithm used in [3] for learning the dictionary and weights.

Algorithm 1 NNSC

- 1: $t \leftarrow 0$
 - 2: Initialize \mathbf{A}^0 and \mathbf{W}^0 to strictly positive values
 - 3: **repeat**
 - 4: $\mathbf{A}' \leftarrow \mathbf{A}^t - \mu(\mathbf{A}^t \mathbf{W}^t - X)(\mathbf{W}^t)^T$
 - 5: Set all negative values in \mathbf{A}' to 0 and rescale each column to unit norm
 - 6: $\mathbf{A}^{t+1} \leftarrow \mathbf{A}'$
 - 7: $\mathbf{W}^{t+1} \leftarrow \mathbf{W}^t \cdot * ((\mathbf{A}^{t+1})^T \mathbf{X}) ./ ((\mathbf{A}^{t+1})^T (\mathbf{A}^{t+1}) \mathbf{W}^t + \lambda)$
 - 8: $t \leftarrow t + 1$
 - 9: **until** convergence
-

2.3 Manifold Geometry

The conventional dictionary learning task uses the vector space structure and metrics of the Euclidean space \mathbb{R}^d . Though, for many applications in directional statistics, machine learning, computer vision and medical image analysis data [6] on which dictionaries are to be learned tend to lie on known Riemannian manifolds. Using structure and metrics of the underlying manifold in the dictionary learning task can lead to better feature representation.

2.3.1 Manifolds and Riemannian Manifolds

A manifold \mathcal{M} of dimension d is a topological space that is locally homeomorphic to open subsets of the the Euclidean space \mathbb{R}^d , i.e., locally resembles Euclidean space near each point. A manifold which is locally similar enough to a linear space, to allow for differential calculus, is called a differentiable manifold. The tangent space at some $x \in \mathcal{M}$, denoted by $T_x \mathcal{M}$, is the space containing all tangent vectors to \mathcal{M} at x . \mathcal{M} becomes a Riemannian manifold if an inner product is defined on the tangent space.

2.3.2 Distances on Riemannian Manifolds

A geodesic $\gamma : [0, 1] \rightarrow \mathcal{M}$ is a smooth curve between two points $x_i, x_j \in \mathcal{M}$, such that $\gamma(0) = x_i$ and $\gamma(1) = x_j$. The geodesic distance between x_i and x_j is defined as the length of the shortest geodesic joining them. At each point $x \in \mathcal{M}$, there is defined a unique geodesic $\gamma_v : [0, 1] \rightarrow \mathcal{M}$ with initial tangent vector $v \in T_x\mathcal{M}$.

A point in the tangent space at x is mapped to a point on the manifold by the Riemannian exponential map. A Riemannian exponential map based at $x \in \mathcal{M}$ is a function of initial tangent vector v and is defined as $\exp_x(v) = \gamma_v(1)$. Thus, the mapping of a point on the tangent plane at x which is a unit distance from x along v is a point on the manifold which lies a unit distance away on the geodesic γ_v . The Riemannian log map is defined as the inverse of the exp map. Under the assumption that \log_x is defined over the entire \mathcal{M} , the following two statements hold true:

- The geodesic distance between x_i and x_j is given by $\mathbf{dist}_{\mathcal{M}}(x_i, x_j) = \|\log_{x_i}(x_j)\|_{x_i}$, where $\|v\|$ is the length of the vector v in the tangent space $T_x\mathcal{M}$
- $\mathbf{dist}_{\mathcal{M}}^2(x, -)$ is a smooth function for all $x \in \mathcal{M}$

Chapter 3

Dictionary Learning on Hyperspheres

Spheres (or hyperspheres, for a higher dimensional space) are the most well known class of closed Riemannian manifolds. Data from several machine learning and computer vision applications, are presented as points on hyperspheres. Dictionary learning for such data should use metrics of the underlying sphere for better learning. Distances on the sphere are measured along the shortest geodesic, which is the greater circle joining two points. In [6], the authors suggest a modification of the traditional dictionary learning algorithm to allow it to use Riemannian metrics on such known manifolds.

3.1 Nonlinear Sparse Coding

The authors in [6] argue that the notion of sparsity in the linear Euclidean space depends on the definition of the origin, and hence it isn't coordinate invariant. But on manifolds, sparsity needs to be coordinate invariant, because there is no notion of an origin on a manifold. Hence, the authors introduce a new notion of sparsity with affine constraint. They define a vector to be an affine sparse vector, if it can be written as an affine linear combination of a small number of basis vectors. Mathematically, this can be written as:

$$x = w_1 a_1 + w_2 a_2 + \dots + w_n a_n$$

$$w_1 + w_2 + \dots + w_n = 1 \quad (\text{which is the affine constraint})$$

In [6], the authors have shown that the following cost function (which needs to be minimised over \mathbf{A} and \mathbf{W}) can be used to learn dictionaries on data lying Riemannian manifolds.

$$\sum_{i=1}^n \left\| \sum_{j=1}^m W_{ji} \log_{\mathbf{x}_i}(\mathbf{A}_j) \right\|_{\mathbf{x}_i}^2 + \lambda \|\mathbf{W}\|_1 \quad (3.1)$$

$$\mathbf{s.t.} \sum_{j=1}^m W_{ji} = 1, \text{ for } i = 1, 2 \dots n$$

The argument for correctness (which is a heuristic one) involves translating the distance metric between the actual data point and the approximation using dictionary atoms, into an equivalent form which uses the geodesic distance between the data point and each of the individual atoms. The details of the argument have been skipped here, for the sake of brevity.

3.2 Manifold Dictionary Learning for images

Patch based image denoising is a task that dictionary learning can be used for. Dictionary atoms learn essential parts of the image, ignoring the noise, and reconstruction of the image from those atoms gives us a denoised image. On visualising these dictionary atoms, we can see that these have learned features in the images, like, edges of different orientations, curves, textures, etc. Thus, the dictionary atoms can be seen to serve as building blocks for patches of the image. Through our experiments we can also see that the learned dictionary generalises well, i.e., it fairs well enough in the reconstruction of images other than the one it was learned from.

Important features of an image reside in its high variance patches and hence we use top-n high variance patches for learning our dictionary. While reconstructing the image, we add a constant intensity patch to the dictionary of learned atoms, which would be used for the constant intensity component in the reconstructed patch.

3.2.1 Dictionary learning on the unit hypersphere

For dictionary learning on the unit hypersphere. using the appropriate log map in the equation 3.1, we get the following objective function.

$$\sum_{i=1}^n \left\| \sum_{j=1}^m W_{ji} \cos^{-1}(\langle \mathbf{X}_i, \mathbf{A}_j \rangle) \frac{\mathbf{u}_{ij}}{\|\mathbf{u}_{ij}\|} \right\|_{x_i}^2 + \lambda \|\mathbf{W}\|_1 \quad (3.2)$$

$$\sum_{j=1}^m W_{ij} = 1, \text{ for } i = 1, 2, \dots, n$$

where $\mathbf{u}_{ij} = \mathbf{A}_j - \langle \mathbf{X}_i, \mathbf{A}_j \rangle \mathbf{X}_i$ and $\langle \cdot, \cdot \rangle$ is the vector dot product.

Since there is no direct single step update optimisation technique here as in the case of NNSC, we use gradient descent to optimise the above cost function. A gradient step may push the \mathbf{A} and \mathbf{W} values out of their confined sets, i.e., the unit hypersphere for the atoms and the affine constraint for the weights. So after each gradient step, we have to project the matrices \mathbf{A} and \mathbf{W} appropriately.

To improve the initialisation for our algorithm, we use k-means to initialise our atoms, the number of means being equal to the number of atoms. Furthermore, we use farthest point heuristics [2] to initialise the cluster centers for k-means clustering. Also, we adaptively tune our gradient step size, for faster training.

3.2.2 Denoising algorithm

For clarity, we present our algorithm in a step wise format.

Algorithm 2 Image reconstruction for denoising

- 1: Find top n high variance patches in the image
 - 2: Vectorize the patches to form the matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$
 - 3: Normalise each of the individual columns of the matrix and store the normalisation factors ($\|\mathbf{X}_i\|_2$ for column i) in \mathbf{N}
 - 4: Initialize $\mathbf{A} \in \mathbb{R}^{d \times m}$ using K-means
 - 5: Normalise the columns of \mathbf{A} to unit norm.
 - 6: Initialize $\mathbf{W} \in \mathbb{R}^{m \times n}$ using K-means and project it onto the plane of the following constraint: $\sum_{i=1}^m w_{ij} = 1 \forall j = 1, 2, \dots, n$
 - 7: **repeat**
 - 8: Perform gradient descent on \mathbf{A} till convergence, keeping \mathbf{W} fixed
 - 9: Perform gradient descent on \mathbf{W} till convergence, keeping \mathbf{A} fixed
 - 10: **until** both \mathbf{A} and \mathbf{W} converge
 - 11: $\mathbf{A}^{(1)} \leftarrow [\mathbf{A}, \mathbf{P}]$ where $\mathbf{P} \in \mathbb{R}^d$ is the constant intensity patch
 - 12: Construct $\mathbf{X}^{(1)}$, the matrix containing all patches of the image, vectorised, along its columns
 - 13: Fit $\mathbf{A}^{(1)}$ to $\mathbf{X}^{(1)}$ to get the new coefficient matrix $\mathbf{W}^{(1)}$
 - 14: Construct $\mathbf{X}^{(2)}$ as $(\mathbf{A}^{(1)}\mathbf{W}^{(1)})$ and rescaling each using the appropriate coefficient in \mathbf{N} .
 - 15: Reshape columns of $\mathbf{X}^{(2)}$ into patches and find average intensity at each pixel
-

3.2.3 Experiments

For the denoising experiments, we use the standard Lena 256x256 image. We run denoising using the dictionary learning routine thrice, on the Lena image. For the first run, we use the original image. For the second, we use the image which has been corrupted with 5% zero-mean gaussian noise, and for the third, we use a noisy image with 10% zero-mean gaussian noise.

For each of these experiments, we chose $\sim 10k$ high variance 9x9 (side measured in number of pixels) patches and learn a dictionary of 50 atoms on these. Before using the patches for learning, we normalise each patch to unit norm. This makes our data lie on a unit hypersphere.

We then reconstruct the images from these atoms by learning the weights for all 9x9 patches using these 50 dictionary atoms. For calculating the intensity at a pixel, we average the intensities from all the reconstructed patches which that pixel belongs to. The 50 atom dictionary learned from patches from the Lena image can be seen in Figure 3.1.


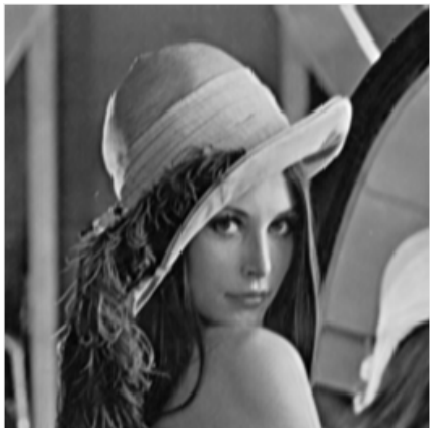




	Original Image	Reconstructed image
No noise (~10k high-variance patches and 50 atoms used)		
5% zero-mean gaussian noise (~10k high-variance patches and 50 atoms used)		
10% zero-mean gaussian noise (~11k high-variance patches and 50 atoms used)		

Table 3.1: Denoising results on the lena image

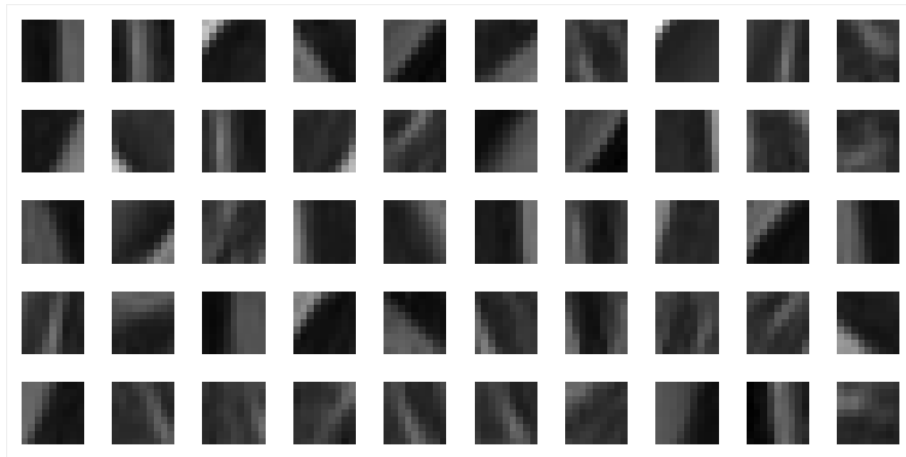


Figure 3.1: Dictionary of 50 atoms (9x9 pixels each) learned from $\sim 10k$ high variance patches of the (uncorrupted) lena image

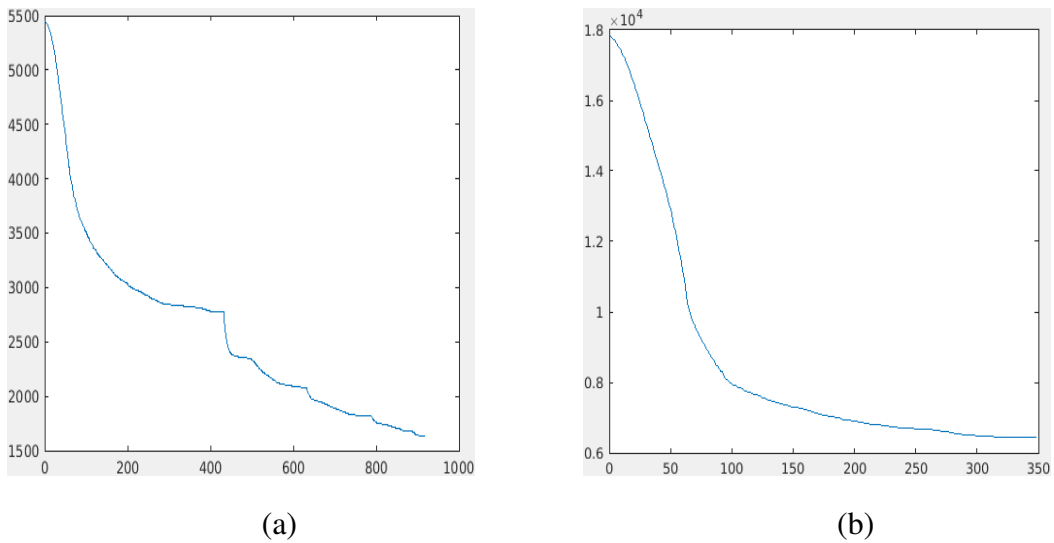


Figure 3.2: Objective functions : (a) learning atoms and corresponding weights from $\sim 10k$ patches from the Lena image (b) learning weights for all the patches using the learned dictionary of 50 atoms



(a)

(b)

(c)

Figure 3.3: (a) Original boat image (b) Boat image reconstructed from atoms learned from the image itself (c) Boat image reconstructed from atoms learned from the Lena image

To test how well this learned dictionary of atoms from the Lena image generalises, we run another experiment using another standard image : the fishing boat. On the Boat image, we run our previous reconstruction routine (after learning 50 atoms from the same image using high variance patches). We compare this reconstructed result with the image reconstructed using the atoms learned from the Lena image. As can be seen in Figure 3.3, the dictionary learned from the Lena image does a good enough job in reconstructing the Boat image.

Chapter 4

Structured sparsity and semi-supervision

In [5], the authors point out that traditional dictionary learning methods have three major weaknesses : sensitivity to outliers and noisy data, difficulty to determine the optimal dictionary size, and incapability to incorporate supervision information, say, in a classification task. To overcome these, the authors have suggested changes to the the objective function for the dictionary learning task. Here, we focus on removing the latter two weaknesses.

4.1 Some background on norms

Given $p > 0$, the l_p - norm of the vector $\mathbf{v} \in \mathbb{R}^n$ is defined as $\|\mathbf{v}\|_p = (\sum_{i=1}^n |\mathbf{v}_i|^p)^{\frac{1}{p}}$. The l_0 -norm, defined as $\|\mathbf{v}\|_0 = \sum_{i=1}^n |\mathbf{v}_i|^0$ counts the number of non-zero elements in \mathbf{v} .

For matrices, we can define mixed norms, which compute a certain l_r -norm on each of the individual rows, and then compute the l_p -norm over these l_r -norm values. The $l_{p,r}$ norm of a matrix \mathbf{M} can be defined as $\|\mathbf{M}\|_{r,p} = \left(\sum_i \left(\sum_j |m_{ij}|^r \right)^{\frac{p}{r}} \right)^{\frac{1}{p}} = \left(\sum_i \|\mathbf{m}^i\|_r^p \right)^{\frac{1}{p}}$. Note that \mathbf{m}^i is the i^{th} row of the matrix \mathbf{M} . The sparsity term that we used for dictionary learning on the sphere was the $l_{2,2}$ norm of the matrix \mathbf{W} .

The l_0 norm or the $l_{2,0}$ of the weight matrix would perhaps be the best sparsity prior

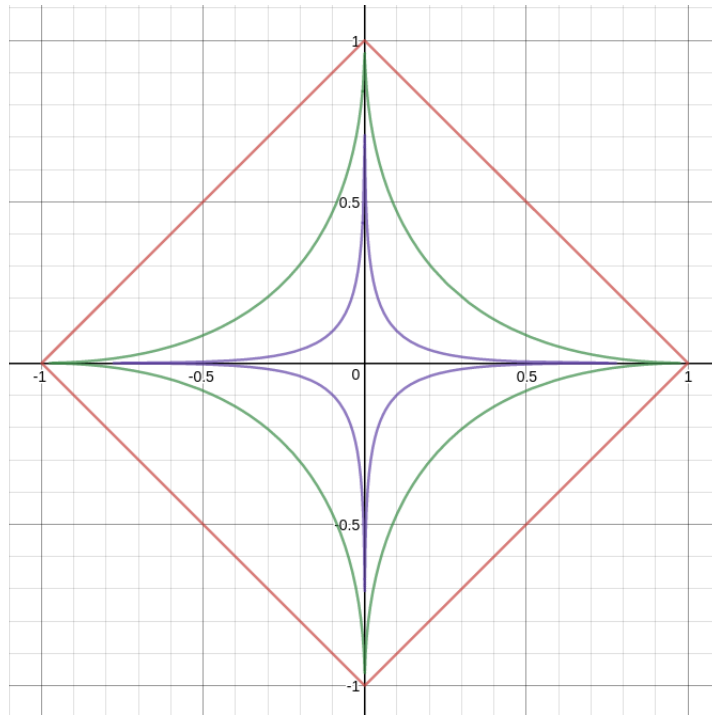


Figure 4.1: The graph for $|x|^p + |y|^p = 1$. Red : $p = 1$; Green: $p = 0.5$; Purple: $p = 0.3$

to use for a dictionary learning task. It follows the most intuitive definition of sparsity. However, the l_0 norm is not a valid vector norm, and is not a continuous function of the vector \mathbf{v} . If we use the l_0 norm in our objective function, we won't be able to learn the dictionary using standard optimisation techniques. The problem of dictionary learning becomes a combinatorial optimisation problem, which is NP-hard. Hence, we resort to other sparsity priors which can be used in the objective function and are smooth enough to be optimised using methods like gradient descent.

An effective way to approximate an l_0 norm is to use an l_p norm, where $p < 1$. As $p \rightarrow 0$, the function becomes sharper and in the limit, resembles the l_0 norm of (x, y) . This can be seen in figure 4.1.

4.2 The optimal dictionary size

In standard dictionary learning settings, the dictionary size is unknown beforehand and is treated as a hyper-parameter. The dictionary is often designed to be over-complete and ends up with many redundant atoms (which might be quite similar to another atom, or may be representable as a sparse linear combination of other atoms). To tackle this issue, the authors in [5] propose to use the following cost function for optimisation.

$$J(\mathbf{A}, \mathbf{W}) = \|(\mathbf{X} - \mathbf{A}\mathbf{W})^T\|_{2,q}^q + \lambda \|\mathbf{W}\|_{2,p}^p$$

where $p, q \in (0, 1)$. The $\|\mathbf{W}\|_{2,p}^p$ norm function is said to impose structured sparsity on the weights, i.e., it penalises all n coefficients of a single atom together, and imposes an l_p -norm over these. Hence, if $p < 2$, such a sparsity term tends to either allow for relatively dense coefficients in the row of weights corresponding an atom or have the entire row to be very sparse, almost eliminating any contribution by the atom. Hence, after optimisation on the above objective, the optimal dictionary size is the number of atoms which have non-zero weights remaining. To put it more precisely,

$$A_X = \{\mathbf{a}_i \mid \|\mathbf{w}^i\|_2 > 0\}$$

where A_X is the effective dictionary, i.e., the set of dictionary atoms that are used for representation of data. The size of this dictionary, $|A_X|$ is learned from the data \mathbf{X} rather than being specified as a hyperparameter. In further sections, we represent a matrix of our dictionary atoms as \mathbf{A}_X , which contains the elements of A_X as its columns. In [5], \mathbf{A} has been termed *super-dictionary* because \mathbf{A}_X is a subset of \mathbf{A} . The size of \mathbf{A} has to be specified beforehand. However, it is usually enough to have the size of \mathbf{A} larger than the final size of \mathbf{A}_X which has been obtained from a large \mathbf{A} .

4.3 Using supervision data

A dictionary that can use label information from the input data set can learn more discriminative features and lead to improved performance of subsequent classification tasks. In [5],

the structured sparsity term on the weight matrix \mathbf{W} has been modified to incorporate label information.

Suppose we need to train our dictionary for a classification task with K classes. We have two kinds of input data: labeled and unlabeled. Labeled data $\{\mathbf{x}^i\}$, also have the associated labels $\mathbf{y}_i \in \{0, 1\}^K$, such that $y_{ik} = 1$ if \mathbf{x}^i belongs to the k^{th} class, and 0 otherwise. Let us denote $\mathbf{X}_i \in \mathbb{R}^{d \times n_i}$, as the set of all labeled data points which belong to class i , where $i = 1, 2, \dots, K$ and n_i is the number of data points in the i^{th} class. Let \mathbf{X}_0 be the matrix of all unlabeled data. Also, let $\tilde{\mathbf{X}} = [\mathbf{X}_0, \mathbf{X}_1 \dots \mathbf{X}_K]$. Similarly, the new weight matrix $\tilde{\mathbf{W}} = [\mathbf{W}_0, \mathbf{W}_1, \dots, \mathbf{W}_K]$, where \mathbf{W}_0 is the coefficient matrix corresponding to the unlabeled data, and \mathbf{W}_k , corresponding to the k^{th} class. The new cost function under this scenario is,

$$J(\mathbf{A}, \tilde{\mathbf{W}}) = \left\| \left(\tilde{\mathbf{X}} - \mathbf{A} \tilde{\mathbf{W}} \right) \right\|_{2,q}^q + \lambda \sum_{k=0}^K \|\mathbf{W}_k\|_{2,p}^p$$

After optimisation, we construct the k^{th} class specific dictionary as \mathbf{A}_k which has columns from the set $\{\mathbf{a}_i \mid \|\mathbf{w}_k^i\|_2 > 0\}$. Each class uses different dictionary atoms for reconstruction of its data points. This has important significance in the classification task. In [5], the authors use a classification technique which, for any data point, tries to reconstruct the given point by learning the coefficients using the set of atoms \mathbf{A}_k for each class k . Whichever class dictionary gives the least reconstruction error, is the class assigned to the data point by the classifier. Since the dictionary has been learned using both labeled and unlabeled data, the term *Semi-supervised Dictionary* has been used in [5] for the dictionary learned by optimising the above objective function.

4.4 An outline of the algorithm

In [5], the authors have used a new efficient algorithm to minimise the objective function. This has been done because the objective function used for training (the one from the previous section) is non-smooth because of the $l_{2,p}$ -norm terms involved. We however, use a different approach and try to make the function smooth by using the ϵ -regularised l_p quasi-

norm instead of the non-regularised version in the sparsity term. For our experiments, we use $q = 2$, and hence, the first term of the objective function, does not need to be regularised to make it smooth.

Let,

$$\bar{\mathbf{w}}_k = \begin{bmatrix} \|\mathbf{W}_k^1\|_2 \\ \|\mathbf{W}_k^2\|_2 \\ \vdots \\ \|\mathbf{W}_k^m\|_2 \end{bmatrix}_{m \times 1} \quad (4.1)$$

for each of the k classes. Our sparsity term includes the ϵ -regularised p -norm of $\bar{\mathbf{w}}_k$ instead of their regular l_p norms. Here, ϵ is a small positive constant and is used to make the norm a smooth function of the vector. The p^{th} power of the ϵ -regularised p -norm of a vector $\mathbf{v} \in \mathbb{R}^{m \times 1}$ is given by

$$\|\mathbf{v}\|_{2,p,\epsilon}^p = \sum_{i=1}^m (|v_i|^2 + \epsilon)^{\frac{p}{2}}$$

where v_i is the i^{th} term of the vector \mathbf{v} . Thus, the objective function becomes

$$J(\mathbf{A}, \tilde{\mathbf{W}}) = \left\| \left(\tilde{\mathbf{X}} - \mathbf{A} \tilde{\mathbf{W}} \right) \right\|_{2,q}^q + \lambda \sum_{k=0}^K \|\bar{\mathbf{w}}_k\|_{2,p,\epsilon}^p$$

Now that the objective function is smooth enough, we can use gradient descent for optimisation. In our experiments, we have additional constraints on \mathbf{A} and \mathbf{W} which are: $\|\mathbf{A}^i\|_2 = 1 \forall i = 1, 2, \dots, m$ and $\sum_{i=1}^m w_{ij} = 1 \forall j = 1, 2, \dots, n$, $w_{ij} > 0 \forall i \forall j$. This means we are only considering affine linear combinations of atoms, with all the coefficients being positive. Due to these added constraints, we use projected gradient descent for optimisation. We first ran experiments to see if the idea of imposing structured sparsity, and semi-supervision do work, and used simple 2-D data on a plane and then, tried the experiment on 3-D data points on a unit sphere. The description of the experiments are given in the following sections. We had also found empirically, that $p = 0.5$, is the optimal value for to use while optimisation of the above cost function using gradient descent. Using smaller p

values along with regularisation leads to convergence to points which might not be optimal and hence, a value of 0.5 for p has been used in all the following experiments.

4.5 Understanding Structured Sparsity

To understand if structured sparsity works (i.e., is the sparsity term lower for less number of atoms used, and can this configuration be attained by minimising the sparsity term using gradient descent) we conduct a very simple experiment. Let us consider data on a line segment joining $(0, 0.5)$ and $(0, 1)$ on a 2-D plane. For further description, refer to figure 4.2. The data can be represented as a combination of three atoms represented by 'x's in the figure, where the weights follow affine constraints and are positive. The data can also be represented by using just 2 atoms, which are marked by the blue 'x's in the figure. The sparsity term for the latter representation is smaller than the former, hence, we expect our algorithm to end up with 2 atoms(the blue 'x's) when we begin training with 3 and this is what we observe. The weights corresponding to the 3rd atom (the red 'x') become zero for all data points on training our dictionary weights with the structured sparsity cost function. Note that for the above experiment we used all unlabelled data so that the cost function translates to the one in section 4.2.

4.6 Experiments-1

4.6.1 Structured sparsity

We chose 1000 data points on the 2-D plane, lying on either of the two line segments, joining $(0, 0)$ and $(1, 0)$ or the one joining $(0, 0)$ and $(0, 1)$. We use the cost function as mentioned in section 4.4. Our aim here, again, is to see if structured sparsity leads to the learning of a sparse dictionary and is it able to prune the dictionary size. The cost function translates to the one in section 4.2 if there is no labelled data. We thus give no labels to the points for our experiments in this section.

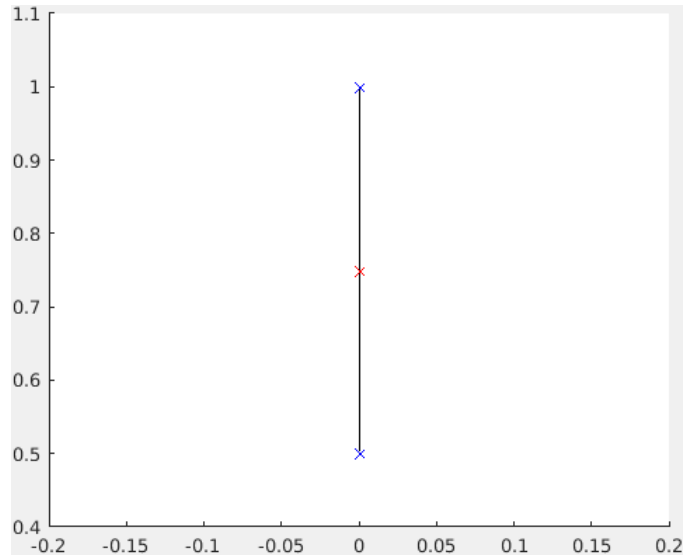


Figure 4.2: Simple example to illustrate working of structured sparsity

We initialise our super-dictionary of m atoms using k-means. We then use projected gradient descent, alternately training \mathbf{A} and \mathbf{W} to convergence, till neither of them changes. After training, let A_X denote the set of dictionary atoms that have a non-zero contribution towards reconstruction. The sizes of A_X can be found in the table 4.1.

m	$ A_X $
10	4
15	5
20	5
25	6
30	6

Table 4.1: Sizes of the final dictionary \mathbf{A}_X learned after training starting with a super dictionary of m atoms

In an ideal scenario, all data points could have been represented as a convex combina-

tion of 3 atoms (the end points of the line segments on which they lie). We can see that the algorithm does a pretty good job at reducing the size of this super-dictionary. We can see structured sparsity in action here, helping learn the optimal dictionary size, as opposed to using an l_2 norm sparsity, where all the atoms in the original dictionary would have been used in reconstruction of data.

4.6.2 Semi-supervised learning

In this experiment, we try to see if semi-supervision works to learn different small dictionaries for different classes. We also see if structured sparsity can work alongside semi-supervision to reduce the size of the super-dictionary. Hence, we choose 1000 labeled points again, each lying on either of two line segments. The ones lying on the segment joining $(0, 0)$ and $(1, 0)$ are given the label 1 and the ones lying on the segment joining $(0, 0)$ and $(0, 1)$ are given the label 2. To our dataset, we also add 500 more unlabeled data points, lying on either of these 2 line segments.

On starting with 30 atoms, the algorithm could eliminate most of them and only 9 atoms remain at the end of training. Each class dictionary ends up with a size of 4. As expected, there are no shared atoms between class 1 and class 2. The unlabeled data class uses some atoms from the class-1 dictionary and some from the class-2 dictionary, and only uses 1 atom which is not in either of these dictionaries. This distribution of learned dictionary atoms among classes motivates the fact that this technique should be well suited for classification tasks.

Though we are able to achieve convergence on the above small data sets, gradient descent seems to be quite slow at optimising the above cost function. This, coupled with the costly projection operations, makes learning quite slow.

4.7 Experiments on data on unit sphere

For these experiments we use the cost function as used in Section 3.2.1, with the difference of the sparsity term used. The sparsity term incorporates the ideas of structured sparsity and semi-supervision. Hence, the cost function becomes:

$$\sum_{i=1}^n \left\| \sum_{j=1}^m W_{ji} \cos^{-1}(\langle \mathbf{X}_i, \mathbf{A}_j \rangle) \frac{\mathbf{u}_{ij}}{\|\mathbf{u}_{ij}\|} \right\|_{x_i}^2 + \lambda \sum_{k=0}^K \|\bar{\mathbf{w}}_k\|_{2,p,\epsilon}^p \quad (4.2)$$

$$\sum_{j=1}^m W_{ij} = 1, \text{ for } i = 1, 2, \dots, n \quad \text{and} \quad W_{ij} > 0 \quad \forall i \forall j$$

where $\bar{\mathbf{w}}_k$ is as defined in equation 4.1.

4.7.1 Structured sparsity

Again, for this experiment, our aim is to start with an overcomplete dictionary and see if training leads to atoms dropping out (in terms of their weights used for reconstruction). For further description, refer to figure 4.3. We choose 100 uniformly distributed points on a 90° arc of a greater circle on the sphere. We start off by initialising our atoms using k-means with 3 cluster centers. After training, only 2 atoms are used in reconstruction, which are at positions represented by the blue marks.

4.7.2 Semi-supervised learning

Refer to figure 4.4 for the description of this experiment. For this experiment, our data lies along the sides of a triangle on the sphere. We divide data points into three classes depending on the side of the triangle they lie on. We start off with 3 atoms (initialised with k-means) and end up with the 3 learned atoms as shown by the blue points in the figure. Moreover, we find that the data points of a class corresponding to an edge use only the atoms close to the end points of that edge.

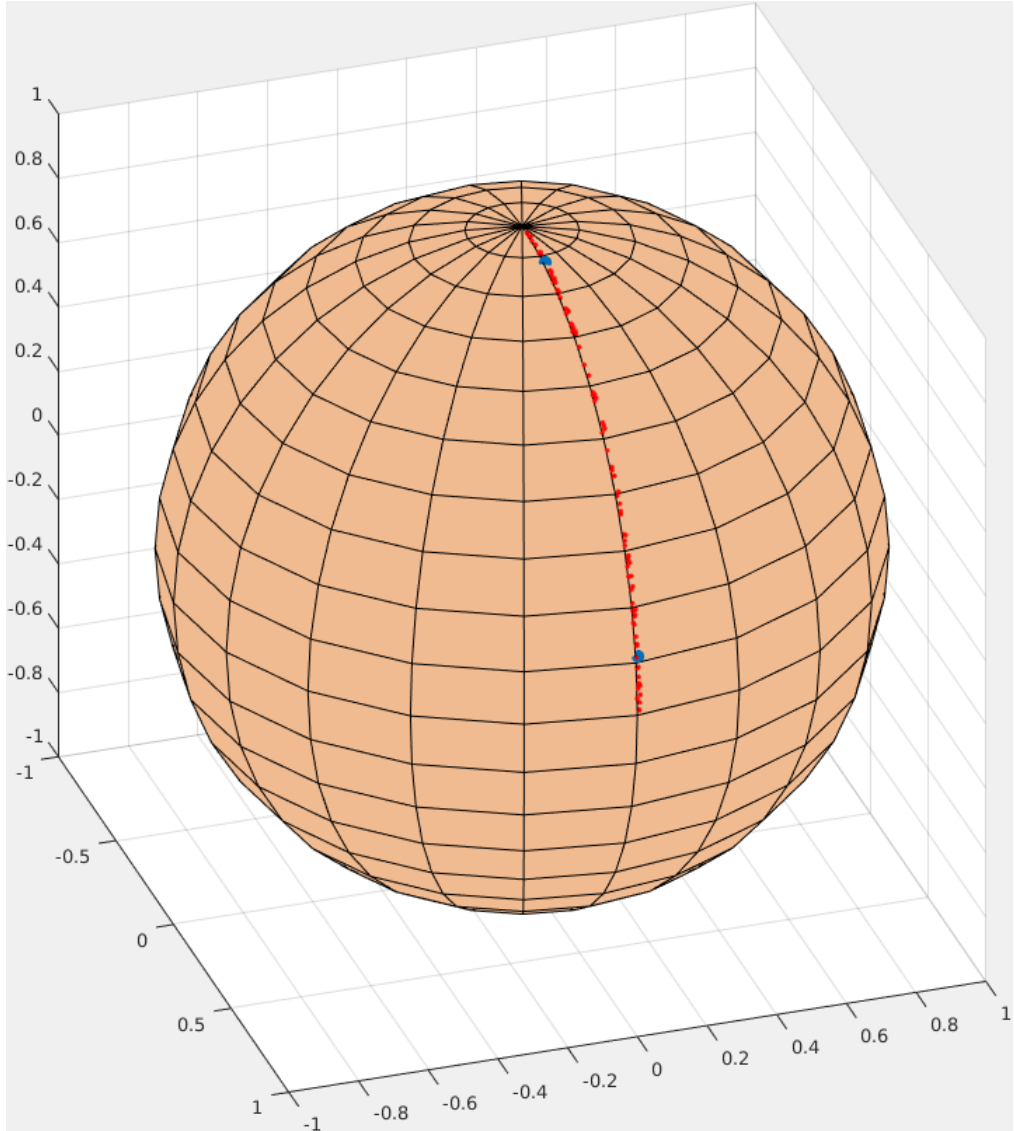


Figure 4.3: Data points and learned atoms on the sphere. The red points are the data points and the two blue points are the two atoms in the learned dictionary

Though our algorithm seems to work fine incorporating both structured sparsity and semi-supervision we face a problem of getting stuck at local minima on training with gradient descent. On throwing in more atoms into the initial dictionary, the final data representation ends up using more and more of those atoms placing them at positions along the edge. For example, on starting with 6 atoms in the experiment in section 4.7.1, the learned dictionary retains 3 atoms, 2 at the end points as before and the third at a point approximately midway between these two.

4.8 Semi-supervised dictionary learning on MNIST dataset

We use the gradient descent algorithm from the previous section to learn a dictionary on the MNIST dataset. Since the algorithm is quite slow to reach convergence, we give up in the affine-positive weight constraint (In [5], the authors used the same approach, i.e., had no added constraints on \mathbf{W}) that requires costly projection steps during gradient descent. We also use only 10000 labeled and 1000 unlabeled images from MNIST and have downscaled each of the images in MNIST by 25%. We start with a super dictionary of size 350, and finally the number of atoms used by class k , can be found in table 4.2.

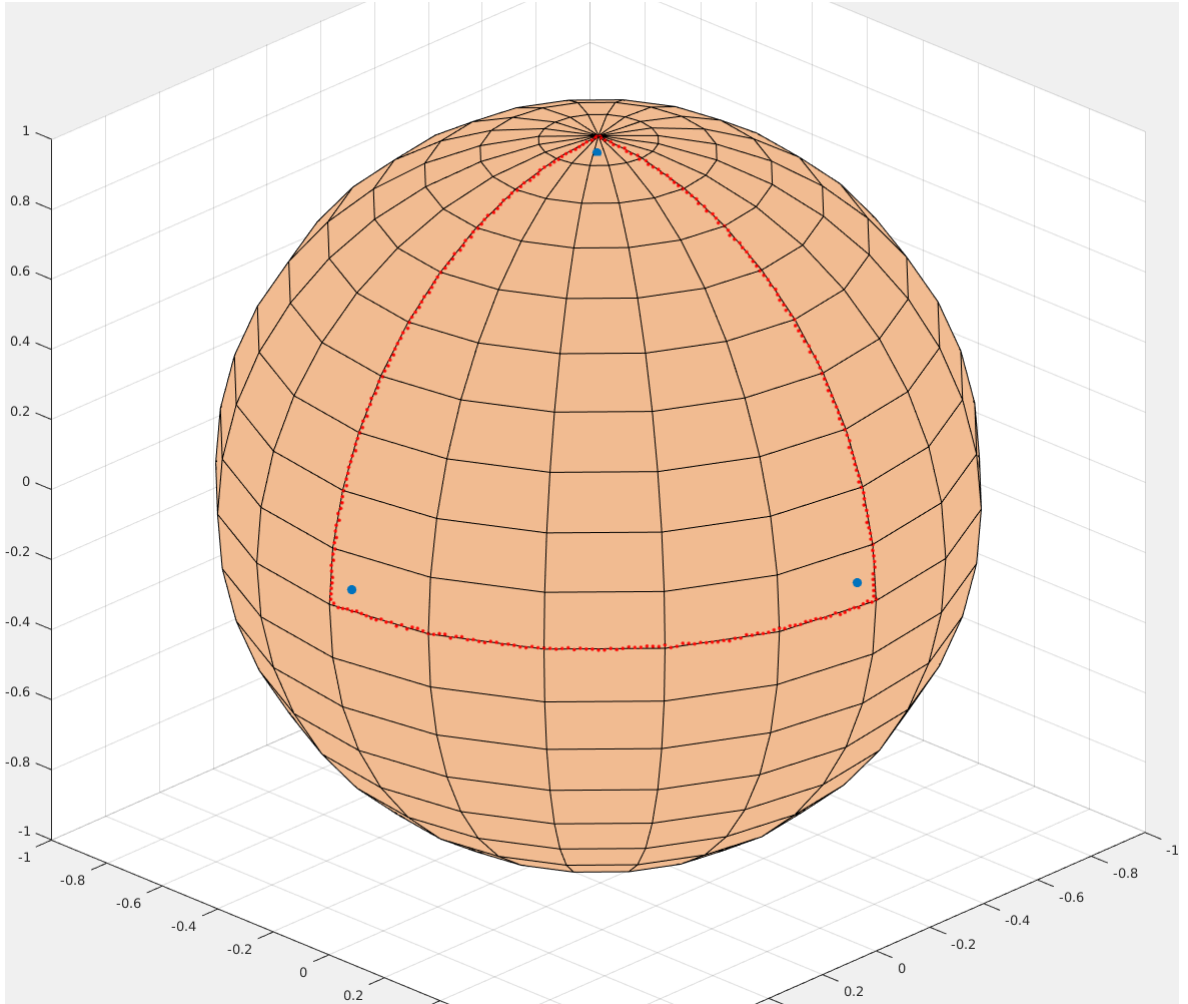


Figure 4.4: Data points and learned atoms on the sphere. The red points are the data points and the blue points are the atoms in the learned dictionary

Class label- k	num columns of \mathbf{A}_k
0	171
1	105
2	186
3	171
4	170
5	172
6	146
7	161
8	187
9	161
unlabeled	226

Table 4.2: Sizes of the final dictionaries \mathbf{A}_k for each of the k classes, learned after training initialised with a super dictionary of 350 atoms

As can be seen, a good number of atoms have been eliminated from the super-dictionary. Also, each class can be distinguished based on which atoms it uses, i.e., no two classes use all the same atoms. Certain other observations that were made during the experiments show that dissimilar numerals have fewer atoms common in their respective class dictionaries as compared to similar numerals, which is an expected result. For example, both '0' and '8' have curved contours. The class dictionaries of '0' and '8' have 125 atoms in common. However, if we compare with a digit with straight contours, like '1', we find that the dictionaries of '1' and '0' share only 68 atoms, thus emphasising the difference between the classes. Another observation that isn't specific to the classification task, is that simple numerals require a smaller number of atoms to reconstruct. This can be seen on comparing the number of atoms in \mathbf{A}_1 and other classes. The digit '1', being a just a vertical straight line in most cases, requires much fewer atoms to reconstruct, as compared to others.

Although the above results are not ideal in the sense that our learning lead the cost function to a local minimum, we can still illustrate that the ideas of structured sparsity and semi-supervision do work.

Chapter 5

Kernel Dictionary Learning

Traditional dictionary learning learns dictionary atoms in the domain of the data, representing data as linear combinations of the atoms. This learning may not be discriminative enough in the so as to learn a linear classifier on the weights, to effectively classify weights. The classifier that needs to be learnt on the weights might be too non-linear to learn in the original data domain. In this case, transforming the data to a higher dimensional domain can make learning linear classifiers possible. This is the key idea behind kernel dictionary learning [1].

5.1 Formulation

The formulation of the problem is quite similar to the what was used in section 2.1, except that the transformed data is now in kernel feature space. Suppose there are n data points in \mathbb{R}^d which are respresented along the columns of the data matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$. The dictionary atoms constitute the columns of the matrix \mathbf{A} and the corresponding weights form the weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$. Let Φ be the transformation function that maps data in \mathbb{R}^d to a high dimensional kernel feature space. The dictionary is learnt such that:

$$\Phi(\mathbf{X}_i) \approx w_{1i}\mathbf{A}_1 + w_{2i}\mathbf{A}_2 + \cdots + w_{mi}\mathbf{A}_m$$

where \mathbf{X}_i is the i^{th} data point and \mathbf{A}_i is the i^{th} dictionary atom in the kernel feature space.

The cost function of the optimization over dictionary atoms and corresponding weights now becomes:

$$C(\mathbf{A}, \mathbf{W}) = \|\Phi(\mathbf{X}) - \mathbf{A}\mathbf{W}\|_2^2 + \lambda f(\mathbf{W})$$

where $f(\mathbf{W})$ is the sparsity term, f being a function that penalises less sparse representations of \mathbf{W} .

For sparsity, we use the ϵ -regularised p -norm, which is defined as :

$$\|v\|_{p,\epsilon} = \left(\sum_{k=1}^d (v_k^2 + \epsilon)^{\frac{p}{2}} \right)^{\frac{1}{p}}$$

where v_k is the k^{th} element of the vector $v \in \mathbb{R}^d$, and ϵ is a small positive number. Note the change in notation as compared to Section 4.4.

Hence, the final optimization problem becomes:

$$\mathbf{A}, \mathbf{W} = \arg \min_{\mathbf{A}, \mathbf{W}} \sum_{i=1}^n \left(\|\Phi(\mathbf{X}_i) - \mathbf{A}\mathbf{W}_i\|_2^2 + \lambda \|\mathbf{W}_i\|_{p,\epsilon}^p \right)$$

under the constraints

$$\sum_{j=1}^m W_{ij} = 1, \text{ for } i = 1, 2, \dots, n$$

$$\|\mathbf{A}_i\|_2 = 1$$

For our purposes, we only use kernels that map data onto the unit sphere in the kernel feature space. Hence, no added constraint to control scaling of the atoms is needed, the atoms are constrained to lie on the unit sphere.

5.2 The kernel trick

When mapping data to high dimensional kernel feature spaces, the function Φ isn't usually calculated explicitly. In many cases it may not even be possible to compute and store it (for

e.g., if the feature space is infinite dimensional). The solution used instead is to define a kernel function κ , such that

$$\kappa(\mathbf{X}_i, \mathbf{X}_j) = \langle \Phi(\mathbf{X}_i), \Phi(\mathbf{X}_j) \rangle$$

where $\langle -, - \rangle$ is the inner product between two points in the kernel feature space.

Thus, distances in the kernel feature space can be computed using the kernel function as:

$$\text{dist}(\Phi(\mathbf{X}_i), \Phi(\mathbf{X}_j))^2 = \kappa(\mathbf{X}_i, \mathbf{X}_i) + \kappa(\mathbf{X}_j, \mathbf{X}_j) - 2 * \kappa(\mathbf{X}_i, \mathbf{X}_j)$$

In most applications, the transformation Φ may not even be explicitly defined. Defining just the kernel κ associated with it is enough. Hence, note that in the previous section, when we write \mathbf{A} , it's just a notation for the group of atoms, in the kernel feature space.

5.3 Formulating the cost function using kernels

In simplifying the cost function, we use the fact that the dictionary atoms lie in the span of data in the kernel feature space. We present the argument as in [1], to justify this. Let us also introduce some new notation, where we use small case letters x_i for representing data points in the original data domain, y_i for $\Phi(x_i)$ and d_i for the i^{th} atom \mathbf{A}_i in the kernel feature space.

The argument goes thus: Suppose all the atoms $\{d_i\}_{i=1}^m$ don't lie in the span of $\{y_i\}_{i=1}^n$. For instance, if all but one atom d_l lie in the above span, then either (i) that atom remains unused, i.e., all weights w_{nl} for that atom d_l are zero, or (ii) the use of atom d_l leads to an increase in the fidelity term as compared to the use of an atom which is its projection onto the span of $\{y_i\}$ because it leads to an additional vector component in the linear combination along a direction orthogonal to the subspace in which the other atoms lie. Thus, if d_l is to be used, replacing d_l with its projection onto the span of $\{y_i\}$ will reduce the objective function and be a better solution. Take another instance where multiple atoms d_k lie outside the span of the data. If there is a space combination of d_k that exactly fits a

data point y_i , i.e., $\left\| y_i - \sum_{j=1}^m d_j w_{ij} \right\|_2 = 0$, then we can use the same weights w_{ij} and the projections of atoms d_k onto the span of $\{y_i\}$ to again get the exact fit. So, we represent each atom as :

$$d_k = \sum_{i=1}^n \gamma_{ik} \Phi(x_i), \text{ where } \forall i : \gamma_{ik} \in \mathbb{R}$$

Hence,

$$\sum_{k=1}^m w_{ik} d_k = \sum_{k=1}^m w_{ik} \sum_{j=1}^n \gamma_{jk} y_j = \sum_{j=1}^n \xi_{ij} y_j$$

where

$$\xi_{ij} = \sum_{k=1}^m w_{ik} \gamma_{jk}$$

Let us also define another matrix \mathbf{U} , where $U_{ij} = \gamma_{ij}$. Thus, we can compute $\{d_k\}_{k=1}^m$ given \mathbf{U} . Our objective function becomes:

$$C(\mathbf{U}, \mathbf{W}) = \sum_{i=1}^n \left(\left\| y_i - \sum_{j=1}^n \xi_{ij} y_j \right\|_2^2 + \lambda \|\mathbf{W}_i\|_{p,\epsilon}^p \right)$$

$$\Rightarrow C(\mathbf{U}, \mathbf{W}) = \sum_{i=1}^n \left(\kappa(x_i, x_i) + \sum_{j=1}^n \xi_{ij}^2 \kappa(x_j, x_j) + 2 * \sum_{j=1}^n \sum_{k=1}^n \xi_{ij} \xi_{ik} \kappa(x_j, x_k) - 2 * \sum_{j=1}^n \xi_{ij} \kappa(x_i, x_j) + \lambda \|\mathbf{W}_i\|_{p,\epsilon}^p \right)$$

We have thus reduced our objective function representing in terms of kernel functions, on pairs of input data. Hence, we need only compute the gram matrix (defined ahead) on input data, and then we can optimise iteratively using gradient descent, alternating between parameters \mathbf{U} and \mathbf{W} . Given data $\{x_i\}_{i=1}^n$, the gram matrix $\mathbf{G} \in \mathbb{R}^{n \times n}$ is defined as $[G_{ij}]$ where $G_{ij} = \kappa(x_i, x_j)$.

Chapter 6

Graph Regularized Sparse Codes

In [7], the authors introduce the idea of using nearest neighbour graphs to impose regularization on sparse codes. The motivation behind this was to learn sparse representations that take into account sub-manifold structure of the data. The authors have mentioned that recent works which incorporate learning from sub-manifold structure of data use the so called locally invariant idea, i.e., nearby points are likely to have similar embeddings. Graph regularised sparse coding (GraphSC) [7] builds a k-nearest neighbor graph to encode the geometrical information in the data. GraphSC uses the graph Laplacian as a smooth operator to preserve the local manifold structure, adding the Laplacian to objective function as a second regularizer (apart from the norm of the weight matrix).

6.1 Formulating graph regularized sparse coding

The regularizer uses the idea that if two points x_i and x_j are close in the intrinsic sub-manifold formed by the data distribution, then their sparse codes (weights of atoms used to encode those data points) w_i and w_j (i^{th} and j^{th} columns of the weight matrix \mathbf{W}), would also be close.

Given a set of d-dimensional data points x_1, x_2, \dots, x_n , we can construct a nearest neighbour graph on n vertices represented by matrix \mathbf{N} (each vertex representing a data

point), where $N_{ij} = 1$ if x_i is among k -nearest neighbours of x_j , otherwise, $N_{ij} = 0$. Degree of x_i is define as $d_i = \sum_{j=1}^n N_{ij}$ and $\mathbf{D} = \text{diag}(d_1, d_2, \dots, d_n)$. Consider the problem of mapping the above weighted graph to the sparse representations given by the matrix \mathbf{W} . A reasonable criterion for choosing a good map is to minimize

$$\frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m (\mathbf{w}_i - \mathbf{w}_j)^2 N_{ij} = \text{Tr}(\mathbf{W}\mathbf{L}\mathbf{W}^T) \quad (3)$$

where $\mathbf{L} = \mathbf{D} - \mathbf{N}$ is the Laplacian matrix. Note that k becomes a user tunable free parameter. By incorporating this regularizer into the objective function for sparse coding, we get:

$$C(\mathbf{A}, \mathbf{W}) = \|\mathbf{X} - \mathbf{A}\mathbf{W}\|_2^2 + \lambda \sum_{i=1}^n \|\mathbf{W}_i\|_{p,\epsilon}^p + \alpha \text{Tr}(\mathbf{W}\mathbf{L}\mathbf{W}^T)$$

We try to use the graphSC regularizer in kernel dictionary learning. In this case, nearest neighbours need to be computed in the kernel feature space to construct the matrix \mathbf{N} . Using values from the gram matrix, distances between each pair of data points can be computed, and based on this, nearest neighbour matrix can then be constructed. The final objective function looks like :

$$C(\mathbf{U}, \mathbf{W}) = \sum_{i=1}^n \left(\left\| y_i - \sum_{j=1}^n \xi_{ij} y_j \right\|_2^2 + \lambda \|\mathbf{W}_i\|_{p,\epsilon}^p \right) + \alpha \text{Tr}(\mathbf{W}\mathbf{L}\mathbf{W}^T)$$

where $y_i = \Phi(x_i)$ and the first term inside the summation can be expanded using kernel functions as in Section 5.3. The above function can be represented in terms of \mathbf{U} and \mathbf{W} after substituting all ξ_{ij} . The function can then be iteratively optimized using gradient descent by fixing \mathbf{U} and \mathbf{W} in alternate steps. We're skipping over the details of how the exact expression is derived.

6.2 Modifications to graphSC implemented

We implemented two variants of the graphSC regularizer:

- The nearest neighbour graph \mathbf{N} was constructed using k -nearest neighbours and then normalised to sum 1 along the columns. This normalisation was done so that the regularization term doesn't get unjustly biased towards smaller k values (in taking k nearest neighbours). In our experiments we'll refer to the graphSC regularizer obtained this way as $k\text{NN}$.
- Instead of a sharp mask of k nearest neighbours, we used smooth gaussian neighbourhood. So the neighbourhood value matrix instead of being full of values $\frac{1}{k}$ and zeros, would now have neighbourhood values which are proportional to $e^{-\frac{\text{dist}(x_i, x_j)^2}{\sigma^2}}$. Here, sigma becomes the free parameter. These values are again normalised to sum 1 along the columns. In our experiments we'll refer to the graphSC regularizer obtained this way as gaussianNN .

Chapter 7

Experiments with graph regularization

We ran tests to compare kernel dictionary learning with graphSC against a baseline of simple kernel dictionary learning as formulated in section 5.3. We study the behaviours of the two different variants of graphSC that we incorporate with our implementation of kernel dictionary learning and compare its performance against the baseline. We have run these tests on the MNIST handwritten digits image dataset.

For each of the tests, we learn a dictionary of 20 atoms from 500 images for each digit. Hence, our training image set size is $500 \times 10 = 5000$ images. We then pool together these dictionaries for different digits (class labels) to create a large dictionary for all class atoms. For classification, a linear SVM is first trained on the sparse codes of the training data. Then, in the testing phase, we learn the sparse codes of test data using the pooled dictionary, and classify using the SVM classifier previously learned.

7.1 Performance of gaussianNN regularization

7.1.1 Gaussian noise

In figure 7.1, we compare performances of our dictionary learning model with on input data with additive gaussian noise of standard deviation 0.2 (0.04 variance). When using p-norm

of the sparse codes (weights for a given data point) in the second term of the objective function, we use $p = 1$.

In figure 7.1, the parameter on the X-axis is a measure of the inverse of the parameter σ , which controls the width of the gaussian used for computing neighbourhood values. We use different σ for each column of the \mathbf{N} matrix. Let the parameter used for column i be σ_i . The relation between σ_i and K is,

$$\sigma_i^2 = \frac{4 * \text{Var}_{j=1}^n(\text{dist}(\Phi(x_i), \Phi(x_j)))}{K}$$

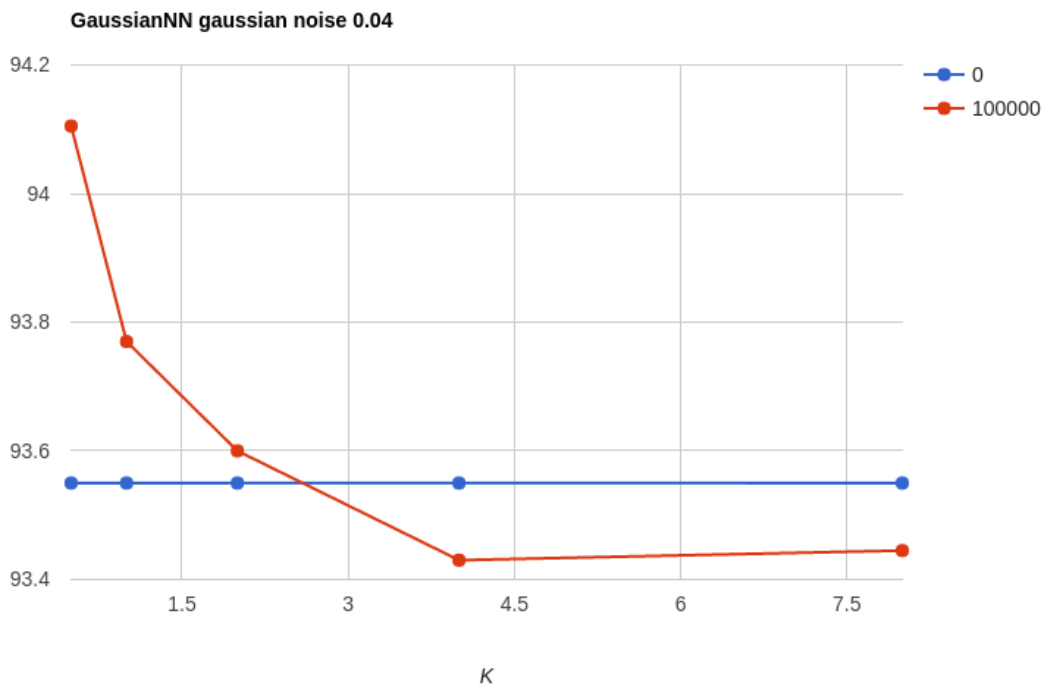
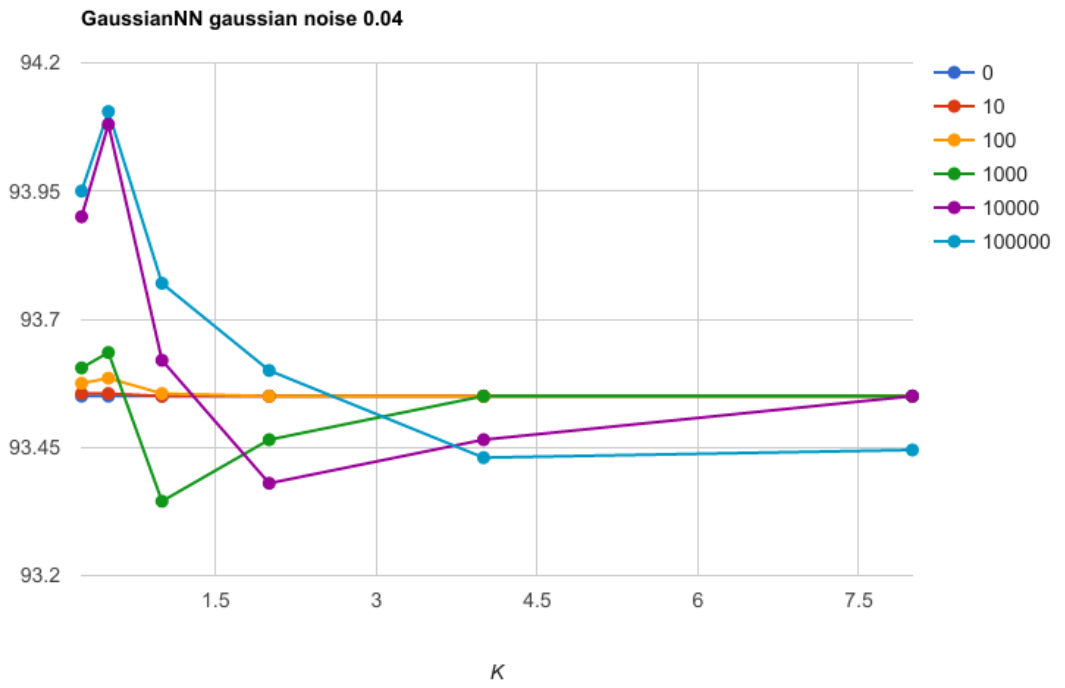
where $\text{dist}(\Phi(x_i), \Phi(x_j)) = \sqrt{\kappa(x_i, x_i) + \kappa(x_j, x_j) - 2\kappa(x_i, x_j)}$

We can see that the general trend across different values of weights α of the gaussianNN regularization term, is that lower values of K perform better. This seems to suggest that higher width of the gaussian used for computing neighbourhoods is preferred.

In figure 7.2, we can see accuracies obtained vs K for a high gaussian noise case where images are corrupted with additive gaussian noise of standard deviation = 0.6 (variance = 0.36). From the trends of different regularization weights α , we see that the peak performance is obtained for relatively larger values of K . This translates to a smaller width of the gaussian used for computing neighbourhood values.

From these, it may be inferred that at low noise levels, performance is optimal when there is contribution from a larger neighbourhood. While for high noise scenario, a large neighbourhood may spoil classification accuracy because the model is lead to think that two dissimilar elements should have close by sparse codes, and hence a smaller neighborhood is preferred.

The variation with different noise levels for optimal parameters $K = 4$ and $\alpha = 10000$ (there parameters aren't optimal for all noise levels, but perform well for most) are shown in figure 7.3. The optimal parameter setting chosen above was for optimal for 0.6 standard deviation gaussian noise. A parameter setting exists where accuracy is better than baseline for 0.8 standard deviation gaussian noise as well.



39
Figure 7.1

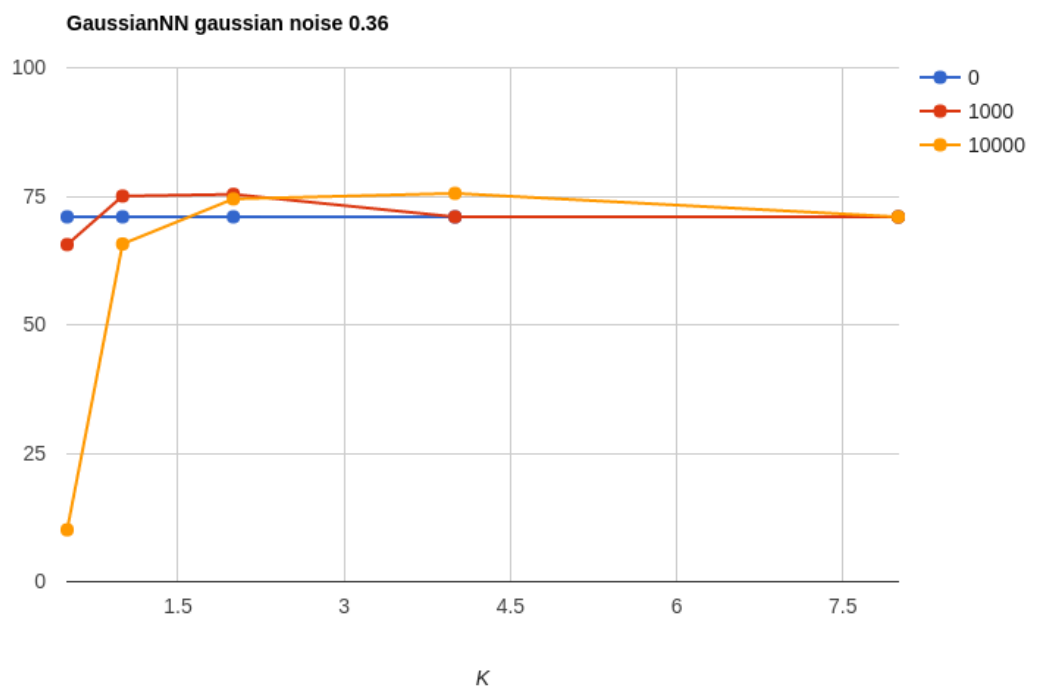


Figure 7.2

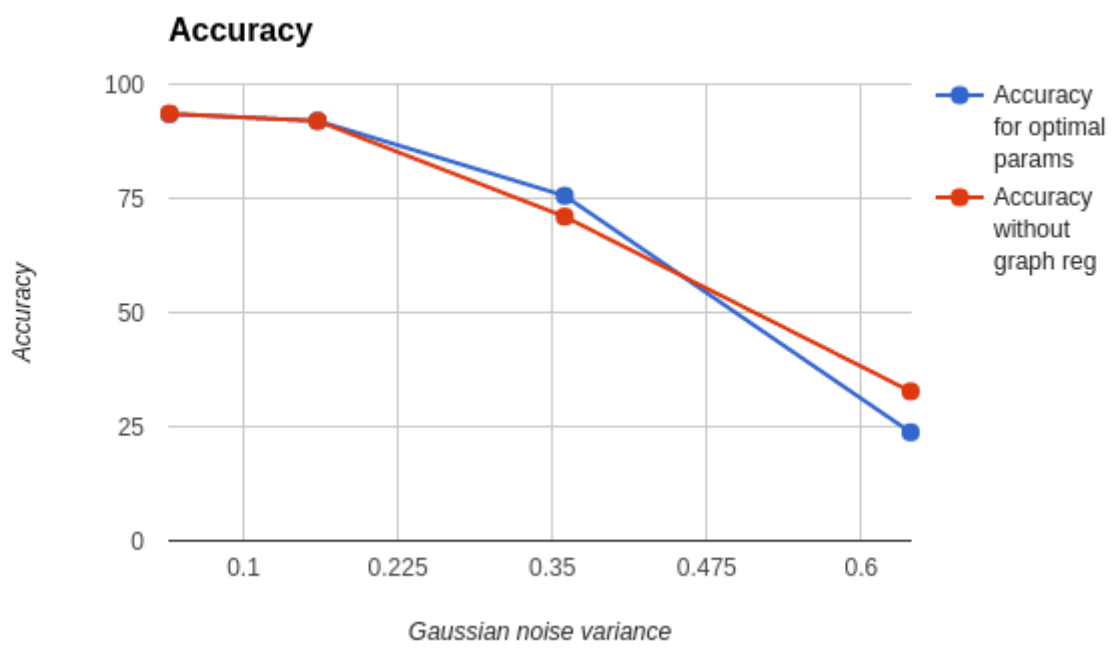


Figure 7.3

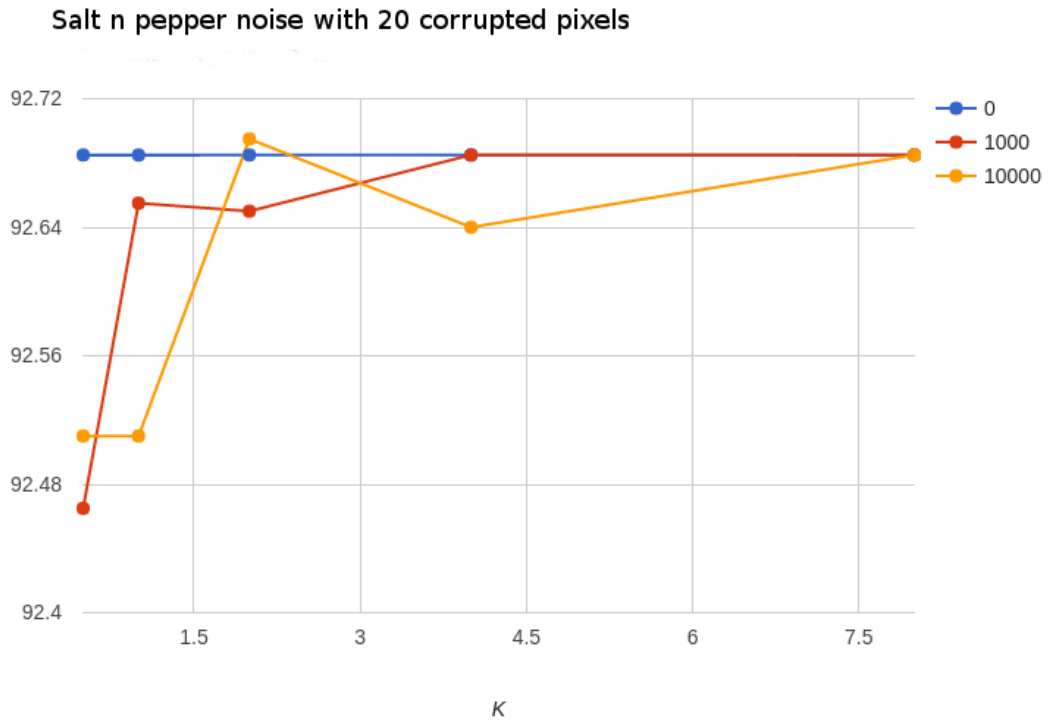


Figure 7.4

7.1.2 Salt and Pepper noise

Each image in our database has 400 pixels. We add salt and pepper noise to the image by corrupting a certain number of pixels randomly to 0 or 1 values. The number of 0s and 1s introduced is roughly equal.

Figure 7.4 shows the variation of performance in terms of accuracy for different parameter set values for a relatively low noise of 20 corrupted pixels. We observe that there is not a big improvement over the baseline for any parameter combination here. From the graphs, we also see that for higher K (or smaller neighborhood) the performance approaches that of the baseline. This seems to suggest that for salt and pepper noise, large neighbourhoods (small K) may not work well even for low noise levels, because even a few pixels, but with high intensities, could alter distances to the extent of bringing different class images

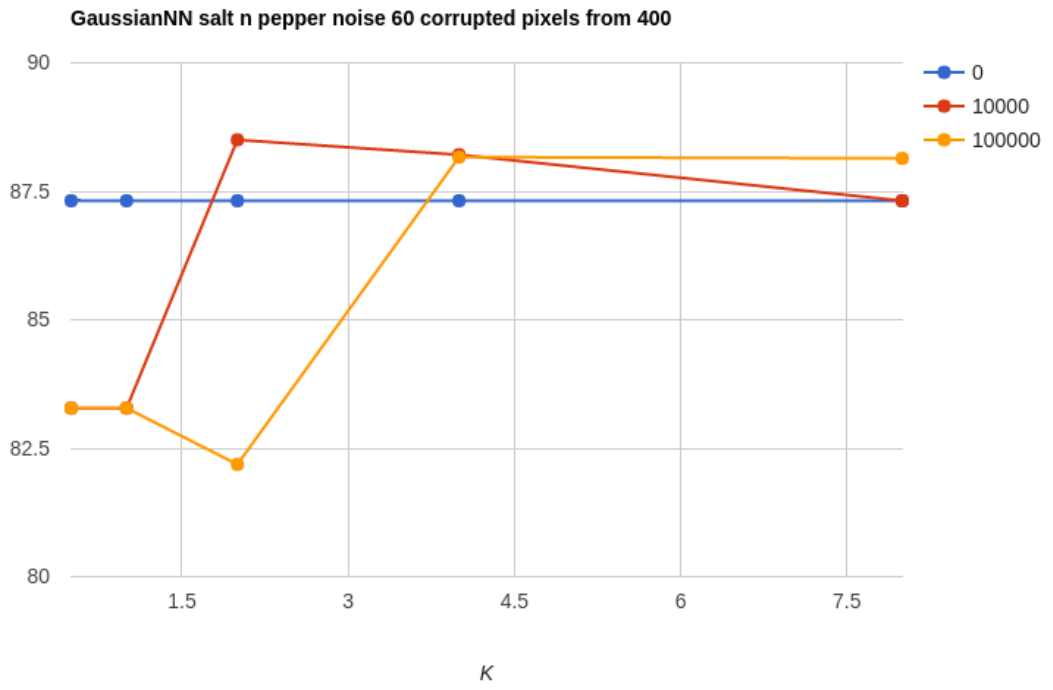
somewhat closer to each other.

In figure 7.5, we show our model performance for different parameter values for high salt and pepper noise scenarios. We can see from the trends that there is a high value of K (small neighborhood), which is optimal.

For 60 corrupted pixels out of 400, we get a performance improvement of approximately 1% in terms of test accuracy over the baseline. For 100 corrupted pixels out of 400, we get a performance improvement of approximately 3% in terms of test accuracy over the baseline.

In figure 7.6, we show a comparison of test accuracy of our model (with optimal parameters $\alpha = 10000, K = 2$) with the baseline for different salt and pepper noise levels. For 200 corrupted pixels on 400, we get a performance improvement of around 1.1% in test accuracy as compared to the baseline.

Note that all the above tests were done with $p = 1$, where p is the parameter in the p -norm used for the sparsity term. We also compared performance of our model and the baseline at $p = 2$, and for optimal parameter settings, our model performed just as good as the baseline, with there being small improvements in some noise instances.



44
Figure 7.5

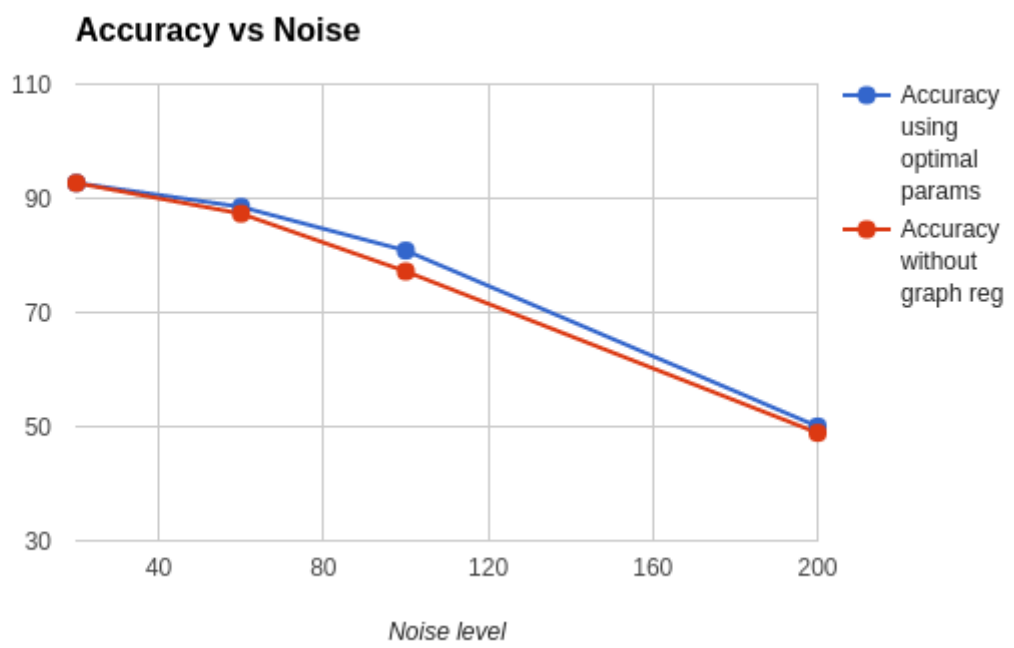


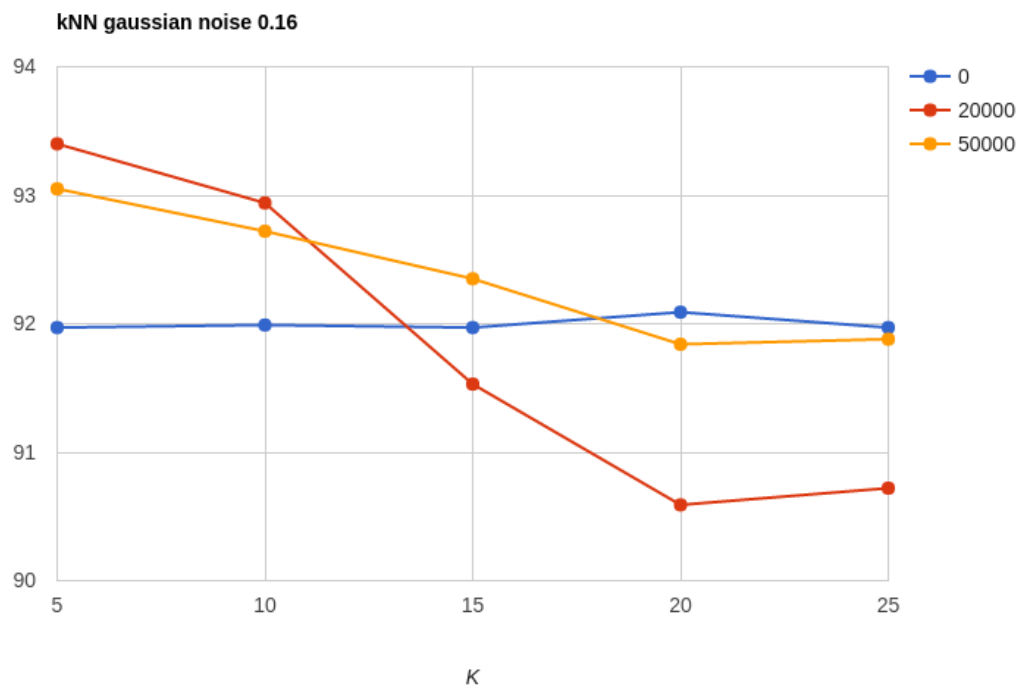
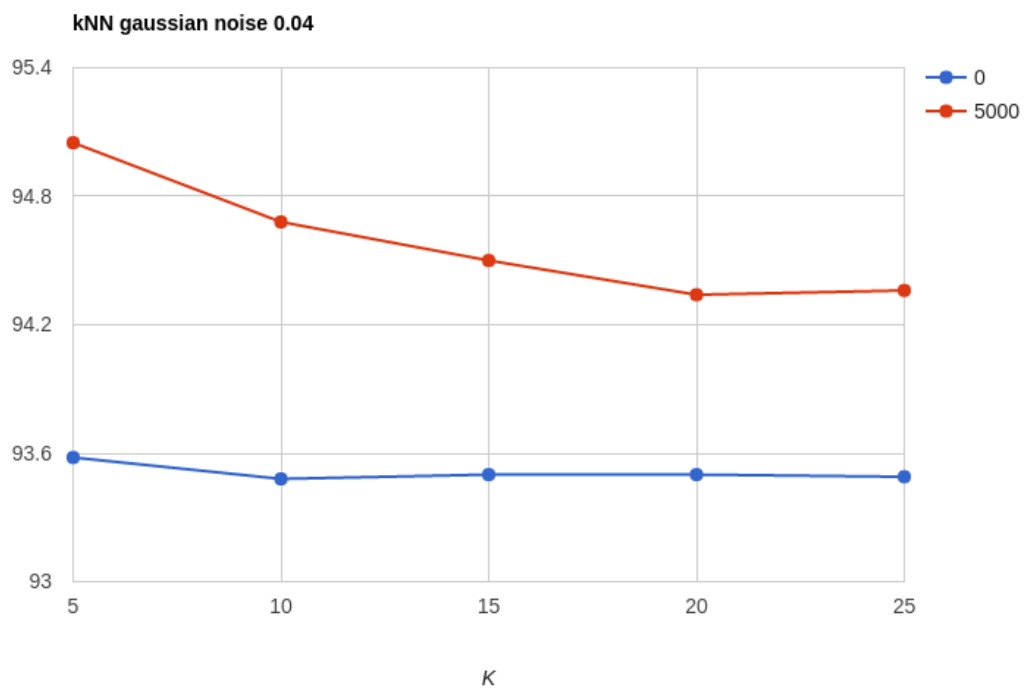
Figure 7.6

7.2 Performance of kNN regularization

In figure 7.7 we see performance of kNN regularisation over different values of K . Note that K here is the number of nearest neighbors to be considered while constructing the nearest neighbour graph N . In the figure, we have shown the performance of our model for two different gaussian noise levels : relatively low noise (standard deviation 0.2) and intermediate-high noise (standard deviation 0.4).

Both seem to show that for lower K , performance in terms of test accuracy is better. This is contrary to what we saw in case of gaussianNN for low noise where high variance in neighbourhood (contribution from a larger neighborhood) was favoured for low noise. This is probably because choosing k nearest neighbours based on distances is quite a non-smooth way of regularising and is prone to noise. Noisy data may cause a point from a different class appear amongst nearest neighbours of a point, and may make a point (perhaps from the same class) which is actually closer to the original point, fall out of its neighbourhood. This effect may be magnified with larger k . This is also perhaps why for very high noise levels, our model doesn't even seem to be competitive with the baseline (Can be seen in figure 7.8 data collected at only 3 values of k).

In figure 7.9 we show a performance comparison of our model with optimal parameters $\alpha = 5000, k = 5$ with the baseline for different noise levels. For the low noise and intermediate noise levels of standard deviation 0.2 and 0.4 respectively, we get an improvement in test accuracy of about 2% over the baseline.



47
Figure 7.7

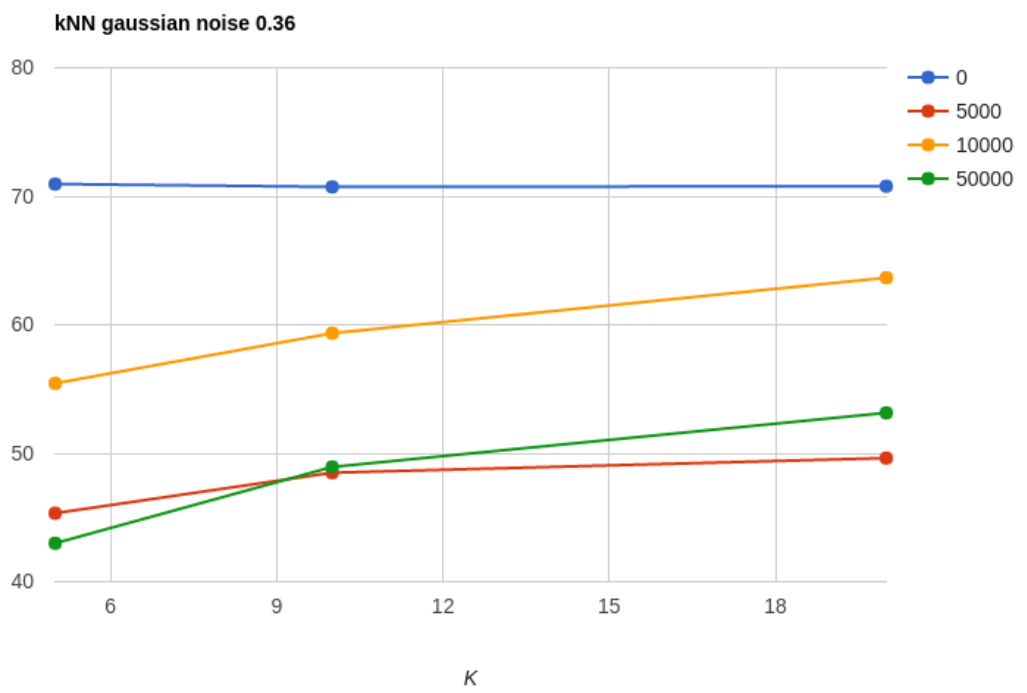


Figure 7.8

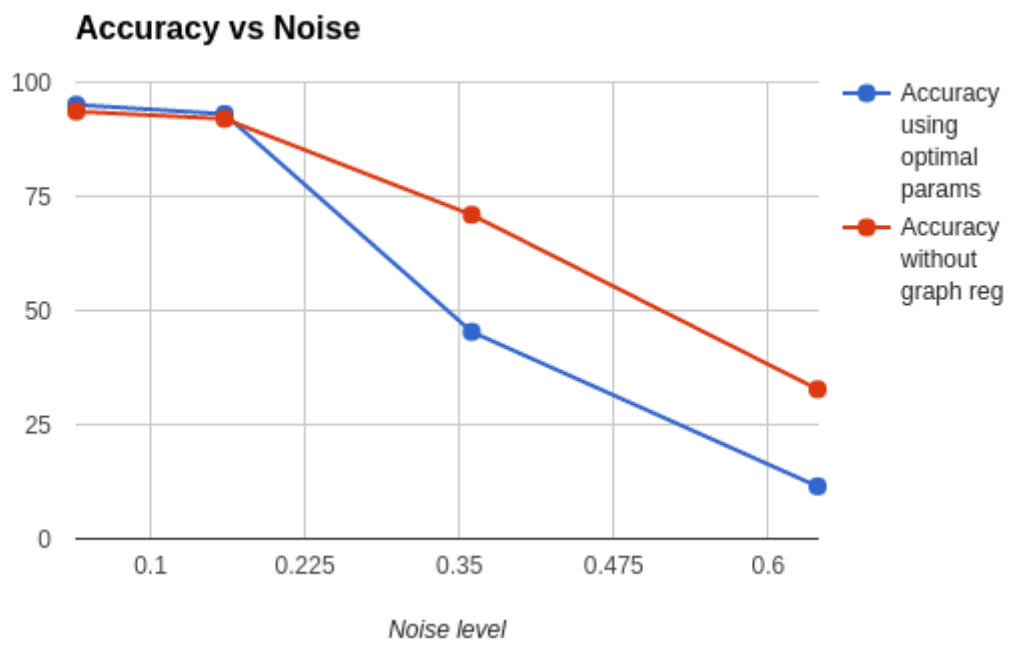


Figure 7.9

7.3 Performance with varying p

In this section, we look at the performance of our model and the base-line by varying the model parameter p . Recall that both our model and the baseline use p -norm of the sparse codes for enforcing sparsity. The idea behind using a smaller p than 1 is to try to model the l_0 norm as closely as possible. This was discussed in detail in section 4.1. The issue sometimes with a smaller p is that the objective function is made more and more non-smooth as we reduce p , and this may introduce many local minima. Many-a-times, with a small p one can only hope to reach a good enough local minimum rather than the absolute global minimum.

For our study, we look at performance at different noise levels : salt and pepper noise with 20 corrupted pixels on 400 in figure 7.10, and gaussian noise with standard deviation 0.2 and 0.4 in figure 7.11.

As is evident from the graphs in figures 7.10 and 7.11, at low and intermediate noise levels (where kNN was able to perform reasonably well), it performed quite better for $p = 2$ than it did for $p = 1$ or lower. This might just be seen as there being scope for improvement at $p = 2$. Though, this is not all. At $p = 2$, test accuracy for our model is even better than baseline at $p = 1$, from which it might be inferred that kNN graph regularization can act as a robustness measure on its own (upto a certain extent of noise). This however, is not seen in gaussianNN where, though the performance improved for $p = 1$ (as compared to $p = 1$ for baseline), $p = 2$ was usually just a little better or no better than baseline with $p = 2$.

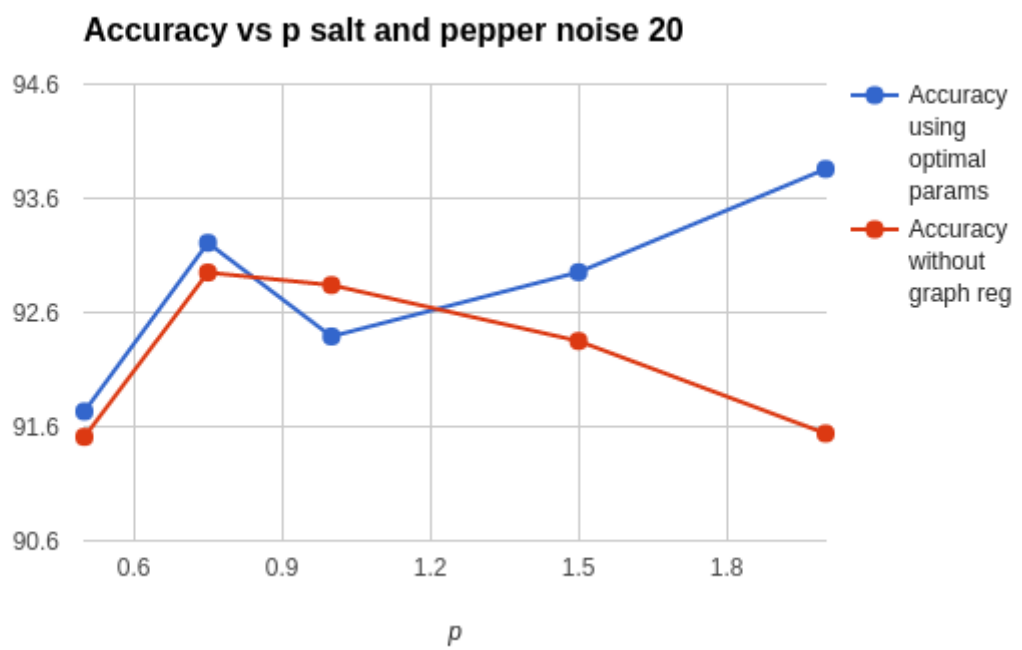


Figure 7.10

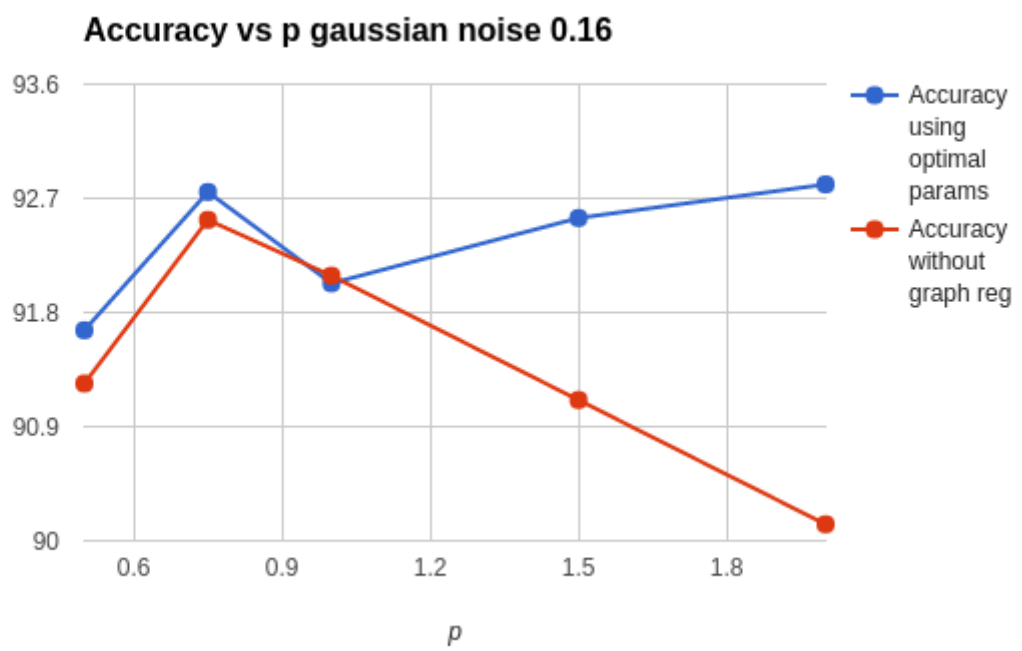
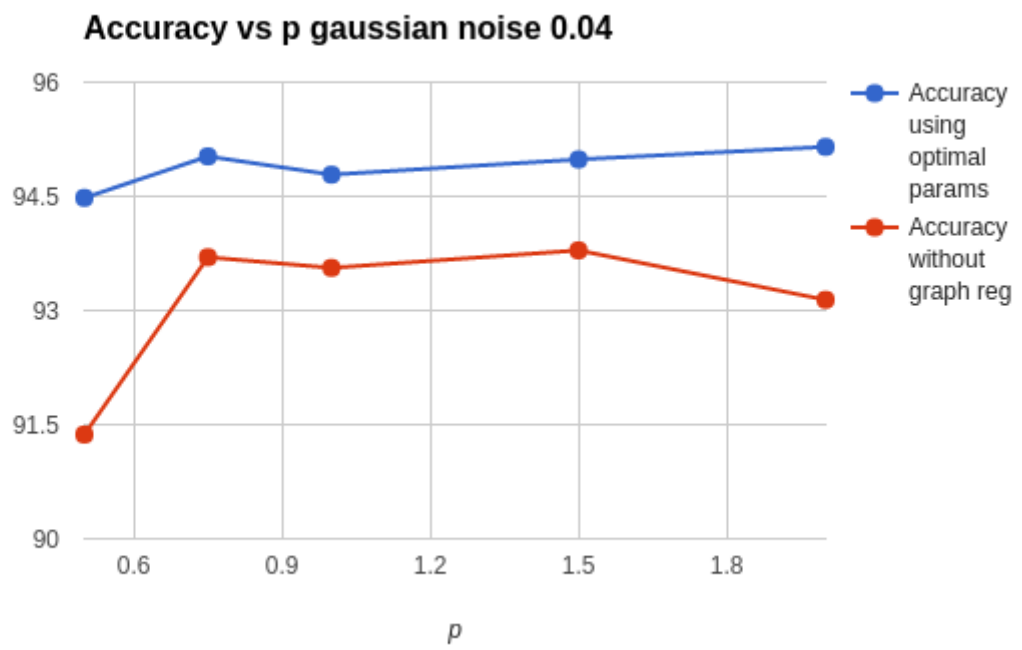


Figure 7.11

7.4 Performance with different kernels

In all preceding tests we used a polynomial kernel of degree 5:

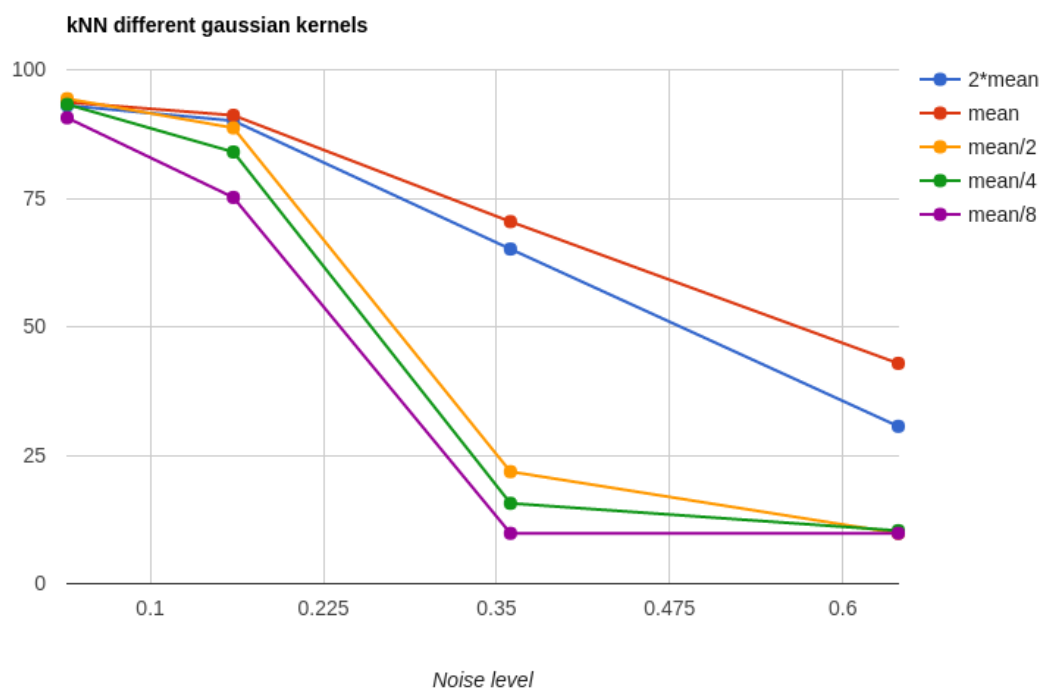
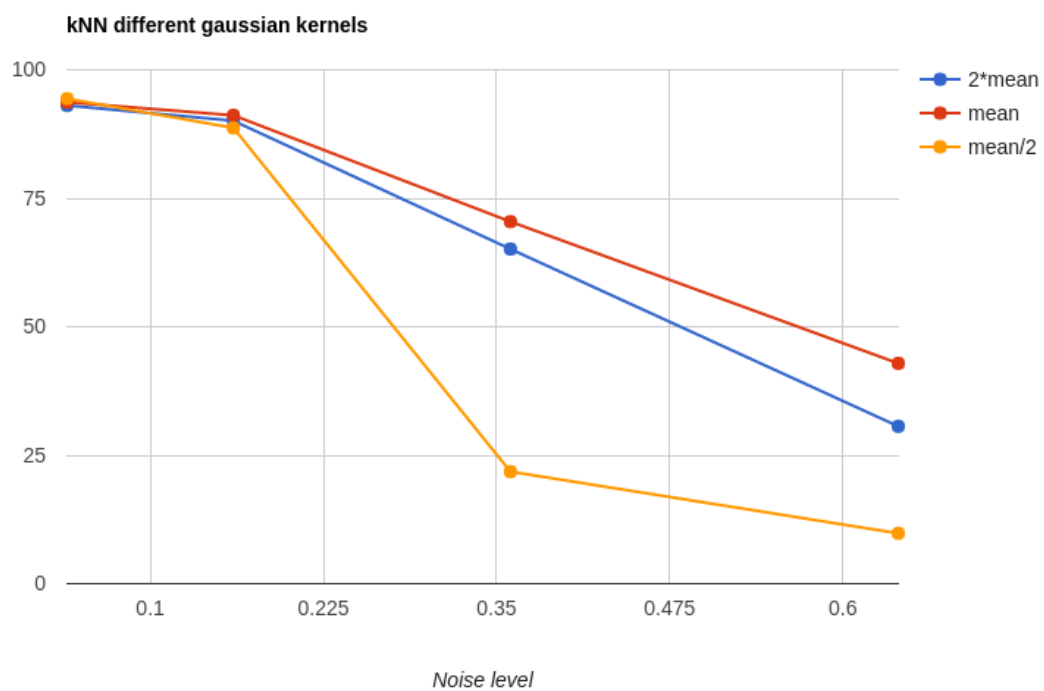
$$\kappa(x_i, x_j) = \frac{(x_i^T x_j + 1)^5}{(\|x_i\|_2^2 + 1)^{2.5} (\|x_j\|_2^2 + 1)^{2.5}}$$

For the purpose of the experiments in this section, we use gaussian kernels of varying standard deviations σ (a measure of width of the kernel). The kernel function looks like:

$$\kappa(x_i, x_j) = e^{-\frac{\|x_i - x_j\|_2^2}{\sigma^2}}$$

A heuristic that we use to tune this parameter is to take the mean of all squared distances between all pairs of data points as an approximate width of the kernel. Let this mean of squared distances be defined as μ . We then vary the parameter σ^2 over values = $[2\mu, \mu, \frac{\mu}{2}, \frac{\mu}{4}, \frac{\mu}{8}]$.

The performances of these different kernels has been shown in figure 7.12. We can see that for low noise, kernel with smaller width ($\sigma^2 = \frac{\mu}{2}$) performs better (about 1% better in test accuracy as compared to the next best, variance = mean). But as noise level increases, a smaller kernel width results in poorer performance because the lesser smoothing resulting from a smaller kernel leaves the model more prone to noise. At higher noise levels, higher kernel width $\sigma^2 = \mu$ gives a better test accuracy.



54
Figure 7.12

Chapter 8

Discussion and Future Work

In section 3.2.3, we used spherical dictionary learning to incorporate the intrinsic geometry of data into the dictionary. In chapter 7, we looked at different experiments trying to incorporate ideas from graphSC in kernel dictionary learning. We found some improvement in performance over the baseline kernel dictionary learning algorithm. Our aim up ahead would be to try and incorporate ideas from graphSC into kernel dictionary learning on the sphere. But before this can be done, we need to run more tests on the our current framework to better understand the behaviour of the graph regularization term.

We tried different variants of graph regularization, and found improvements over baseline, for different variants in different noise scenarios. We need to run more tests to see if we can find a single model that incorporates graph regularization that performs significantly better than the baseline for all scenarios. This would also lead to a better understanding of what happens when the parameters corresponding to the graph regularization term is changed.

Bibliography

- [1] S. P. Awate and N. N. Koushik. *Robust Dictionary Learning on the Hilbert Sphere in Kernel Feature Space*, pages 731–748. Springer International Publishing, Cham, 2016.
- [2] Z. He. Farthest-point heuristic based initialization methods for k-modes clustering. *arXiv preprint cs/0610043*, 2006.
- [3] P. O. Hoyer. Non-negative sparse coding. In *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*, pages 557–565. IEEE, 2002.
- [4] B. Schölkopf, A. Smola, and K.-R. Müller. *Kernel principal component analysis*, pages 583–588. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.
- [5] H. Wang, F. Nie, W. Cai, and H. Huang. Semi-supervised robust dictionary learning via efficient l-norms minimization. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2013.
- [6] Y. Xie, J. Ho, and B. Vemuri. On a nonlinear generalization of sparse coding and dictionary learning.
- [7] M. Zheng, J. Bu, C. Chen, C. Wang, L. Zhang, G. Qiu, and D. Cai. Graph regularized sparse coding for image representation. *IEEE Transactions on Image Processing*, 20(5):1327–1336, 2011.