Greedy algorithms for Sparse Dictionary Learning

Varun Joshi

26 Apr 2017

Background. Sparse dictionary learning is a kind of representation learning where we express the data as a sparse linear combination of an overcomplete basis set. This is usually formulated as an optimization problem which is known to be NP-Hard. A typical solution uses a two-step iterative procedure which involves either a convex relaxation or some clustering based solution. One problem with the typical approach is that it may be difficult to interpret the meaning of dictionary atoms.

Aim. Present a greedy framework to solve the sparse dictionary learning problem which results in an interpretable dictionary. Compare the performance of the resulting algorithms with the K-SVD algorithm. Apply the greedy framework to compression and classification tasks.

Data. We use five benchmark datasets to test the algorithms: 1) ionosphere dataset - measurements of radar signals returned from the ionosphere; 2) CTG dataset - fetal heart rate and uterine contraction features derived from cardiotocograms; 3) WDBC dataset - features derived from the image of fine needle aspirate of a breast mass; 4) MNIST database - greyscale labeled images of handwritten digits; 5) 20newsgroups dataset - 20,000 newsgroup messages partitioned across 20 different newsgroups like politics, religion, etc.

Methods. We present three greedy algorithms named **dp**, **dl**, and **dch** to solve the problem in which the sparse code is learned as a direct consequence of the dictionary learning process. The algorithms start with a random data point as the initial dictionary and at each iteration add to the dictionary the data point farthest from it until a specified error threshold is reached. The algorithms dp, dl, and dch differ in how the "distance" of a point to a dictionary is defined. The representation of a point is then naturally given by its projection on the dictionary.

Results. dch gives the best compression ratio of 27.5% with average sparsity 2.8 using the max stopping rule; dl gives the best compression ratio of 21.4% with average sparsity 1.8 using the mean stopping rule. KSVD outperforms both in terms of compression ratio with an average value of 20.1% while fares worse in terms of sparsity with an average value of 4.9.

Conclusions. The greedy framework successfully solved the sparse dictionary learning problem with a sparser representation than KSVD; however, overall compression given by KSVD was better. An advantage of our approach is that data points are chosen as dictionary atoms which facilitates qualitative interpretation of the dictionary. Also, since the labels of the atoms are available we can use the dictionary for various supervised learning tasks.

Keywords: Dictionary learning, Sparse coding, Representation learning

DAP Committee members:

Avrim Blum ⟨*avrim@cs.cmu.edu*⟩ (Computer Science Department);
Roni Rosenfeld ⟨*roni@cs.cmu.edu*⟩ (Machine Learning Department);

1 Introduction

Sparse dictionary learning is a kind of representation learning where we express the data as a linear combination of a fixed set of overcomplete basis elements. This has applications in compression, image denoising, topic modeling, etc. The sparse dictionary learning framework consists of the following two problems: **Dictionary learning**: learning the set of basis elements from the data, and **Sparse Coding**: learning the representation of the data in terms of the dictionary. The goal is to get a representation that is sparse.

Sparse dictionary learning is usually formulated as an optimization problem which is known to be NP-Hard. The typical solution of this problem involves a two-step iterative procedure in which we solve the dictionary learning and sparse coding problem alternatively. One class of algorithms solve the convex relaxation of the problem [Olshausen and Field, 1997], [Mairal et al., 2009]. This is done by approximating the sparsity of the representation by the L1-norm of the representation vector. Both the sparse coding and dictionary learning stage are then solved using one of the algorithms for convex optimization. However, this is based on certain assumptions on the distribution of representation matrix and noise. Another category of algorithms use a clustering based approach to solve the problem. One such popular algorithm is K-SVD from [Aharon et al., 2006]. K-SVD uses a similar two step optimization procedure where the sparse coding stage is solved approximately using Orthogonal Matching Pursuit [Pati et al., 1993]. An iteration of dictionary learning stage in K-SVD involves updating each dictionary atom sequentially by computing the SVD of the restricted error matrix (reconstruction error matrix without using the current dictionary atom). However, K-SVD requires us to specify the dictionary size K beforehand and is not guaranteed to converge. One of the drawbacks of the previous approaches is that it may be difficult to provide a meaningful interpretation to dictionary elements since they are made up of some combination of data points e.g., when data points represent patients.

In this project, we present algorithms based on [Blum et al., 2015] to solve the dictionary learning and sparse coding problem simultaneously. We present three variants of the algorithm named **dp**, **dl**, and **dch** which do not make any assumptions on the distribution of the representation coefficients or noise. These algorithms use a greedy approach to learn the dictionary in which the sparse code is learned as a direct consequence of the dictionary learning process. The algorithms do not require us to specify the dictionary size beforehand. The dictionary is made up of actual data points and each data point is represented as a convex combination of dictionary atoms. Although, this formulation may result in a less compact dictionary/representation it has several advantages. Choosing actual data points as atoms facilitates qualitative interpretation of the dictionary. A representation using convex combinations is a natural soft clustering method. Further, since the class labels for atoms of the dictionary are available we can use the dictionary as a set of representative points in a nearest neighbour classification/regression scheme. In addition, the algorithms require only dot product access to data and hence can be kernelized.

2 Problem Statement

The goal of the project is to explore a new approach to solve the sparse dictionary learning problem. Specifically, we adapt the algorithm presented in [Blum et al., 2015] to solve the sparse coding and dictionary learning problem simultaneously. We then compare the resulting algorithms with the K-SVD algorithm in terms of the dictionary size, compression ratio, and average sparsity. We also explore the applications of these algorithms to various machine learning tasks.

3 Background and Related Work

Sparse dictionary learning is a kind of representation learning where we express the data as a linear combination of an overcomplete basis set. There are two problems that need to be solved in order to do this: learning the set of basis vectors from the data (Dictionary Learning problem), and learning the representation of the data in terms of the basis (Sparse Coding problem). The set of basis vectors is called the *dictionary* and each element of the dictionary is called an *atom*. The set of coefficients in the linear combination for a data point form a *coefficient/representation* vector. The goal is to get a representation vector that is sparse. This can be expressed via the following optimization problem:

$$\begin{aligned} \underset{x\in\mathbb{R}^k,\, D\in\mathbb{R}^{d\times k}}{\text{minimize}} \quad & \|x\|_0 \\ \text{subject to} \quad & \|y - Dx\|_2 \leq \epsilon,\ k \leq K. \end{aligned} \tag{1}$$

where $y \in \mathbb{R}^d$ is a data point/signal, $D \in \mathbb{R}^{d\times k}$ is the dictionary, $x \in \mathbb{R}^k$ is the representation vector under D, ϵ is an error tolerance, and K is a restriction on dictionary size. Some other equivalent ways of formulating the above problem are:

1. Get the sparsest representation vector subject to reconstructing the signal exactly with some restriction on the dictionary size.

2. Minimize the reconstruction error subject to a restriction on dictionary size and sparsity of representation vector.

3. Minimize a linear combination of reconstruction error and L0-norm of representation vector with dictionary size as a constraint.

A more detailed discussion of the various formulations and how they relate to each other is available in [Tropp, 2004].

If we fix the dictionary D in eq (1), we obtain the sparse coding problem while fixing the representation vector x gives us the dictionary learning problem. Hence, the typical way of solving the sparse dictionary learning problem involves a two-step iterative procedure. At each iteration, first the sparse coding problem is solved for a fixed dictionary and then the dictionary is updated using the obtained sparse code [Tosic and Frossard, 2011]. Therefore, in section 3.1, we first review algorithms for solving the sparse coding problem. In section 3.2, we review algorithms for sparse dictionary learning which utilize one of the methods from section 3.1 in the sparse coding stage.

3.1 Algorithms for Sparse Coding

The different formulations of the sparse coding problem, as stated above, are NP-Hard [Tropp, 2004]. This has led to the development of two main approaches to solve the problem: approximate solutions using greedy algorithms, and solving the convex relaxation of the problem. A greedy approach to solve the sparse coding problem is the Matching Pursuit (MP) algorithm [Mallat and Zhang, 1993]. At each iteration, MP finds the atom with the highest correlation with signal residual and removes from the residual its projection on the atom. This process is continued until a stopping criterion is met. One of the problems with MP is that it may select the same atom multiple times. Orthogonal Matching Pursuit (OMP) [Pati et al., 1993] corrects this issue by computing the orthogonal projection of the signal on the subspace of all atoms selected so far. This procedure ensures that each atom is selected at most once and results in faster convergence. Both MP and OMP converge to a locally optimal solution in the general case.

A different approach to solve the sparse coding problem is by doing a convex relaxation of the optimization problem. This is accomplished by replacing the L0-norm of the representation vector with the L1-norm in the objective or constraints, as the case may be. This modification makes the problem convex which could be solved in polynomial time by standard algorithms e.g., gradient descent, etc. Two popular formulations are the Basis Pursuit (BP) [Chen et al., 2001] and the Lasso [Tibshirani, 1996]. BP finds that solution which reconstructs the signal perfectly and is the sparsest in the L1 sense. Lasso minimizes a linear combination of reconstruction error and sparsity (in the L1 sense). A variety of convex optimization algorithms exist for solving the BP and the Lasso efficiently. There are some other approaches for sparse coding based on e.g., Bayesian learning. More details about these approaches can be found in [Tropp, 2004].

3.2 Algorithms for Sparse Dictionary Learning

There are two main categories of algorithms for the sparse dictionary learning problem: 1) Probabilistic learning methods, and 2) Clustering based methods.

The probabilistic learning methods are based on the idea of approximating the maximum likelihood estimate of the dictionary under sparseness priors on the coefficients. Under suitable assumptions on the prior distribution of coefficients and noise (refer [Olshausen and Field, 1997] for details), this formulation reduces to minimizing a linear combination of the representation error and L1-norm sparsity of representation vector. This formulation was first given by [Olshausen and Field, 1997]. The optimization problem is then solved alternatively for the dictionary and the sparse code using gradient descent. The Method of Optimal Directions (MOD) [Engan et al., 1999] solves the L0-norm constrained version of the same problem. The sparse code is obtained by OMP and dictionary update is done by the following closed-form solution: $D = YX^{+}$, where X^{+} represents the pseudoinverse of the representation matrix. This algorithm is not guaranteed to converge. Further, the computational complexity of matrix inversion makes it inefficient in high dimensions. Recently, some algorithms [Mairal et al., 2009] have been designed which update the dictionary in an online fashion which make them more efficient for large high-dimensional datasets.

A different approach for sparse dictionary learning is based on vector quantization. K-means clustering provides a natural solution where the cluster centers form the atoms of the dictionary and

sparsity is obtained by default as each data point is approximated by a single cluster center. This has the limitation of giving an inflexible representation where each signal is reconstructed using a single atom. The generalization of K-means clustering is the K-SVD algorithm given by [Aharon et al., 2006]. The KSVD algorithm updates each dictionary atom sequentially by getting the best rank-1 approximation of the restricted error matrix using the SVD. The columns of the restricted error matrix are the signal residuals when the contribution of the current atom is removed. KSVD uses OMP in the sparse coding stage. For a more detailed review of dictionary learning methods refer [Tosic and Frossard, 2011].

4 Data

We use five benchmark datasets to compare the performance of the algorithms in terms of the metrics - dictionary size, compression ratio, and average sparsity. The idea is to use data of different types (image, text, etc.) and collected from a variety of problem domains. For all the datasets we scale the feature values to be between 0 and 1 before performing any experiments. The description of the datasets is as follows.

- The MNIST database consists of 8-bit greyscale labeled images of handwritten digits (ranging from 0-9) of size 28x28 pixels. It contains 60,000 training examples and 10,000 test examples. The data is available from this website: `http://yann.lecun.com/exdb/mnist/`. We work with a random subset (of size 2000) of the training set of the MNIST database.

- The 20 Newsgroups dataset is a collection of around 20,000 newsgroup messages partitioned across 20 different newsgroups. Each newsgroup corresponds to a topic e.g., politics, religion, etc. The data is available from this website: `http://qwone.com/~jason/20Newsgroups/20news-bydate-matlab.tgz`. For this dataset, we first remove stopwords based on the list provided here: `https://github.com/vjoshi345/dap/tree/master/data/stopwords.txt`. Then, we use the top 500 most frequently occurring words in the newsgroups to create a bag of words model.

- The ionosphere dataset contains measurements of radar signals returned from the ionosphere showing the presence/absence of some structure in the ionosphere. The data has 351 examples of signals each of which is made up of 34 continuous attributes. The label is "good"/"bad" indicating the presence/absence of structure. The data is available from this website: `https://archive.ics.uci.edu/ml/datasets/Ionosphere`.

- The CTG dataset consists of measurements of 21 different fetal heart rate (FHR) and uterine contraction (UC) features on cardiotocograms (CTG). There are 2126 instances of CTGs each of which is classified into one of 10 morphologic patterns and one of 3 fetal states. The data is available from here: `https://archive.ics.uci.edu/ml/datasets/Cardiotocography`.

- Each instance of the WDBC dataset contains 30 features describing different characteristics of cell nuclei. The features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. There are a total of 569 instances each of which is classified as

malignant/benign tumor. The data is available from here: `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)`.

5 Methods

In this section we describe our approach to solve the sparse dictionary learning problem. We use a greedy algorithm to solve the problem which is based on two main ideas from [Blum et al., 2015]. Let $convex(Y)$ denote the convex hull of a set of points Y. Then, [Blum et al., 2015] present an algorithm which given an error threshold ϵ, computes in polynomial time, a subset D of Y such that $convex(D)$ ϵ-approximates $convex(Y)$. Further, they show that every point of Y can be ϵ-approximated by a convex combination of points of D that is $O(1/\epsilon^2)$-sparse.

5.1 Design

We use the algorithm from [Blum et al., 2015] to learn the dictionary D as a subset of the data points Y. A sparse representation for points in Y in terms of D is also guaranteed by the theorem above. The general framework is presented in algorithm 1.

Algorithm 1: Greedy framework for sparse dictionary learning

Data: Dataset Y and error tolerance ϵ.
Result: Dictionary matrix D and representation matrix X.
begin
  Choose a random data point from Y as the first atom (column) of D;
  while *stopping_function(Y, D)* $>$ ϵ **do**
    $s \leftarrow \underset{y \in Y}{\text{argmax}}\ dist(y, D)$;
    Update D by adding s as the next atom (column);
  end
  Project data points on D and express them as convex combinations of dictionary atoms;
  Collect the coefficients of the convex combination into columns of X;
end

A specification of $dist(y, D)$ and $stopping\_function(Y, D)$ completes the description of an algorithm to solve sparse dictionary learning. We present three algorithms named dp, dl, and dch which differ in how the distance to the dictionary function $dist(y, D)$ is defined. For a query point y, the distance to the dictionary is defined as follows:

- **dp**: distance from y to the closest point in D
- **dl**: distance from y to the closest line segment induced by pairs of points in D
- **dch**: distance from y to $convex(D)$

The approximate distance to the convex hull of a set of points from a query point can be found using algorithm 2.

Algorithm 2: Approximate distance to convex hull (dch algorithm)

input : Query point q, a set of points P, number of iterations n.
output: Approximate distance d and projection t from q to $convex(P)$.
begin
  $t \leftarrow \underset{p \in P}{\operatorname{argmin}} \|q - p\|_2$;
  $d \leftarrow \|q - t\|_2$;
  for $i \leftarrow 2$ **to** n **do**
    $v \leftarrow q - t$;
    $p \leftarrow \underset{s \in P}{\operatorname{argmax}} < v, s >$;
    Update t to be point closest to q on the line segment $p\,t$;
    $d \leftarrow \|q - t\|_2$;
  end
end

In addition, we also use algorithm 3 to compute distance to convex hull from a query point. The algorithm for sparse dictionary learning in this case is termed **dchperceptron**.

Algorithm 3: Approximate distance to convex hull (dchperceptron algorithm)

input : Query point q, a set of points P, number of iterations n.
output: Approximate distance d and projection t from q to $convex(P)$.
begin
  $t \leftarrow \underset{p \in P}{\operatorname{argmin}} \|q - p\|_2$;
  $d \leftarrow \|q - t\|_2$;
  for $i \leftarrow 2$ **to** n **do**
    $s \leftarrow \underset{p \in P}{\operatorname{argmin}} \|iq - (i-1)t - p\|_2$;
    if $\|q - \frac{(i-1)t+s}{i}\|_2 < d$ **then**
      $t \leftarrow \frac{(i-1)t+s}{i}$;
      $d \leftarrow \|q - t\|_2$;
    end
  end
end

The choice of $stopping\_function(Y, D)$ reflects the constraint of our optimization problem. We use the following two stopping criteria for the algorithms during the dictionary learning process:

- **max** stopping rule: $stopping\_function(Y, D) = \max_{y \in Y} dist(y, D)$
- **mean** stopping rule: $stopping\_function(Y, D) = \frac{1}{|Y|} \sum_{y \in Y} dist(y, D)$

5.2 Sparsity of representation

dp approximates each point using the nearest point in the dictionary. So, the representation is sparse by default (sparsity=1). Similarly, for dl each point is expressed using a convex combination of the two end points of the closest line segment formed by pairs of points in the dictionary. This makes the representation sparse with default sparsity=2.

For dch and dchperceptron, note that in algorithms 2 and 3 the number of iterations n is the sparsity of representation. So, in the sparse coding stage we consider the distance of points from $convex(D)$ for different levels of sparsity. For each y we compute $dist(y, D)$ (at the current sparsity level) and consider it for higher sparsity only if it satisfies the following condition:

- For max stopping rule: $dist(y, D) > \epsilon$
- For mean stopping rule: $dist(y, D) > \max(2\epsilon - \max_{p \in Y} dist(p, D), 0)$

These criteria ensure that we are judicious in the use of dictionary atoms to express a point. In particular, we greedily try to express a point with as few atoms as possible. Note that for the mean stopping rule, the above criteria may result in violation of the constraint of our optimization problem. However, in our experiments we found that this criteria was stringent enough that this problem did not arise.

5.3 Computational complexity

It is interesting to consider the per iteration computational complexity of the greedy algorithms. For each iteration of dp, we compute the distance of points outside the dictionary to the newly added point in the dictionary to update their representation. This takes $O(nd)$ computation for n points of dimension d. Similarly, dl requires us to compute the distance to the newly created line segments (due to the addition of a point) in the dictionary. This takes $O(ind)$ computation where i is the current dictionary size. dch (and dchperceptron) are slightly different because they have the additional parameter of maximum allowed sparsity s for a point (denoted by number of iterations n in algorithms 2 and 3) which would multiply the computation required by a factor. However, s will generally be a small fixed constant which does not affect the asymptotic complexity. Hence, each iteration of dch (or dchperceptron) involves finding (for a constant number (s) of times) the next best dictionary atom to maximally reduce the error vector in the representation for a query point. This takes $O(ind)$ time for n points.

We can also consider the amount of computation required in terms of calls to the "distance to dictionary" function for each of the algorithms. Since, for each iteration, we need to compute the distance to the dictionary exactly once for all points outside of it, the cost in terms of function calls is $O(n)$. We summarize these observations in table 1. Table 1 also lists the time required to perform k iterations which could be obtained by summing over the per iteration complexity. For completeness, we also include the cost of the K-SVD algorithm which can be found in [Rubinstein et al., 2008].

Algorithm	**Per iteration**	**"distance" function calls**	k **iterations**
dp	$O(nd)$	$O(n)$	$O(knd)$
dl	$O(ind)$	$O(n)$	$O(k^2nd)$
dch	$O(ind)$	$O(n)$	$O(k^2nd)$
dchperceptron	$O(ind)$	$O(n)$	$O(k^2nd)$
KSVD	$O(n(dK^2 + d^2K))$	-	$O(kn(dK^2 + d^2K))$

Table 1: Computational cost of each algorithm (per iteration and for k iterations), # calls to "distance to dictionary" function per iteration for each algorithm. n=#points, d=dimension of points, i=current dictionary size (for greedy algorithms), K=fixed dictionary size for KSVD, k=#iterations

6 Analysis

We employ the greedy framework described in section 5 to solve the sparse dictionary learning problem. Specifically, we use the dp, dl, dch and dchperceptron algorithms (with max and mean stopping rule) to learn a dictionary and sparse code for the five benchmark datasets. Further, we also use the K-SVD algorithm to solve the problem and compare its performance with the above mentioned algorithms. The metrics for comparison are dictionary size (k), avg. sparsity (s) and compression ratio (c). Note that since KSVD requires dictionary size as an input, we experiment with various values of k and report the one resulting in lowest avg. sparsity (s). For our problem with the data matrix $Y \in \mathbb{R}^{d\times n}$, dictionary $D \in \mathbb{R}^{d\times k}$, and representation matrix $X \in \mathbb{R}^{k\times n}$, their definitions are as follows:

- Avg. sparsity $(s) = \frac{1}{n}\sum_{x\in X} \|x\|_0$
- Compression ratio $(c) = \frac{dk+sn}{dn}$

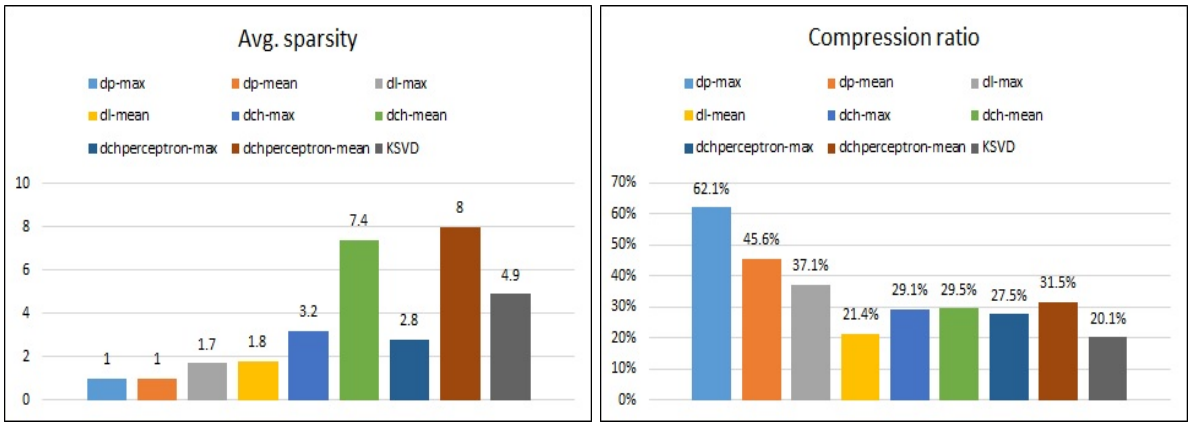
Avg. sparsity is thus the average L0-norm of the representation vector. The compression ratio considers the overall compactness of the representation by combining the dictionary size and sparseness of representation vector into a single value. We take the error tolerance ϵ (constraint in the optimization problem) to be the average distance between pairs of closest points. Our goal through this exercise is to compare the performance of the algorithms as well as to understand the intrinsic compressibility and sparsity of the datasets.

In addition to solving the sparse dictionary learning problem, we consider an application of the greedy framework to active learning based classification. Specifically, we use the greedy framework to learn the dictionary and then use the dictionary to classify the remaining points. For each point not in the dictionary we apply the kNN algorithm utilizing only the dictionary atoms as neighbours. The prediction is based on a weighted (by distance) majority voting over the k nearest neighbours. This scheme requires the labels of only the chosen dictionary atoms from the training data.

7 Results

The implementation of the greedy algorithms (along with KSVD) and the full set of code to reproduce the set of experiments described here is available at: `https://github.com/vjoshi345/dap`.

We apply the greedy algorithms and KSVD to solve the sparse dictionary learning problem on the five datasets. It is found that all the algorithms are able to learn a dictionary and sparse code such that the constraint of our optimization problem is satisfied. It is instructive to look at compression ratio and avg. sparsity averaged across datasets and algorithms. This helps us empirically explore the intrinsic compressibility and sparseness of a dataset (independent of algorithm used); and that of the algorithm (independent of the dataset). Figure 1 presents the metrics averaged across datasets and figure 2 across algorithms.



(a) Avg. sparsity for each algorithm

(b) Compression ratio for each algorithm

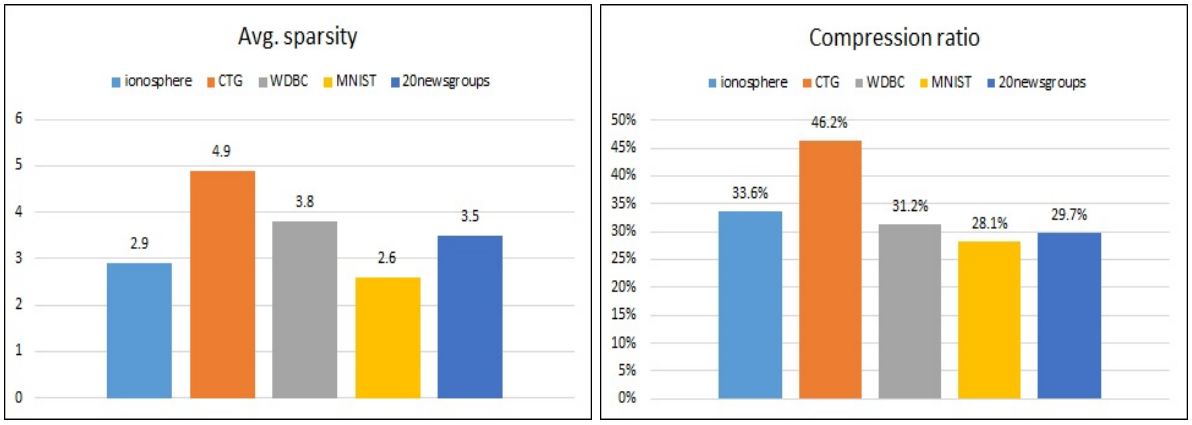
Figure 1: Avg. sparsity and compression ratio averaged across the datasets

A more detailed set of results are shown in table 2 and 3. Table 2 lists down the compression ratio for each algorithm and dataset pair. Table 3 shows the dictionary size learned by each algorithm and the average sparsity of the representation (average L0-norm of the representation vector). Note that KSVD requires the dictionary size as an input; hence the dictionary size leading to lowest average sparsity is reported here.

A sample of the dictionary atoms for MNIST data learned by dch-mean and KSVD algorithms is shown in figure 3. We can see that the dictionary learned by dch is interpretable (as compared to KSVD) since it consists of actual data points.

7.1 Classification results

We use the dictionary learning based active learning classification scheme on MNIST data. The dictionary is learned using the greedy algorithms and the rest of the training data is classified by weighted-by-distance kNN algorithm on the dictionary atoms. The learned dictionary size and classification error rate for all the algorithms is shown in Table 4. Table 4 also lists the randomized



(a) Avg. sparsity for each dataset

(b) Compression ratio for each dataset

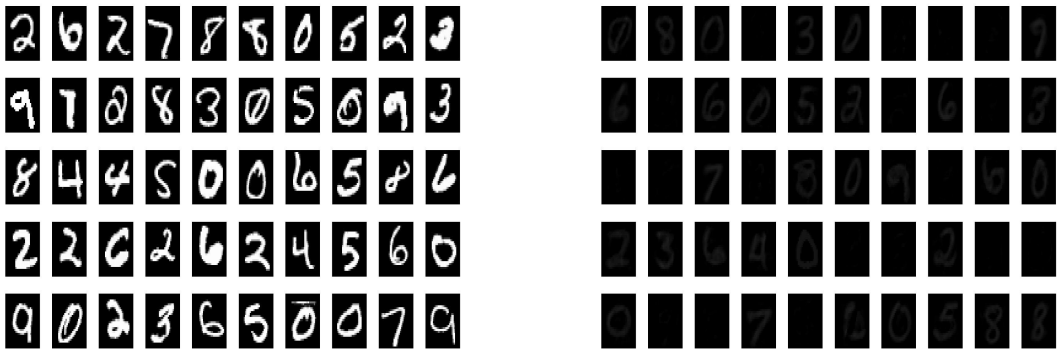
Figure 2: Avg. sparsity and compression ratio averaged across the algorithms

Data →	**iono**	**CTG**	**WDBC**	**MNIST**	**20news**
dp-max	48.8%	71.4%	61.2%	68.2%	60.7%
dl-max	39.1%	37.7%	24.7%	41.4%	42.6%
dch-max	34.2%	36.9%	22.5%	22.5%	29.4%
dchperceptron-max	33.7%	33.4%	20%	21.8%	28.7%
dp-mean	39.7%	50.2%	42.3%	48.8%	47.2%
dl-mean	26.1%	22.4%	13.1%	19.4%	25.7%
dch-mean	23.3%	72.8%	32.6%	10%	8.7%
dchperceptron-mean	28.6%	68%	42%	9.9%	8.9%
KSVD	29%	23.4%	22.3%	10.8%	15.3%

Table 2: Compression ratio for each algorithm and dataset pair

Data →	**iono**		**CTG**		**WDBC**		**MNIST**		**20news**	
Algorithm ↓	k	s	k	s	k	s	k	s	k	s
dp-max	161	1	1417	1	329	1	1361	1	908	1
dl-max	120	1.7	628	1.7	106	1.8	824	1.6	635	1.6
dch-max	95	2.4	349	4.3	61	3.5	443	2.9	433	2.8
dchperceptron-max	94	2.3	314	3.9	60	2.8	428	2.8	424	2.3
dp-mean	129	1	966	1	222	1	973	1	705	1
dl-mean	73	1.8	288	1.9	38	1.9	384	1.8	381	1.7
dch-mean	34	4.6	90	14.4	20	8.7	190	3.7	113	5.6
dchperceptron-mean	34	6.4	77	13.5	18	11.7	191	2.8	116	5.6
KSVD	50	5	250	2.4	100	1.4	200	6	200	9.7

Table 3: Dictionary size (k) and average sparsity (s) for each algorithm and dataset pair



(a) Dictionary atoms using dch-mean (b) Dictionary atoms using KSVD

Figure 3: Sample dictionary atoms for MNIST learned by (a) dch-mean and (b) KSVD

error rate which is the error rate when we apply the weighted-by-distance kNN algorithm on a dictionary made up of a random subset of the training data.

Algorithm	**Dictionary size**	**Error rate (%)**	**Randomized error rate (%)**
dp-max	1361	0.9	4.9
dl-max	824	2.9	6.3
dch-max	443	7.1	7.8
dchperceptron-max	428	6.1	8
dp-mean	973	4.2	6.3
dl-mean	384	8.9	9.3
dch-mean	190	19.4	12.9
dchperceptron-mean	191	19.6	12.9

Table 4: Dictionary size and classification error for MNIST data using dictionary learning based active learning classification scheme

7.2 Computational complexity

In table 5, we report the number of iterations required by the greedy algorithms and KSVD to solve the problem for each dataset.

8 Discussion

Figure 1 shows the compression ratio and avg. sparsity averaged across the datasets for an algorithm. Note that dp and dl are sparse by design since they reconstruct a point using at most one or

Data →	**iono**	**CTG**	**WDBC**	**MNIST**	**20news**
dp-max	161	1417	329	1361	908
dl-max	120	628	106	824	635
dch-max	95	349	61	443	433
dchperceptron-max	94	314	60	428	424
dp-mean	129	966	222	973	705
dl-mean	73	288	38	384	381
dch-mean	34	90	20	190	113
dchperceptron-mean	34	77	18	191	116
KSVD	3	17	5	3	3

Table 5: Number of iterations taken by each algorithm for each dataset

two atoms respectively. Hence, their low average sparsity values. dch-max and dchperceptron-max generate a sparser representation than KSVD on average. It is interesting to note that their mean counterparts perform worse on sparsity than KSVD. This is explained by the smaller dictionary size learned by the mean version which might mean the dictionary misses some important atoms and the reconstruction of some points requires larger number of atoms (poorer sparsity on average). This is seen consistently across datasets in table 3. At the same time the compression given by dch-mean and dchperceptron-mean is worse than their max versions. So, in this case the reduction in dictionary size is overcompensated by the increased avg. sparsity leading to a worse compression ratio. However, the situation is reversed for dp and dl. The compression given by mean versions is better than the max version. This time since the algorithms constrain the sparsity the reduction in dictionary size is able to provide better compression. In fact, dl-mean provides the best compression among the greedy algorithms on average and is comparable with KSVD. At a high level, greedy-max algorithms give better sparsity while overall compression given by KSVD is better.

From table 3, we can see that the dictionary size learned by dp is largest followed by dl and dch (or dchperceptron) consistently across datasets. This is a direct consequence of the definition of "distance to dictionary" used by these algorithms. For a query point, the distance to dictionary given by dch (or dchperceptron) is less than or equal to that given by dp. And for a dataset of sufficient size we would expect distance given by dl to lie somewhere between the two; its dictionary size lying between dp and dch is consistent with this observation. KSVD can solve the problem for different dictionary sizes for a dataset; here we report the dictionary size leading to lowest average sparsity (the objective of the optimization problem). Hence, we cannot compare the dictionary size of KSVD directly with the greedy algorithms.

From figure 2 we see that both MNIST and 20newsgroups are compressed well while CTG is seen to be least compressible. This may be because CTG contains features derived from cardiotocograms which might not be made up of a small set of building blocks. In contrast, MNIST might have a simpler structure being made up of 0-9 digits or 20newsgroups might have a natural clustering among topics.

Table 4 shows the error rates of classification under the dictionary learning based active learning classification scheme. The max version of the algorithms seem to perform well compared to a

dictionary selected randomly. dp and dl seem to work especially well. This might be because dp and dl reconstruct a point more directly using atoms while dch involves a convex combination of potentially many atoms (does not imply that the atoms themselves are closer to the point being reconstructed). It is interesting to note that the mean version of the algorithms perform significantly worse than the max versions. One possible explanation is that the greedy algorithms preferentially choose atoms from among points which are far away (near the boundary of the dataset). So, the extra atoms chosen by the max version (over the mean version) are from the "center" of the dataset and act as a better set of neighbours for a kNN classification scheme.

Table 5 lists the number of iterations required by the algorithms to solve the problem for each dataset. Note that for the greedy algorithms the number of iterations equals the dictionary size (since at each iteration we add a new point to the dictionary). Thus, the relationship (and reasons thereof) described between the relative dictionary sizes for the algorithms apply directly to the number of iterations. It is interesting to note that K-SVD requires significantly fewer iterations than the greedy algorithms. This observation suggests that KSVD makes much more progress per iteration than the greedy algorithms. This is also consistent with the quadratic computational cost on dictionary size and dimensions of points for KSVD versus linear for greedy algorithms (from table 1).

9 Conclusions

The greedy framework presented here is able to successfully solve the sparse dictionary learning problem. We found that the representation given by the greedy algorithms is sparser on average than KSVD; however, the overall compression given by KSVD is better. This is expected because KSVD allows for dictionary atoms to be linear combinations of data points and represents a point in terms of linear combinations of atoms. This results in a more compact representation. However, the advantage of the greedy framework is that the dictionary is interpretable. Further, the labels of dictionary atoms are available which could be used for supervised learning tasks. Another advantage of the greedy algorithms is that they can be kernelized since they require only dot product access to data points. Future work might include assessing the quality of the representation learned for various machine learning tasks or developing a framework for kernelization of the algorithms.

10 Acknowledgements

I would like to thank my advisor Avrim Blum for his valuable guidance throughout the project. Thank you to Roni Rosenfeld for being a part of my committee. I would also like to thank Roy Maxion for discussions about datasets and for taking the DAP Prep course which was useful in understanding how to write an experimental paper. Special thanks to Susrutha Gongalla for insightful discussions about the DAP and Machine Learning in general.

References

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

Blum, A., Har-Peled, S., and Raichel, B. (2015). Sparse approximation via generating point sets. *CoRR*, abs/1507.02574.

Chen, S. S., Donoho, D. L., and Saunders, M. A. (2001). Atomic decomposition by basis pursuit. *SIAM Review*, 43(1):129–159.

Engan, K., Aase, S. O., and Husoy, J. H. (1999). Method of optimal directions for frame design. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, volume 5, pages 2443–2446 vol.5.

Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2009). Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 689–696, New York, NY, USA. ACM.

Mallat, S. G. and Zhang, Z. (1993). Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415.

Olshausen, B. A. and Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, 37(23):3311 – 3325.

Pati, Y. C., Rezaiifar, R., and Krishnaprasad, P. S. (1993). Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, pages 40–44 vol.1.

Rubinstein, R., Zibulevsky, M., and Elad, M. (2008). Efficient implementation of the k-svd algorithm using batch orthogonal matching pursuit. *Cs Technion*, 40(8):1–15.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288.

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

Tropp, J. (2004). *Topics in Sparse Approximation*. PhD thesis, Univ. of Texas at Austin.