Masaryk University
Faculty of Informatics

Dimensionality reduction methods for vector spaces

Master's Thesis

Bc. Jan Brázdil

Brno, Spring 2016

Replace this page with a copy of the official signed thesis assignment and the copy of the Statement of an Author.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Jan Brázdil

Advisor: prof. Ing. Pavel Zezula, CSc.

Acknowledgement

I would like to thank Professor Zezula for helpful consultations and advice. I would also like to thank members of DISA laboratory for useful remarks and cooperation. Also I would like to thank my parents and my roommates for their support and for letting me work on this thesis in peace.

Abstract

This thesis tries to approach the problem of large data in image similarity search by using a dimensionality reduction method. In this thesis the principal component analysis is evaluated on a large data set of DeCAF image descriptors. It was found that the principal component analysis can be used to reduce the size of the data up to one hundred times while keeping high precision by using a candidate set refiment.

Keywords

Dimensionality Reduction, Similarity Search, Principal Component Analysis, DeCAF

Contents

1 Introduction

This thesis tries to approach the problem of large data in image similarity search. The similarity search is an important field in the modern information age, namely in domains like multimedia, molecular biology, marketing and many others.

This thesis was made in the cooperation with laboratory DISA of Masaryk University. The laboratory works, among other things, on the image similarity search using state-of-the-art image descriptors. The challenge is a large amount of data that needs to be processed and worked with. In this thesis I investigate the option of using dimensionality reduction methods to reduce the size of the data while retaining as many of the good qualities the data have.

1.1 Structure of this thesis

This thesis is divided into six chapters.

In this *Chapter 1*, there is introduction to this thesis, its goals and structure.

In *Chapter 2* dimensionality reduction is introduced and several dimensionality reduction techniques are presented.

Chapter 3 introduces the problematics of information retrieval and specifically image similarity search. This chapter also describes several metrics for evaluation of an information retrieval system.

Chapter 4 presents the implementation of selected dimensionality reduction method and describes the data set used in the evaluation of the method.

In *Chapter 5* the results of the evaluation are shown. Also, the suitability for an application of the selected method is discussed.

In the last *Chapter 6*, the results of this thesis are summarized.

2 Dimensionality reduction

In machine learning, information retrieval, statistics and other fields where data is analyzed it is important to have a lot of good data. Advancements in data collection over the past years resulted in bigger data then before. This means that both the number of data objects and the number or dimension of the data objects has grown. However, as stated in [1], having bigger data usually comes with higher noise. Also, having more measured variables (*features*) does not guarantee that all of them are important.

Big dimensionality of data can cause problems, often referred to as *curse of dimensionality*. Some methods can make use of high-dimensional data, but many benefits from *dimensionality reduction* [2]. Reducing the number of the data features can help improve learning performance, create better generalizable models, lower computational complexity, decrease required storage and help visualize the data. Dimensionality reduction removes noisy and redundant features by obtaining key low-dimensional uncorrelated features. A good source on dimensionality reduction is [3].

Dimensionality reduction can be divided into two approaches: *feature selection* and *feature extraction*. Feature selection focuses on selecting a small subset of features. Feature extraction projects features into new lower dimensional feature space. Another distinction of dimensionality reduction methods is to *linear* and *nonlinear*. Linear methods do a linear combination of the original features, nonlinear methods use nonlinear transformation of the data.

2.1 Curse of dimensionality

Curse of dimensionality covers several problems that occur when we deal with a high-dimensional data. The term was first introduced by Richard Bellman in [4]. One of the problems is that having training data set of fixed size, increasing the number of features increases performance of a data processing system only to some point. After that point, increasing the number of features decreases the performance. This is caused by the fact that increasing the dimension makes the

data sparse. In sparse data, vector densities are not represented well and this have bad effect on the system [5].

Also, there are other problems that comes with high dimensionality. Bigger space is required for storing and handling the data. More computational resources are needed for processing the data. Greater amount of training data is needed to train the system properly.

2.2 Feature selection

Feature selection tries to find small subset of the original features. It uses a relevance evaluation mechanism to select such features that minimize redundancy and maximize relevance. Because Feature selection preserves physical meaning of the original features the readability and interpretability of selected features is better than in Feature extraction.

The simplest feature extraction strategy would be to select all the possible subsets of features, use the relevance evaluation mechanism to evaluate each of the subsets and select the best performing subset. However, because number of subsets grows exponentially with dimension this strategy would not be feasible on high-dimensional data. More sophisticated methods are necessary to efficiently perform feature selection on high-dimensional data.

According to the used data set, feature extraction methods can be divided into *supervised*, *unsupervised* and *semi-supervised*. Supervised feature extraction methods can be used when the data are labeled. They can be divided further into *filter models*, *wrapper models* and *embedded models*. Unsupervised feature extraction uses unlabeled data set and semi-supervised methods can be used on data sets where only part of the data is labeled.

Filter model uses measures of general characteristics to select the features. The measures are applied to the features or feature subsets to asses their importance. Common characteristic include distance, consistency, mutual information, dependency or correlation. The filter model is fast computing; however, it is not performing as well as wrapper model.

Wrapper model uses a predictive model to score each subset. For each subset, the model is trained and evaluated, giving the subset

a score. The best scoring subset is then selected. The wrapper model is computationally intensive, but gives good feature set.

Embedded model combines both approaches to achieve both fast computability and good performing feature set. The filter model is used to provide candidate subsets and the wrapper model is used to score the subsets.

Examples of feature selection methods include Information Gain [6] or Relief [7]. A review of feature selection is in [1].

2.3 Feature extraction

Feature extraction transforms features from the original feature space into a new low-dimensional feature space. The new features are usually combinations of the original features but it may be difficult to link the new features to meaning of the original features. On the other hand it can extract features with a hidden meaning.

Feature extraction is usually divided into *linear* and *nonlinear* methods. Linear method does a linear combination of the original features. The prominent linear feature extraction method is *principal component analysis*. It does linear mapping of the original data to a low-dimensional space in such a way that variance of the new features is maximized [8]. Other examples of linear mapping include *linear discriminant analysis* or *singular value decomposition*. Nonlinear method uses a nonlinear transformation of the data. Common approach is to assume that the data lies on a non-linear manifold within the high-dimensional space. Some examples of nonlinear methods are *kernel PCA*, *Isomap* or *locally linear embedding*.

2.4 Linear dimensionality reduction methods

Linear methods do a linear combination of the original features. Becase principal component analysis and singular value decomposition make use of *eigenpairs*, they are explained first. After that follows a presentation of several linear dimensionality reduction methods.

2.4.1 Eigenpair

When we multiply a vector by a transformation matrix, almost all vectors change direction. The vectors that do not change direction when the transformation is applied to them are called eigenvector. Eigenvalue then tells what happens to the vector length when it is transformed.

As defined in [8], for square matrix M there is an eigenvector e and corresponding eigenvalue λ considering following: λ is a constant and e is a nonzero column vector with the same number of rows and M. Multiplication of the matrix M by the eigenvector equals multiplication of the eigenvector by its eigenvalue:

$$Me = \lambda e \tag{2.1}$$

Because the multiplication of eigenvector of M by constant c is also an eigenvector of M with the same eigenvalue, we require that every eigenvector is unit vector (the sum of the squares of the components of the vector is 1) to avoid ambiguity. Also, because multiplication of unit vector by -1 does not change the sum of squares of the components, we also require that first nonzero component of an eigenvector is positive.

Each square matrix have several eigenpairs with non-zero eigenvalue, but the maximum is equal to the rank of the matrix. The eigenvectors and eigenvalues are often used in dimensionality reduction.

2.4.2 Principal Component Analysis

Principal component analysis, described in [8], is a linear feature extraction technique that finds directions in a data along which the variance of the data points is maximized. These directions are called *principal components*, they represent linearly uncorrelated features in the data. First principal component has the largest possible variance and each consecutive principal component has the largest possible variance while being orthogonal to previous components. An example of PCA on 2-dimensional data can be seen in figure 2.1.

To compute PCA the data set is treated as matrix M. Then from the matrix M it is computed MM^T or M^TM. The multiplication which produces smaller matrix can be chosen. From the MM^T or M^TM all

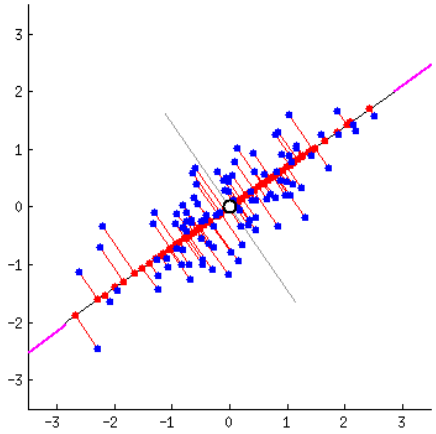

Figure 2.1: Illustration of PCA. First principal component is represented by line with magenta ends. Red points are projection of the data onto the first principal component. Taken from [9].

the eigenvectors are computed. The matrix of these eigenvectors is used as a transformation matrix to transform original data. The axis of the first eigenvector (corresponding to the largest eigenvalue) is the one along which the variance of the data is maximized. The axis of the second eigenvector is orthogonal to the first one and the variance of distances from the first axis is the greatest along it. Similarly, for the remaining axes.

The principal component analysis can be used for dimensionality reduction by selecting the most important eigenvectors (corresponding to the largest eigenvalues). The selected eigenvectors then create transformation matrix that transforms the original data into a low-dimension space.

To estimate how much information is preserved by the PCA dimensionality reduction, the *retained energy* can be computed. The *total energy* of PCA is defined as a sum of eigenvalues from the eigenmatrix of the PCA. The retained energy can be computed by summing the eigenvalues of the selected eigenvectors and dividing the sum by the total energy.

2.4.3 Singular Value Decomposition

Singular value decomposition computes representation of an original matrix M by three matrices U, Σ and V. The matrices U, V are column-orthonormal (columns are unit vectors and each two columns are orthogonal) and Σ is diagonal matrix consisting of the singular values (square root of eigenvalues) of M.

$$M = U\Sigma V^T \tag{2.2}$$

SVD is useful when there is a small number of concepts that connect the rows and columns of the original matrix. SVD can be used for dimensionality reduction by removing columns corresponding to the smallest singular values from U, V and Σ. More information on the SVD can be found in [8] along with the CUR decomposition.

2.4.4 CUR Decomposition

In singular value decomposition, even when the original matrix M is sparse, the matrices constructed by SVD are dense. CUR decomposition seeks to decompose a sparse matrix into sparse, smaller matrices C, U and R whose product approximates the original matrix.

$$M \approx CUR \tag{2.3}$$

CUR decomposition chooses a set of columns C and set of rows R from matrix M. These sets resemble U and V^T in SVD. The columns for C and rows for R are chosen randomly with a distribution that depends on the square root of the sum of the squares of the elements in the column or row. Middle matrix U is then constructed by a pseudo-inverse of the intersection of the chosen rows and columns. Dimensionality reduction can be done by CUR decomposition by selecting lower amount of rows and columns.

2.4.5 Linear Discriminant Analysis

Linear discriminant analysis (LDA) [10] is a dimensionality reduction technique used in classification. It tries to reduce dimensionality while preserving class discriminatory information. LDA seeks lines in the feature space on which when the data is projected the class

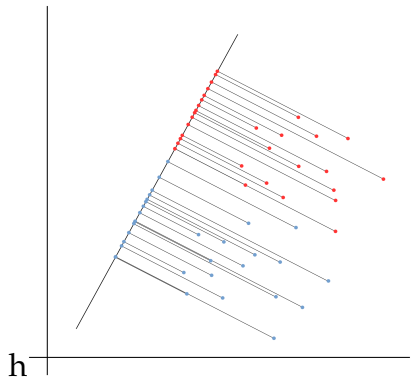

Figure 2.2: Illustration of linear discriminant analysis

separability is maximized. This is illustrated in figure 2.2. For K-class problem LDA projects the data from original feature space to lower dimensional space using $K - 1$ different lines. The lines are oriented as to maximize the separability of the classes.

2.5 Nonlinear dimensionality reduction methods

2.5.1 Kernel PCA

Kernel principal component analysis was introduced in [11]. It is a generalization of linear PCA into nonlinear case using the kernel method. First, kernel PCA maps original input vectors into a high-dimensional feature space using non-linear mapping, and then calculates the linear PCA of it. The difference of linear PCA and kernel PCA is illustrated in figure 2.3.

2.5.2 Isomap

Isomap, presented in [12], is a manifold-learning algorithm that learns the internal model of the data. It connects nearest neighbours forming a graph connecting all the data points. Next it computes the shortest path between all the nodes in the graph. This approximate the geodesic distance (distance of two points with respect to a surface) of the points. Finally, it applies multidimensional scaling on the matrix of graph distances yielding a low-dimensional embedding of the orig-

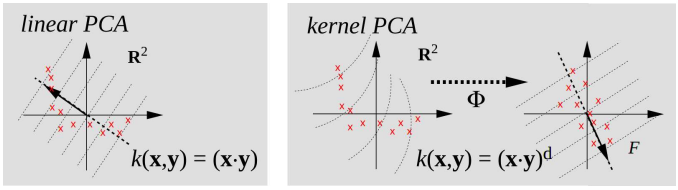

Figure 2.3: *"Basic idea of kernel PCA: by using a nonlinear kernel function k instead of the standard dot product, we implicitly perform PCA in a possibly high-dimensional F space which is nonlinearly related to input space. The dotted lines are contour lines of constant feature value.* [11]"

inal data. Example of the Isomap finding the distance of two points in Swiss-roll data set is in figure 2.4.

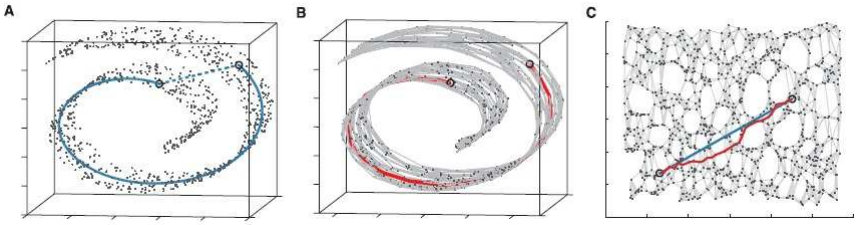

Figure 2.4: Illustration of Isomap [13]

2.5.3 Locally Linear Embedding

Locally linear embedding (LLE), similarly to Isomap, is an unsupervised algorithm that computes low-dimensional embedding that preserve neighborhood of the original high-dimensional data. For each point in the data set LLE finds a set of the nearest neighbours. Then it represents each point as a linear combination of its neighbours. The linear combination gives weight for each of the neighbor. Finally, it finds low-dimensional embedding using the weights that approximate the original data well. Locally linear embedding was presented in [14].

3 Similarity search and retrieval evaluation

In this thesis I focus on using dimensionality reduction in information retrieval, specifically in image similarity search. This chapter briefly introduces information retrieval in general and then focuses on image similarity search.

In this chapter I also mention several evaluation techniques that can be used to measure how well the information retrieval system meets the information needs of a user.

3.1 Information retrieval

For a long time people have collected and organized information for further retrieval and searching. Starting with first libraries the volume of information is growing and so the techniques to effective and efficient retrieval needed to improve too. With introduction of World Wide Web information retrieval was no longer area just for librarians and information experts. Millions of users now create and search enormous numbers of documents (or generally *objects*) in the largest human repository of knowledge in history.

Information retrieval can be viewed from two angles: a human-centered and a computer-centered. The human-centered focus studies the behavior of the user, understanding the needs of the user and determining how these understandings affect the retrieval system. The computer-centered view studies building of indexes, developing ranking algorithms and processing user queries. Because of the sheer amount of information available, the usefulness of information depends on quality and speed at which it can be retrieved. The information retrieval problem states: "*the primary goal of an information retrieval system is to retrieve all the objects that are relevant to a user query while retrieving as few irrelevant objects as possible.* [15]" The secondary goal is to retrieve the information fast.

Important task of information retrieval is to measure relevance. Because it is impossible to measure relevance for each query by hand, there is a need for an algorithm which determine the relevance of objects. Also, because users mostly look only at first few objects retrieved, the most relevant objects should be at the top of the result. By

ranking retrieved objects by relevance, we can order the result. And to allow fast searching of information, the information must be organized. The organization can be done by indexing - preprocessing the information so they are easily searchable. An excellent resource on information retrieval is [15].

One of the area in information retrieval is similarity search where the only available comparison is by the similarity between pair of objects. Similarity search is applicable for collections of sophisticated objects like scientific data records, biochemical and medical data, product catalogs with preference-based search or various multimedia objects – sounds, video or images. Similarity search in metric spaces is described in [16].

3.2 Image similarity search

When searching in large databases of complex data like image databases, object collections, time series databases or genome databases the data objects are seldom indexable by simple attribute data. There is usually no natural ordering of the data so sorting is impossible. But often the objects can be compared by similarity to each other.

Similarity search in metric space allows construction of indexes and is general enough to most search problems. The metric space needs to meet four requirements: non-negativity, symmetry, identity and triangle inequality.

The primary objective of similarity search is for a given query retrieve ranked subset from available data collections. The most common types of queries are the *k-nearest neighbours query* and *range query*. Similarity search can order the results for the query by the distance of the retrieved objects to the query.

This thesis focuses on image search, but methods described here are adaptable to other sophisticated objects like mind maps, sounds, video or movements.

When dealing with images, we can not just compare two images and get a similarity measure. This is neither practical nor easy. Instead, we use a data reduction techniques to extract interesting features (*descriptors*) that capture the application-driven characteristics of the image data. These descriptors are usually high-dimensional

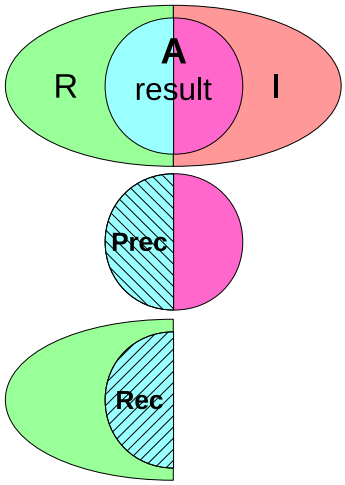

Figure 3.1: Illustration of precision and recall. *R* are relevant and *I* are irrelevant objects. *Prec*ision is part of retrieved objects that is relevant. *Rec*all is part of relevant objects that is retrieved.

vectors or other objects which allow just pairwise distances measurement. An example of such descriptors can be MPEG-7 descriptors or DeCAF descriptors.

In order to be successful, the similarity search needs to be effective in image processing to achieve high quality of retrieval and efficient in search to make the system work in real time on a large scale.

3.3 Retrieval evaluation

To measure performance of an information retrieval system or to compare its quality with another IR system, systematic evaluation measures are needed. The most widely used techniques *precision* and *recall* are introduced. Then *discounted cumulated gain* and *Spearman's coefficient* are presented. More details and other measures are detailed in [15].

3.3.1 Precision, Recall

Precision and *recall* are major evaluation metrics in information retrieval. Precision denotes the fraction of the relevant objects *R* in the result to all the objects in the result *A*. Having high precision means that most of the retrieved results were relevant and only a few irrele-

vant.

$$Precision = \frac{|R \cap A|}{|A|} \tag{3.1}$$

Recall is the fraction of the relevant objects that were retrieved. Having high recall means that most of the relevant results were retrieved.

$$Recall = \frac{|R \cap A|}{|R|} \tag{3.2}$$

The precision and recall are illustrated in figure 3.1.

3.3.2 P@5, P@10, P@k

When a web search is considered, high recall is not usually required, but the number of relevant results at the top of the ranking is important. Precision can be measured when a certain number of first objects was retrieved. Precision at k (or P@k) measure precision of first k results. Most commonly used ks are 10 for the usual number of results in a web search and 5 for the first results a user see.

3.3.3 Discounted Cumulated Gain

Precision and recall work only with binary relevance. If we want to distinguish between different levels of relevance, we need to use measures that takes into account relevance level. One of these measures is discounted cumulated gain. It works with graded relevance and considers that highly relevant objects should be at the top of the ranking and that relevant objects at the end of the ranking are less useful.

Assuming we have graded each object d with score $G(d)$. We can compute *cumulated gain* $CG(i)$ at each position i of a ranking, where d_i is object at the position i.

$$CG(i) = \begin{cases} G(d_1) & \text{if } i = 1 \\ G(d_i) + CG(i-1) & \text{otherwise} \end{cases} \tag{3.3}$$

We also consider discount factor that penalizes the objects that occur later in the ranking. One simple discount factor is log_2. Using the discount factor gives us discounted cumulated gain $DCG(i)$:

$$DCG(i) = \begin{cases} G(d_1) & \text{if } i = 1 \\ \frac{G(d_i)}{log_2 i} + DCG(i-1) & \text{otherwise} \end{cases} \tag{3.4}$$

Discounted cumulated gain can be hard to read alone, so *ideal discounted cumulated gain* can be used as a baseline. Ideal gain is when all the results are sorted by their relevance score. Finally, the ideal discounted cumulated gain $IDCG(i)$ can be used to compute *normalized discounted gain* $NDCG(i)$:

$$NDCG(i) = \frac{DCG(i)}{IDCG(i)} \tag{3.5}$$

We can average multiple measures of normalized discounted gain and plot the average on a graph to compare the rankings.

3.3.4 Spearman's Coefficient

When we are comparing two ranking functions and we want to compare the relative ordering, we can use Spearman's coefficient. Spearman's coefficient is often used rank correlation metric. It is based on the differences between the positions of the same object in two rankings. It determines how well the relationship between two variables can be described using monotonic function.

Let $s_{1,j}$ and $s_{2,j}$ be positions of a object d_j in two rankings R_1 and R_2, the Spearman's coefficient then is:

$$S(R_1, R_2) = 1 - \frac{6 \cdot \sum_{j=1}^{K} (s_{1,j} - s_{2,j})^2}{K \cdot (K^2 - 1)} \tag{3.6}$$

The $S(R_1, R_2)$ is in the range $-1 \leq S(R_1, R_2) \leq 1$. Coefficient 1 means the two rankings are completely correlated, coefficient -1 means they are reverse of each other and 0 means the two rankings are completely independent.

4 Data set and implementation

In this chapter the data used in the evaluation part of this thesis are described here and it is explained how they were obtained using the state-of-the-art DeCAF image descriptors. The implementation of selected dimensionality reduction method PCA and implementation of related tools and experimental program are described.

4.1 Data set

Since this thesis was done in cooperation with DISA Laboratory of Masaryk University[1], the data set used in evaluation part of this thesis was provided by the laboratory. The data set consists of 20 million DeCAF descriptors of images from Profiset image collection.

4.1.1 DeCAF descriptors

DeCAF descriptors, introduced in [17], are state-of-the-art image descriptors obtained using the pre-trained deep convolutional neural network. Convolutional neural networks have been used in machine learning and computer vision for a long time and in the recent years they have been able to outperform hand-engineered methods. However, deep models need large amount of training data because on small training data they tend to overfit. Nevertheless, it was shown that a well trained model can be reused for different tasks than it was trained for. The DeCAF descriptors are an example of this transfer learning.

To obtain DeCAF descriptors, a deep convolutional model had to be trained first. The used neural network model was proposed by Krizhevsky et al. in [18]. The model consist of eight learning layers: the first five layers are convolutional and the last three layers are fully connected. The input layer is a color image of size 224×224 pixels. The output layer has 1000 neurons for classification into 1000 classes. The model was trained using supervised training.

Because the training and real world images can be in various sizes, each image is resized to 256×256 pixels ignoring the original aspect

1. Laboratory of Data Intensive Systems and Applications disa.fi.muni.cz

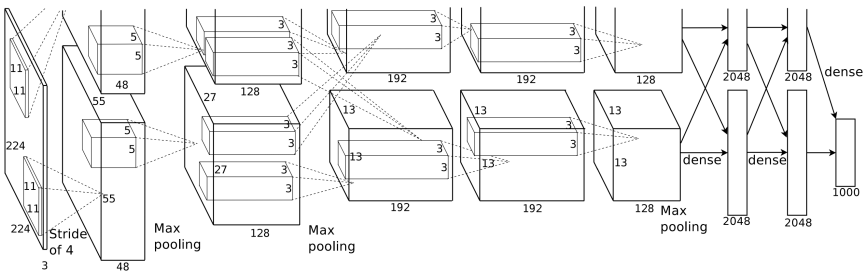

Figure 4.1: Illustration of architecture of the neural network [18]

ratio. Then the center 224×224 pixels are cropped and used as input to the neural network.

Three features were extracted from the model and tested. The features DeCAF_7 representing activations of the final hidden layer in the model, DeCAF_6 representing the layer before DeCAF_7 and DeCAF_5 representing the layer before DeCAF_6. The DeCAF_6, consisting of 4096 float numbers, showed the best overall performance. The DeCAF features were shown to be able to cluster semantically similar images of categories on which the network was never trained or distinguish different subcategories (e.g. bird species).

The trained network parameters were made publicly available together with *decaf* – an implementation of neural network framework. Decaf was later superseded by the *Caffe* framework[2].

4.1.2 Profiset image collection

Profiset image collection, presented in [19], is collection of 20 million high-quality images with rich and systematic annotations. The images were obtained from Profimedia[3]. The image databases consists of a thumbnail image, a link to the image on the Profimedia website, a title, keywords and 5 MPEG-7 visual descriptors.

This collection of images was processed by the Caffe framework, producing image descriptor for each of the images. Each of these descriptors is a 4096 dimensional float vector. This results to over 300

2. Caffe | Deep Learning Framework caffe.berkeleyvision.org
3. Photos, news, video - photobank Profimedia www.profimedia.com

GiB of raw data. Collection of these descriptors was used in the evaluation part of this thesis.

4.1.3 Description of the data set

The data set provided by DISA laboratory consist of 20 million image descriptors. Because of limitation of computational resources, only first 1 million descriptors were used. The data were provided in *MESSIF* text format (see figure 4.2) compressed by *gzip*. Each image is identified by a key and described by 4096-dimensional float vector. The vectors are sparse, in average *62.75%* elements in a vector are zero. The distance of two objects is computed as a Euclidean distance of their vectors.

```
#objectKey messif.objects.keys.AbstractObjectKey 0000000006
0.0340718 0.0 0.0 0.0 0.0 0.0 0.766039 0.139893 0.0 0.0
```

Figure 4.2: Example of the MESSIF data text format. The example data is 10-dimensional float vector with a key "0000000006".

4.1.4 Sparse vector compression

Because the data set of DeCAF descriptors is sparse it can be considerably well compressed by a sparse vector compression. One such compression uses bitmap to mark position of non-zero elements and float vector to store the values of the non-zero elements. In this case the size of the bitmap is 4096 bits represented by array of 64 8-byte words of type long. The non-zero elements are stored in an array of 4-byte floats. Given the average vector contains *62.75%* zeros the theoretical compression ratio is around *2.476*, or in other words, the resulting size is around *40.38%* of the original size.

The precision of the compressed vector is unchanged, since the original vector can be retrieved. The performance characteristics are evaluated later in section 5.5. When the focus is on reducing the size of a data set, the compressibility can work as a threshold. This threshold

tell us that reductions to some dimensions are pointless. In this case, we sometimes omit the evaluation of reduction to more than *40.38%* of the dimensionality (more than 1654 dimensions).

4.2 Choice of dimensionality reduction method

Because of the nature of the data and the main interest in reducing the physical size of the data, Principal Component Analysis was selected as a candidate dimensionality reduction technique to be evaluated. Since the data were obtained from hidden layer of a neural network, individual features seemed to be of very similar importance. This was the reason to not use feature selection techniques but focus on feature extraction. Because the task of similarity search does not involve classification, only techniques that works with unlabeled data were considered. To allow fast computation and easy expandibility of the data, linear techniques were targeted. Principal Component Analysis, being the prominent linear feature extraction technique, was therefore selected.

4.3 Implementation

To perform dimensionality reduction using PCA and to evaluate possible configurations, I developed a set of tools and experimental programs. I have chosen to program them in Java programming language because I am already proficient with it and there exists similarity search framework MESSIF implemented in Java.

The work was done with the latest version of Java: Java 8. New features of Java 8 – streams and lambda expressions – were used widely. Parallel streams were used to improve performance on multi-core processors. Functional interfaces were used to create generic implementations.

Two Java libraries were used. First is MESSIF – Metric Similarity Search Implementation Framework. It was developed in DISA laboratory and introduced in [20]. MESSIF is a framework that helps with building prototypes or applications for similarity search in metric space. It is a modular system enabling easy extensibility. It has a support for indexing of metric spaces with several indexing algorithms

already implemented. It allows using distributed or centralized data structures. And it also provides automatic measurement of numerous statistics. MESSIF library is open-source and licensed under the GNU GPL v3 license.

The second library used is JAMA[4], a library for basic linear algebra. It provides implementation for matrices and operation on matrices like multiplication or computation of eigenpairs. The JAMA library is open-source and has been released to the public domain.

To build the project Maven build system was used. It manages the whole build process: downloading dependencies, compiling sources and packaging the application. Maven provides central repository with many projects that can be used as a library. The JAMA library is located in the repository; however, MESSIF framework is not. Nevertheless, MESSIF framework supports Maven and can be downloaded from source code management system[5] and build locally.

4.4 Tools for principal component analysis

I developed several tools executable from the CLI. To perform a PCA dimensionality reduction of a data file, the `EPCompute` is first used to compute eigenpairs of the data. Next `DataTransform` uses the eigenpairs to transform the data and stores it as a transformed data file. Finally, first k columns of the transformed data file is taken as a reduction to k-dimensional space.

4.4.1 Eigen pair

```
EPCompute DATA [EIGENPAIRS]
```

To compute eigenpairs the `EPCompute` class is used. It computes all the eigenpairs for data in given data file (`DATA`) and stores them in a serialized object (`EIGENPAIRS`). The name of the serialized object file is optional, when it is not given the file name of the data with suffix `.ep` is used.

In order to compute eigenpairs a square matrix is needed. It is assumed that the number of the data rows is larger than the num-

4. JAMA: Java Matrix Package math.nist.gov/javanumerics/jama
5. disalab / MESSIF – Bitbucket bitbucket.org/disalab/messif

ber of the columns. To get the square matrix from the input data M, M^TM is computed by class `FastStreamSquare`. Because loading whole data set into memory is not possible for large data sets, the `FastStreamSquare` class works in batches. It starts with zero matrix R and iteratively loads batch of rows N and updates the matrix R:

$$R_{n+1} = R_n + N^TN \tag{4.1}$$

To compute the eigenpairs of the square matrix the JAMA library is used. The eigen pairs are then stored in an object of `EigenMatrix` class (figure 4.3). The object is then serialized and saved to a file.

Figure 4.3: `EigenMatrix` contains eigenpairs and eigenvalues.

4.4.2 Eigen pair info

```
EPInfo EIGENPAIRS
```

`EPInfo` is a small class that outputs information about a serialized `EigenMatrix` object (`EIGENPAIRS`). It outputs size of the eigen matrix, total energy of the eigenmatrix (sum of the eigen values) and tab separated list of energies for each column in percentage of the total energy.

4.4.3 Data Transform

```
DataTransform EIGENPAIRS DATA [OUTPUT]
```

`DataTransform` is a class that transforms data using an eigen matrix. As arguments it takes a serialized `EigenMatrix` object (`EIGENPAIRS`), a data file name (`DATA`) and optionally an output data file name (`OUTPUT`). It loads the eigen matrix and use the entire matrix as a transformation

matrix. It then reads the data file and use the transformation matrix to transform the data. Transformed data are written to the output file. If the output file name is not specified, the data are stored to file that has the same name as input file but with `.transformed` suffix.

There is no need to select target dimension. The data are transformed to dimension of the eigenmatrix and when lower dimension k is desired, the first k columns of the transformed data can be taken. However, if a need to select the dimension would arise (e.g. because the target dimension is known beforehand and the data is so large that the full transformation could not fit on available disk space) the `DataTransform` class could be easily adjusted. The advantage of this implementation is noticeable in evaluation. When evaluating reduction to several dimensions, the transformation can be done only once and the different dimensions can be extracted from the transformed data file just by selecting adequate number of the first columns.

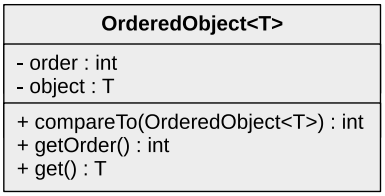

Figure 4.4: `OrderedObject` contains the data object and order number.

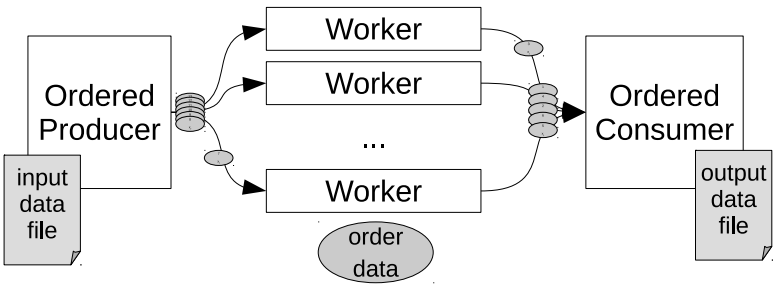

Figure 4.5: Illustration of the ordered Producer–Consumer workflow.

To utilize parallel processing the `DataTransform` class uses an ordered *Producer–Consumer* workflow. Class `OrderedProducer` use the Java `Supplier` functional interface to supply data objects, wraps them into the `OrderedObject` (figure 4.4) and put the wrapped object into a shared input queue. The objects are taken from the input queue by several `Workers`. The `Worker` class perform a function on the data ob-

ject and re-wraps the resulting objects into `OrderedObject` again with the same order number. Then it puts it into an output queue. From the output queue the objects are taken by the `OrderedConsumer`. The `OrderedConsumer` class processes the objects in the order given by the order number in `OrderedConsumer` and use the Java `Consumer` to process the data. The `DataTransform` class use the `OrderedProducer` to read data from input file, the `Workers` to transform the data and the `OrderedConsumer` to output the transformed data to output file. The process is illustrated in figure 4.5.

4.5 Implementation of experiments

To evaluate the reduced data an experimental program was implemented. It is named simply `Experiment`. It executes similarity search on the evaluated data and computes measures and statistics.

4.5.1 Experiment

```
Experiment ORIGINAL_DATA TEST_DATA [LOGFILE]
```

This program executes similarity search on the test file (`TEST_DATA`) and compares the result with results of a search on the original data file (`ORIGINAL_DATA`). User can optionally provide a log file (`LOGFILE`) where identifiers of all the queries and found neighbours are stored for further analysis.

The `Experiment` class is just a wrapper providing basic CLI interface for `Experimenter` class. The `Experimenter` class provides methods for running the similarity search and evaluation (see figure 4.6).

Constructor of the `Experimenter` takes file names of the original data file and test data file. The test file created by the `DataTransform` tool contains all the dimensions of original file, but transformation to a lower dimensions can be done just by selecting first few columns. The `Experimenter` can run the evaluation on several dimensions. The dimensions can be specified using the `addSizes` method.

The experiment can be run either by `runMemoryExpr` method or by `runSequentialExpr` method. Both of them takes number of queries to execute and number of the nearest neighbours to find. Each uses a `NeighbourFinder` to find the neighbours, the first uses in-memory

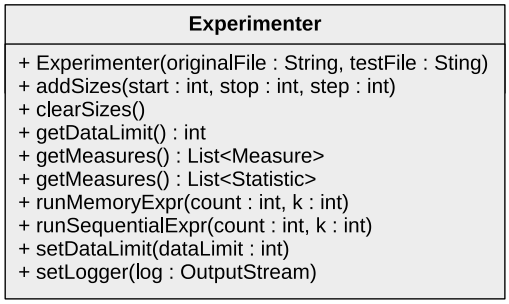

Figure 4.6: Public API of `Experimenter` class.

implementation, the second uses sequential scan implementation. Result of the queries are then evaluated using `Statistics` and `Measures`, single-value and multi-value metrics.

4.5.2 Neighbour finders

To find the nearest neighbours, an implementation of the interface `NeighbourFinder` is used. A neighbour finder works with both the original and a test data set. It selects query objects from the original data and finds k nearest neighbours. `Experimenter` can then select target dimension. Then it uses the neighbour finder to reduce the testing data to that dimension and find neighbours in the reduced data. The neighbour finder returns two lists of neighbours for each query: the k nearest neighbours from the original data set and list of so many nearest neighbours from the test data set so all the k original objects are among them. This is achieved by finding all the original objects by their identifier in the test data set and for each query computing the maximal distance to these objects. Then a range query is performed using the computed distance. This process is illustrated in figure 4.7.

Two implementations of the interface `NeighbourFinder` were created. One uses in-memory indexing algorithm, second uses sequential scan algorithm. `MemoryNeighbourFinder` is using *vantage point tree* (VPT) algorithm [16]. It loads the data and stores them in the tree. When the dimension of the test data is changed the tree is discarded and reloaded with the new dimension. `SequentialNeighbourFinder` is using sequential scan of the data file. This algorithm has advantage of low memory consumption so it can be used for large data sets.

4.5.3 Measures and statistics

Evaluation of the test data can be done using several metrics. Single-value metrics implements `Statistic` interface, multi-value metrics implement `Measure` interface.

Both measures and statistics takes two list of identifiers. First represents neighbours found in the original data set, the second represents neighbours found in the test data set. Several statistics and measures were implemented, some of them include:

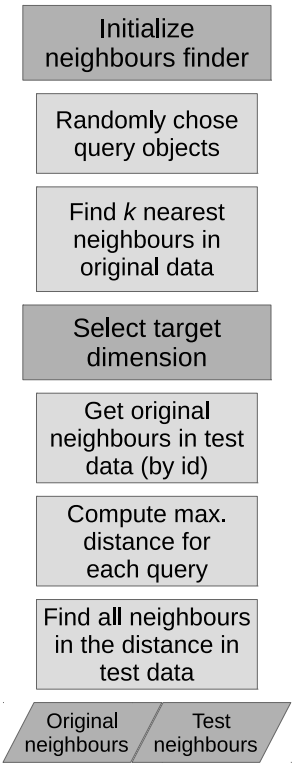

Figure 4.7: Process of `NeighbourFinder`.

Precision at k

Precision at k ($P@k$) statistics is implemented by class `PrecisionAtK`. Given number k this statistics selects set of the first k neighbours from both original neighbours O and test neighbours T. Then it makes an intersection of these sets and divides the intersection size by k.

$$P@k = \frac{|O \cap T|}{k} \tag{4.2}$$

Spearman's Coefficient

Spearman's coefficient is implemented in enum `Statistics` under name `SPEARMAN`. Because Spearman's coefficient compares two ranking of the same objects, it first discards the test neighbours that does not appear among the original neighbours. Then the Spearman's coefficient is computed as described in section 3.3.4.

Discounted Cumulated Gain

The cumulated gain, the discounted cumulated gain and the normalized discounted cumulated gain (described in section 3.3.3) are implemented in enum `Measures` under names `CG`, `DCG` and `NDCG` respectively. Discount factor for the discounted cumulated gain is log_2. Score $G(d)$ of an object d is computed from its position i among original neighbours:

$$G(d) = 1 - \frac{i-1}{k} \tag{4.3}$$

Where k is number of the original neighbours. For example if the number of original neighbours is 100, this gives score 1 for the first object and 0.01 for the last object. Objects that are not present among the original neighbours have score 0.

Position of Last

Position of last is measure that tells how much bigger amount of test neighbours is needed to get so they contain all the original neighbours. For first k object from the original neighbours, the position of the objects among the test neighbours are found and the largest position is reported as position of last for k. For an example see figure 4.8. Position of last is implemented in enum `Measures` under name `LAST`.

```
Original neighbours: a b c d e f
Test neighbours: o j d h r t a w c b n r u i k m n e g q y x f
Position of last for first object (a): 7
Position of last for first 2 objects (a, b): 10
Position of last for first 3 objects (a, b, c): 10
Position of last for first 6 objects (a, b, c, d, e, f): 23
```

Figure 4.8: Example of the position of last measure.

5 Evaluation and results

In this chapter experimental results are presented. Experiments were done on the first million from the 20 million of DeCAF descriptors of Profiset images. The energy of the PCA reduction is shown first, then precision is evaluated on different numbers of dimensions, training set sizes and various nearest neighbour searches. After that, the order of the nearest neighbours is investigated in the reduced data. Recall for different candidate set sizes is then presented. In the end of this chapter the possibility of use of the PCA reduction in real application is discussed.

5.1 Principal component analysis energy

Energy of principal component analysis suggests how much information is retained after a reduction. Total energy is the sum of the eigenvalues. Sum of first n eigenvalues is the retained energy of PCA reduction to n dimensions.

Retained energy was evaluated for all the possible reductions and eight sizes of training sets. By training set it is meant the data set that was used to compute an eigen matrix. Since the number of original dimensions is 4096, the smallest training set possible is 4096 objects. Training set sizes 5 000, 10 000, 15 000, 20 000, 25 000, 50 000, 100 000 and 1 000 000 were used.

Graph 5.1 shows the retained energy. Because the size of the eigenmatrix extracted from the training set is always 4096×4096, decrease in the retained energy can be seen when the training size grows. However, it can be seen that the decrease converges to zero as the training set size grows. E.g. for the reduction to 800 dimensions:

- When the training size grows ten times from 10 000 to 100 000, the retained energy decreases by 2.49%.
- When the training size grows ten times from 100 000 to 1 000 000, the retained energy decreases by 0.57%.

Thanks to that property, PCA transformation computed from larger training set should be always better because they have more information.

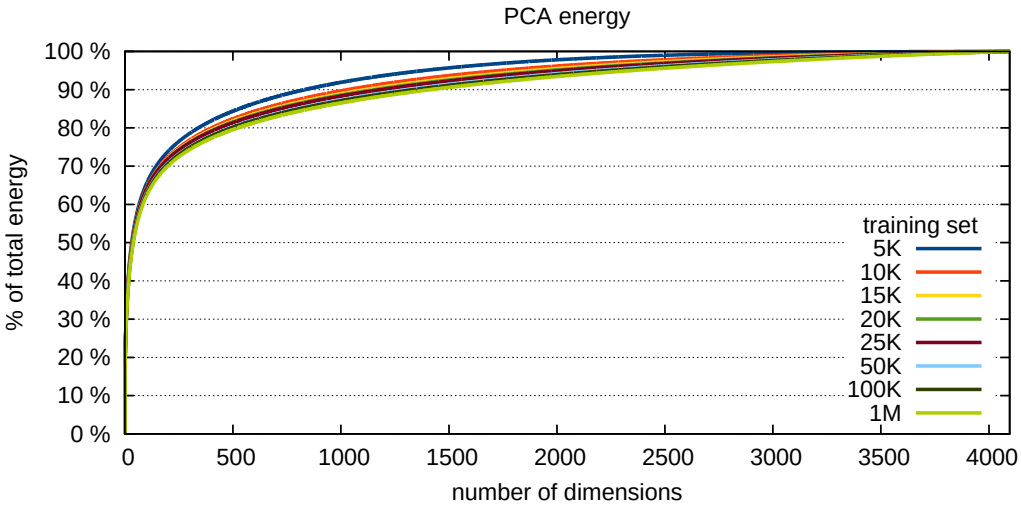

Figure 5.1: Retained energy for transformation matrices with different target number of dimensions and size of training set. Larger and detailed figure is at page 48.

5.2 Precision

Precision of the dimensionality reduction was evaluated for different target numbers of dimensions, training set sizes and for k-NN searches with various number of neighbours k:

- Dimensions: 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1 000, 1 500, 2 000, 2 500, 3 000, 3 500 and 4 000.
- Training sets: first 5 000, 10 000, 15 000, 20 000, 25 000, 50 000 and 100 000 objects from the data set.
- Number of neighbours: from 1 to 100.

The precision was evaluated on a data set of 1 million DeCAF image descriptors. First, 1 000 query object were randomly chosen from the data set. Then for each query object a k-NN search was performed on original data set to obtain k original neighbours. After that the reduced data set was generated and a k-NN search was performed on it. Finally, the two results were compared and averaged over the 1 000 queries to get the precision.

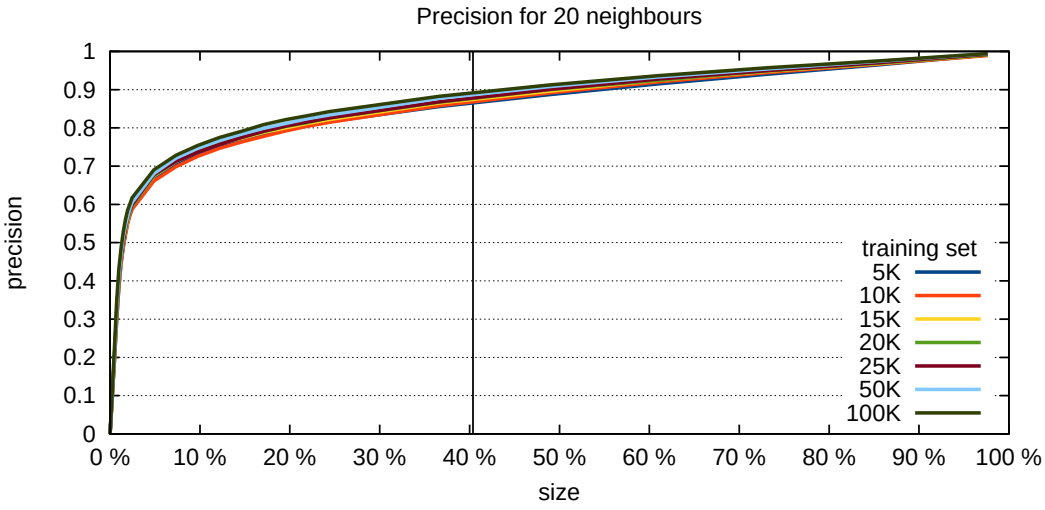

Figure 5.2: Average precision of 20 nearest neighbours search for different dimensionality reductions and various training set sizes. The x-axis denotes size (number of dimensions) in percentage of the original dimensionality (4096). The vertical line at 40.38% marks the compression ratio of the tested data (section 4.1.4). Larger and detailed figure is at page 49.

Because the distance of an object to itself is zero both in the original and in a reduced data set, the query object was ignored in all k-NN searches.

Precision for the 20-NN search over range of dimension sizes can be seen on graph 5.2. For example, precision 0.5, that means half of the retrieved objects on the reduced data set were the same as on the original, is possible with reduction to just 1.5% of the original size (60 dimensions). When the data is reduced to 10% size the precision is over 0.75 for the 100 000 training set.

It can be seen that there is not a big difference in precision among the different training set sizes. Nevertheless, larger training sets performed better than smaller sets. For this reason in the following tests the focus was on the 100 000 training set.

We can focus just on the reductions to lower than 40.38% dimensions, thanks to the compressibility of the data described in section 4.1.4. At the 40.38% mark the precision is almost 0.9 for the 100 000

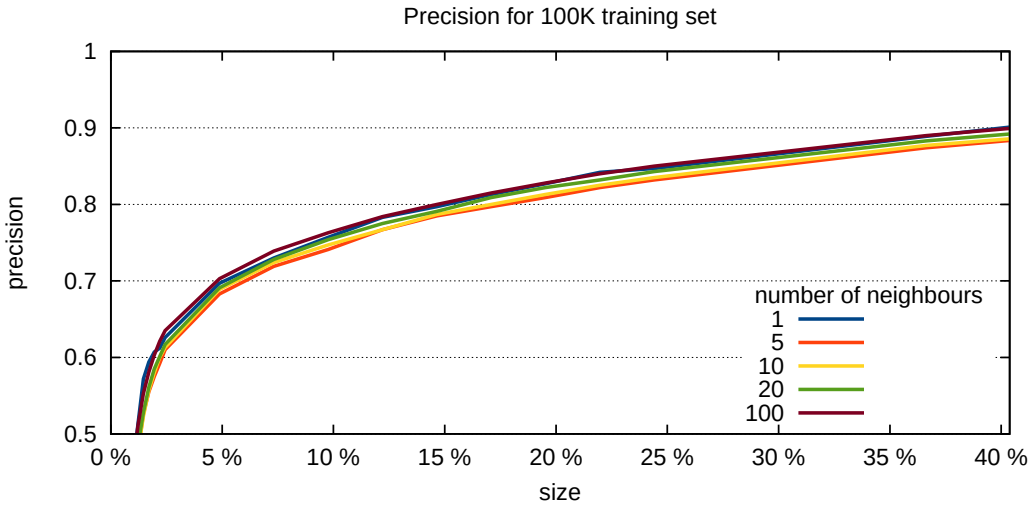

Figure 5.3: Average precision of k nearest neighbours search for different dimensionality reductions and number of neighbours. The x-axis denotes size (number of dimensions) in percentage of the original dimensionality (4096). Because of the compressibility (section 4.1.4) of the tested data only first 40.38% dimensions is shown. Larger figure is at page 50.

training set. If precision higher than 0.9 is required, the sparse vector compression should be considered instead of PCA.

The number of the nearest neighbours does not affect the precision much as can be seen on graphs 5.3 and 5.4. The stability of precision over the number of neighbours k suggests that the order and distribution of neighbours is preserved (up to the precision) in the transformed data set. If the precision rose with the k, order of the neighbours would not be preserved well (first the later object would be retrieved, then the earlier). If the precision declined with the k, the distribution of neighbourhoods would not be preserved well (first the immediate neighbours would be retrieved, then unexpected objects would be encountered).

Interesting trait of 1-NN search can be seen in graph 5.4: the precision of 1-NN search is noticeably larger than 2-NN search. Note that the query object in a k-NN search results were ignored, otherwise the precision for 1-NN search would always be 1. The anomaly of the 1-

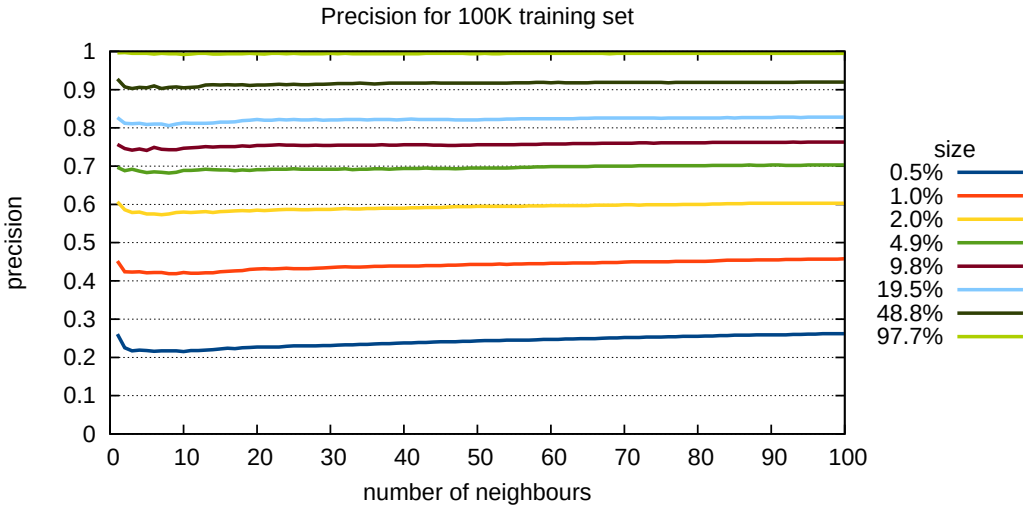

Figure 5.4: Average precision of k nearest neighbours search for different number of neighbours and various dimensionality reductions. Each line represents reduction to a particular size. Larger figure with more data is at page 51.

NN search suggest that in the tested data set there are duplicate data or pairs of abnormally close objects[1].

5.3 Order of neighbours

Besides precision, the order of retrieved neighbours is important because the most similar neighbours should be at the top of the results. Also, the correct order of the first neighbours is more important than the order neighbours at the end of the result.

The overall Spearman's correlation of the ranking of 100-NN from original set and ranking of the neighbours from reduced set can be seen in graph 5.5. The correlation is moderate to strong for sizes from 1% to 13% and very strong for larger sizes. The relationship of the PCA energy with precision and Spearman's coefficient can also be seen in the graph, confirming the PCA energy as a good estimate of the retained information.

1. Analysis of the first 100 000 objects of the data set revealed 41 duplicate objects.

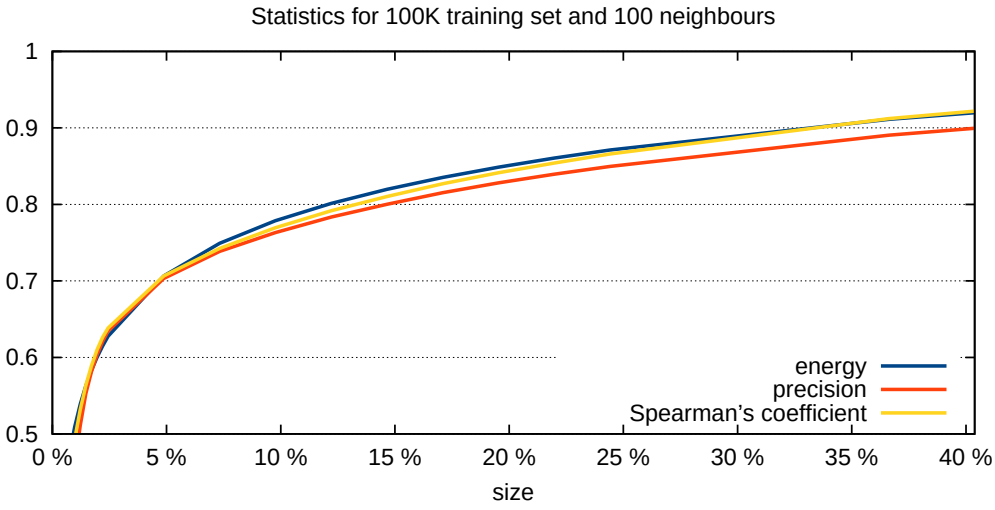

Figure 5.5: Compassion of PCA energy, precision and Spearman's coefficient for different dimensionality reductions. Larger figure is at page 52.

Graph 5.6 shows the normalized discounted cumulated gain described in section 3.3.3. It shows NDCG of the first 100 neighbours for the 100 000 training set. The parameters of the NDCG are specified in section 4.5.3. It can be seen that the order of the first few neighbours is better preserved than order of the latter.

Another look at the order of neighbours is to look at the positions of the original objects in the neighbours from a reduced data set. In other words, how many nearest neighbours in the reduced set is it necessary to be retrieved, so all the original neighbours are among the retrieved. For an example of the *position of last* measure see figure 4.8 on page 27.

In graph 5.7 it can be seen that for the first ten neighbours there is steep grow of required neighbours and after 50 neighbours the growth stabilizes. For example with reduction to 4.9%:

- The first nearest neighbour from the original set is at average position 1.9 among the neighbours from reduced set. To retrieve that neighbour, it is needed to look at 1.9 times more neighbours.

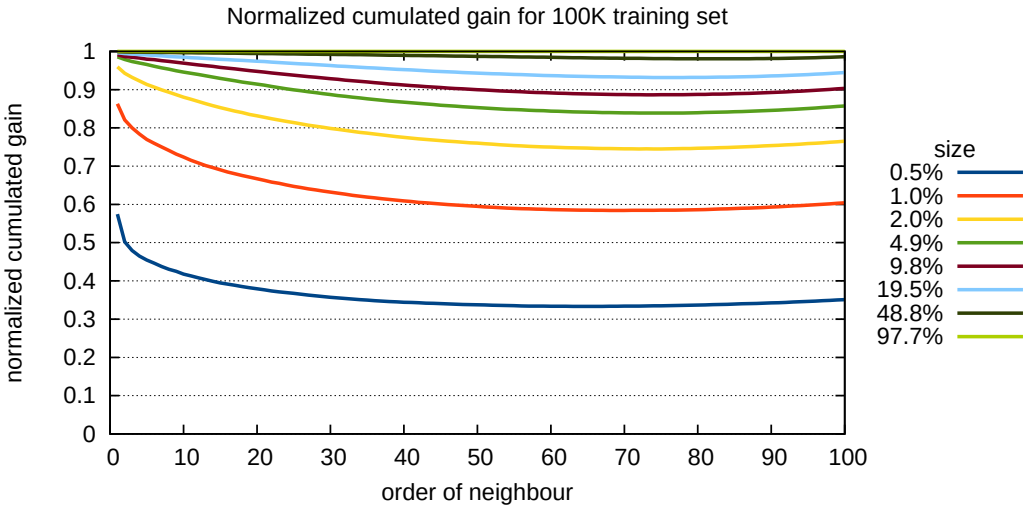

Figure 5.6: Normalized discounted cumulated gain for first 100 neighbours and various dimensionality reductions. Larger figure is at page 53.

- First 5 neighbours are among 13.6 (×2.7) neighbours and 10 among 31.4 (×3.1).
- First 50 neighbours are among 205.8 (×4.1) neighbours and 60 among 249.8 (×4.2).

This also indicates that the order of the first few neighbours is better preserved than the latter.

5.4 Search refinement

When high precision is required the reduced data set can be used to efficiently find a candidate set of neighbours and then refine the set by using the original data. Thanks to its smaller size the reduced data set can be stored in main memory and only the candidate objects are accessed from disk. This way it is possible to obtain highly accurate results more efficiently. The candidate set is typically several times larger than number of the desired neighbours.

For example, graph 5.7 shows that with reduction to 1% of size, 20-NN search needs to access 500 objects in average to retrieve all the

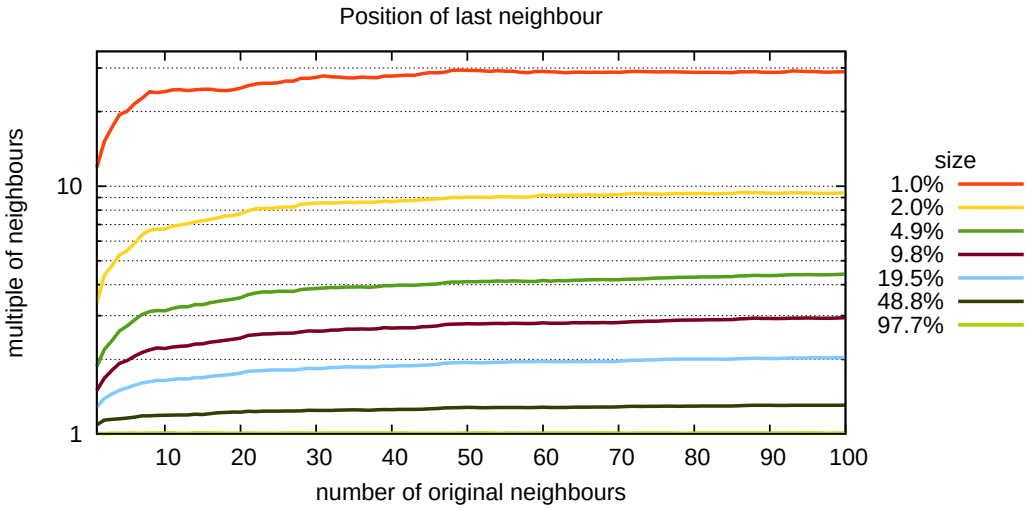

Figure 5.7: Average position of the last neighbour in k nearest neighbours search for different number of neighbours and various dimensionality reductions. The x-axis specify k for k-NN search on original data. The y-axis shows the average number of neighbours (in the multiple of the number of the original neighbours) that were necessary to be retrieved to cover all the original neighbours. Each line represents reduction to a particular size. Larger figure with more data is at page 54.

original neighbours. However, the average number does not work in determining how big the candidate set should be so it would contain all the exact neighbours. Better suggestion can be derived from the distribution of the last positions.

The distribution of the last neighbours in 100-NN search can be seen in graph 5.8. Considering reduction to 4.9% size and 100-NN search it can be seen that 99.9% of the queries accessed up to 2043 objects and 85% of queries accessed up to 680 objects. To put it differently, if the candidate set is 20 times larger than the desired amount of neighbours, there is 99.9% chance that all the correct neighbours are accessed. However, when some imprecision is allowed, it is possible to have notably smaller candidate sets.

Graph 5.9 shows how is recall affected by the size of the candidate set. For example when the candidate set is 10 times larger (200 neigh-

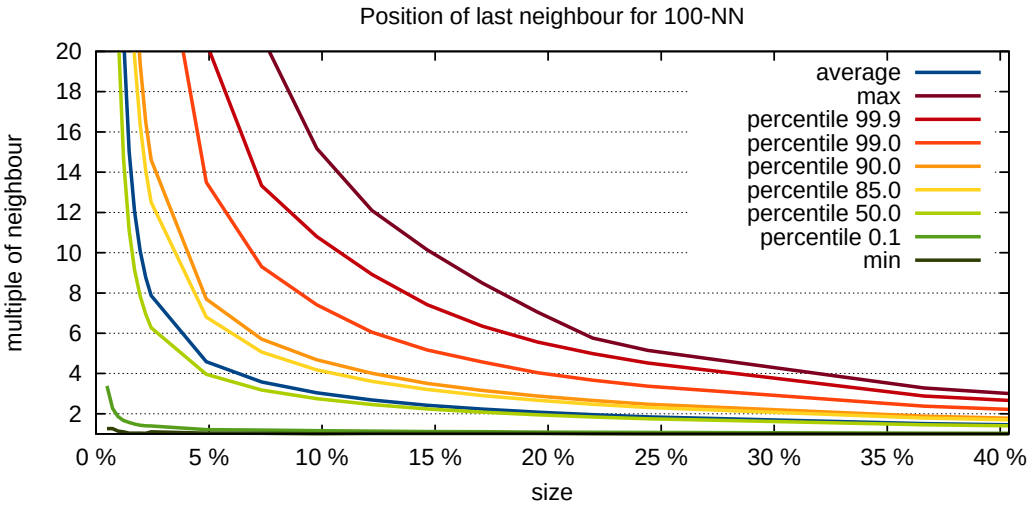

Figure 5.8: Distribution of the last neighbours in 100 nearest neighbours search for different dimensionality reductions. On the reduced data as many neighbours were retrieved as was needed to retrieve 100 nearest neighbours from original data. Larger figure with more data is at page 55.

bours), for the reduction to 2% size the recall is 0.98, that means in average 19.6 out of 20 neighbours are found in the candidate set.

Based on all the results several configurations were selected for further analysis. Reduction to 40 (1%), 80 (2%), 200 (4.9%) and 400 (9.8%) dimensions. Reduction to fewer than 40 dimensions does not preserve enough information and reduction to more than 400 dimension does not significantly lower the size. Summary of characteristics for these reductions, as well as for the original data set and for data set compressed by sparse vector compression, are in table 5.10

5.5 Size and speed

Size of the selected data sets were measured in two ways. The first is raw size of the data object. For instance the original objects consists of 4096 32-bit floats, that is 128 Kib. The individual sizes can be seen in table 5.11, the size of sparse vector compression is an average size over the 1 million objects. The second measurement is the size of 1

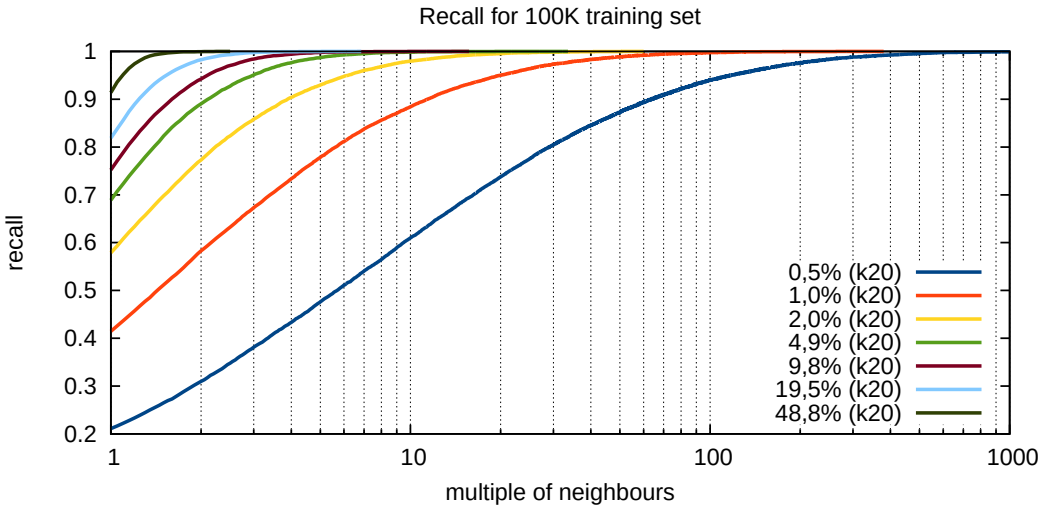

Figure 5.9: Average recall of 20-NN search on original data for different number of neighbours on reduced data and various dimensionality reductions. The x-axis denotes number of neighbours on reduced data as multiple of the number of the original neighbours. Each line represents reduction to a particular size. Larger figure with more data is at page 56.

million objects in gzip compressed MESSIF text data file (see example in figure 4.2). The relatively minor reduction of SVC data set (reduced by 9%, in contrast to 60% reduction of the raw object) suggest that the gzip compression handle the frequent zeros in the original data set well.

Effect of the dimensionality reduction on the speed of k-NN search was evaluated using *file sequential scan* and *vantage point tree*. In both cases 10-NN search was used. The file sequential scan was performed on 500 000 objects stored in the gziped format. The second algorithm used 100 000 objects stored in vantage point tree in memory. The results can be seen again in table 5.11. The PCA reduced data set are clearly faster, since fewer data is needed to read from disk and distance computation is computed on smaller vectors. Surprising was the bad behavior of sparse vector compression – the distance computation on the compressed object is evidently very slow. There might be other implementations of sparse vector compression with faster

Data set	Precision			Recall 0.99 at		
	10-NN	20-NN	100-NN	10-NN	20-NN	100-NN
Original	1.0			9.9	19.8	99
SVC	1.0			9.9	19.8	99
PCA 10% (400)	0.747	0.754	0.763	39	68	315
PCA 5% (200)	0.689	0.691	0.703	64	108	485
PCA 2% (80)	0.580	0.585	0.603	174	282	1082
PCA 1% (40)	0.422	0.431	0.458	708	1064	3266

Figure 5.10: Precision and size of candidate set with recall 0.99 for original data set, sparse vector compression and PCA reductions to different sizes (number of dimensions).

Data set	Size		Time	
	object	1M gzip	500K FSS	100K VPT
Original	128 Kib	6.55 GiB	348.7 s	360.5 ms
SVC	51.68 Kib	5.98 GiB	228.7 s	2247.2 ms
PCA 10% (400)	12.50 Kib	1.77 GiB	76.7 s	47.0 ms
PCA 5% (200)	6.25 Kib	0.89 GiB	39.0 s	29.6 ms
PCA 2% (80)	2.50 Kib	0.36 GiB	15.9 s	19.5 ms
PCA 1% (40)	1.25 Kib	0.18 GiB	8.3 s	14.7 ms

Figure 5.11: Size and search time of different data sets. *Object size* – raw data object. *Gzip size* – gziped MESSIF text data file. *FSS time* – file sequential scan algorithm. *VPT time* – in-memory vantage point tree algorithm.

distance function; however, evaluation of different implementations is out of the scope of this work.

5.6 Suitability for application

It was shown that larger training sets perform better. If the PCA reduction method were to be used on the Profiset data set, I would suggest that all the 20 million data would be used to compute the PCA transformation matrix.

By using the reduction to 10% (400) dimensions, the total size of the 20 million data set would be 29.8 GiB for the data only. This could fit to memory of a server machine and would speed up the search significantly. But the precision around 0.75 might not be sufficient for many applications.

The second approach is the candidate set refinement. With this approach the reductions to 2% or 1% would be more favourable. The 20 million data set would be reduced to just 6 GiB or 3 GiB. This size could fit even to memory of a desktop computer. The candidate set refinement of 200 to 3300 objects would result in precision around 0.99. This should be more than sufficient for most applications.

The benefit of the candidate set generated from the PCA reduced data set is that it is ordered by distance. This could be used in a two-step k-NN search. First, the candidate set is generated for a query and user receives fast reply containing first k neighbours from the candidate set. This reply, considering the 1% reduced set, has precision only just over 0.4. However, the user might already get the result he seeks or the application (web browser) may start a processing of the results like fetching the thumbnail images. Then, when the refinement is done, the user receives the refined set. The application then may just update the view and process smaller amount of data (fetch only missing thumbnails.)

For a reall application the parameters would have to be tweaked to fit the application needs, but the presented results could be used as a starting point.

6 Conclusion

The goal of this thesis was to implement and evaluate a dimensionality reduction method on an image descriptor data set. I have selected a linear feature extraction method – *principal component analysis*. I implemented a set of tools, that perform the dimensionality reduction, using Java programming language and a library for basic linear algebra JAMA. The performance of the dimensionality reduction using the principal component analysis was evaluated on a collection of 1 million DeCAF image descriptors.

Conventional dimensionality reduction methods were presented in two sections – linear methods in section 2.4 and nonlinear methods in section 2.5. The reasons for selecting PCA for implementation and testing are explained in section 4.2.

The dimensionality reduction was evaluated on a data set provided by laboratory DISA of Masaryk University. The data set consist of state-of-the-art image descriptors of 20 million images. The data set was described in section 4.1. To evaluate the PCA method on the problem of similarity search, the similarity search framework MESSIF was used.

The results of the evaluation are presented in chapter 5. In section 5.6 the applicability of the PCA was discussed. I offered a possibility to use a data set reduced by the PCA method to generate candidate sets for a k nearest neighbours search and then refine the candidate set by using the original data.

For further research I would suggest a comparison of the PCA candidate set refinement approach to similar approaches that use candidate set refinement, such us PPP codes.

I believe I have completed all the points that were requested and that I have shown the principal component analysis as a viable option to reducing the size of large image descriptor databases.

Bibliography

1. TANG, Jiliang; ALELYANI, Salem; LIU, Huan. Feature Selection for Classification: A Review. In: *Data classification: algorithms and applications*. Boca Raton: CRC Press, Taylor & Francis Group, [2014], pp. 37–64. ISBN 1466586745.
2. FODOR, Imola K. *A survey of dimension reduction techniques* [online]. Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, 2002 [visited on 2016-04-10]. Available from: `https://computation.llnl.gov/casc/sapphire/pubs/148494.pdf`.
3. BURGES, Christopher J. C. Dimension Reduction: A Guided Tour. *Foundations and Trends in Machine Learning*. 2010, vol. vol. 2, no. issue 4, pp. 275–364. ISSN 19358237. Available from DOI: `10.1561/2200000002`.
4. BELLMAN, Richard. *Adaptive control processes: a guided tour*. Princeton, N.J.: Princeton University Press, 1961.
5. CEVIKALP, Hakan. *Feature extraction techniques in high-dimensional spaces: Linear and nonlinear approaches*. Nashville, Tennessee, 2005.
6. MITCHELL, Tom M. *Machine learning*. Boston: McGraw-Hill, c1997. ISBN 0070428077.
7. KIRA, Kenji; RENDELL, Larry. The Feature Selection Problem: Traditional Methods and a New Algorithm. In: *AAAI-92*. 1992, pp. 129–134.
8. LESKOVEC, Jurij; RAJARAMAN, Anand; ULLMAN, Jeffrey D. *Mining of massive datasets*. 2nd ed. Cambridge: Cambridge University Press, c2014. ISBN 9781107077232.
9. AMOEBA, http://stats.stackexchange.com/users/28666/amoeba. *Making sense of principal component analysis, eigenvectors & eigenvalues* [Cross Validated]. 2015 [visited on 2016-05-10]. Available from: `http://stats.stackexchange.com/revisions/140579/5`.
10. WANG, Xuechuan; PALIWAL, Kuldip K. Feature extraction and dimensionality reduction algorithms and their applications in vowel recognition. *Pattern recognition*. 2013, vol. 36, no. 10, pp. 2429–2439.
11. SCHÖLKOPF, Bernhard; SMOLA, Alexander; MÜLLER, Klaus-Robert. Kernel principal component analysis. *Artificial Neural Networks – ICANN'97*. 1997, pp. 583–588.
12. TENENBAUM, Joshua B; SILVA, Vin De; LANGFORD, John C. A global geometric framework for nonlinear dimensionality reduction. *Science*. 2000, vol. 290, pp. 2319–2323.
13. KAVRAKI, Lydia E. Dimensionality Reduction Methods for Molecular Motion. In: [online]. OpenStax CNX, 12.6.2007 [visited on 2016-04-21]. Avail-

able from: http://cnx.org/contents/02ff5dd2-fe30-4bf5-8e2a-83b5c3dc0333@10.

14. ROWEIS, Sam T.; SAUL, Lawrence K. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*. 2000, vol. 290, no. 5500, pp. 2323–2326.
15. BAEZA-YATES, Ricardo; RIBEIRO-NETO, Berthier. *Modern information retrieval: the concepts and technology behind search*. 2nd ed. Harlow: Pearson, 2011. ISBN 9780321416919.
16. ZEZULA, Pavel; AMATO, Giuseppe; DOHNAL, Vlastislav; BATKO, Michal. *Similarity Search: The Metric Space Approach*. New York, NY: Springer, 2005. ISBN 0387291512.
17. DONAHUE, Jeff; JIA, Yangqing; VINYALS, Oriol; HOFFMAN, Judy; ZHANG, Ning; TZENG, Eric; DARRELL, Trevor. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. *Proceedings of the International Conference on Machine Learning*. 2014, pp. 647–655.
18. KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*. 2012, vol. 25, pp. 1097–1105.
19. BUDIKOVA, Petra; BATKO, Michal; ZEZULA, Pavel. Evaluation Platform for Content-based Image Retrieval Systems. In: *International Conference On Theory And Practice Of Digital Libraries 2011*. Berlin: Springer, 2011, pp. 130–142. ISBN 978-3-642-24468-1.
20. BATKO, Michal; NOVÁK, David; ZEZULA, Pavel. MESSIF: Metric Similarity Search Implementation Framework. In: *Digital Libraries: Research and Development, Lecture Notes in Computer Science*. 4887th ed. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 1–10. ISBN 978-3-540-77087-9.

A Archive content

Archive name: `DR_for_vector_spaces.zip`

Folder `Diploma` contains soure codes.
Documentation is in file `Diploma/README.md`.
File `Diploma.jar` is compiled project packaged with dependencies.
Folder `graphs` contains gnuplot scripts and data files for graphs.
Folder `graphs/out` contains generated graphs.
Folder `results` contains following libreoffice calc files:

- `zeros.ods` – histogram of number of zeros in the original data.
- `eigenpairs.ods` – PCA energy for different training set sizes.
- `1M_test.ods` – results of evaluation on 1 million data.
- `speed.ods` – results of speed evaluation.

B Graphs

- Graph B.1: PCA energy.
- Graph B.2: precision for 20 neighbours.
- Graph B.3: precision for 100K training set.
- Graph B.4: precision for 100K training set.
- Graph B.5: statistics for 100K training set and 100 neighbours.
- Graph B.6: normalized cumulated gain for 100K training set.
- Graph B.7: position of last neighbour.
- Graph B.8: position of last neighbour for 100-NN.
- Graph B.9: recall for 100K training set.

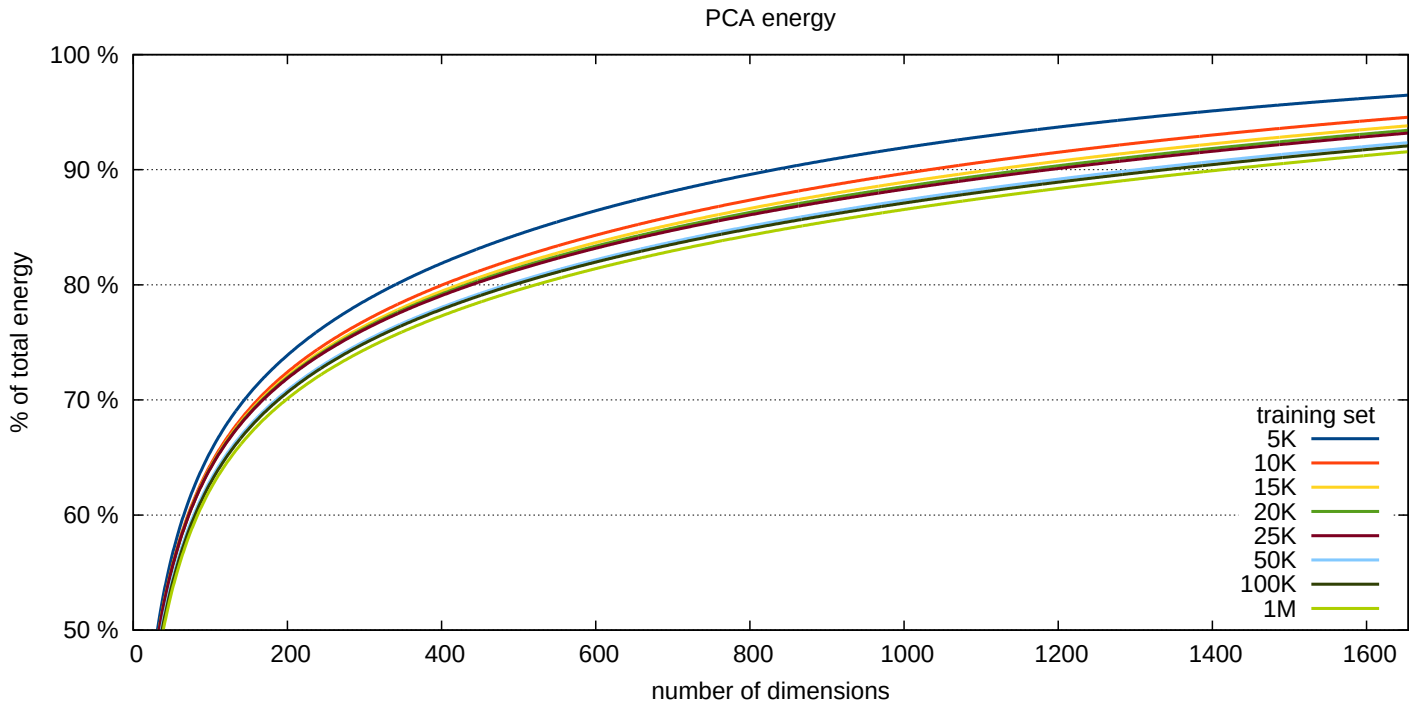

Figure B.1: Retained energy for transformation matrices with different target number of dimensions and size of training set. Because of the compressibility (section 4.1.4) of the tested data, this detailed graph shows only first 1654 dimensions.

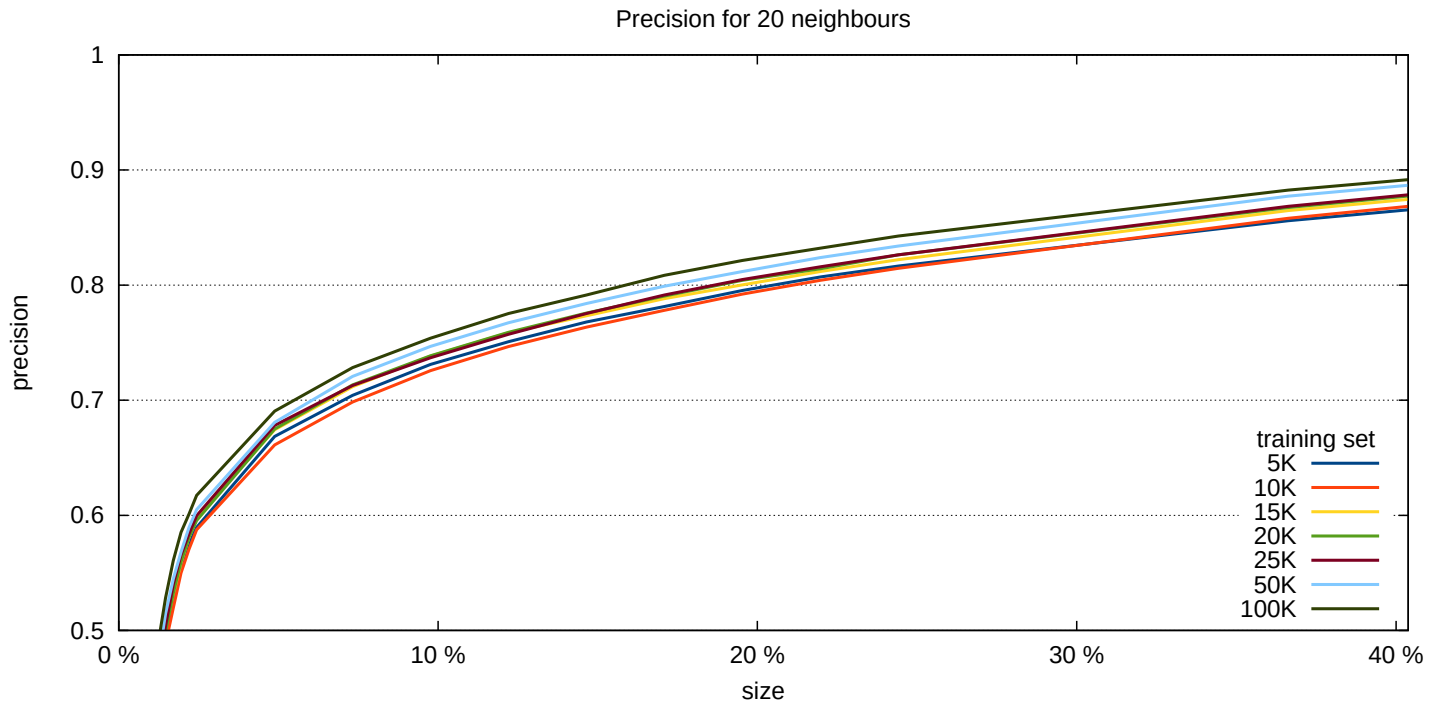

Figure B.2: Average precision of 20 nearest neighbours search for different dimensionality reductions and various training set sizes. The x-axis denotes size (number of dimensions) in percentage of the original dimensionality (4096). Because of the compressibility (section 4.1.4) of the tested data, this detailed graph shows only first 40.38% dimensions.

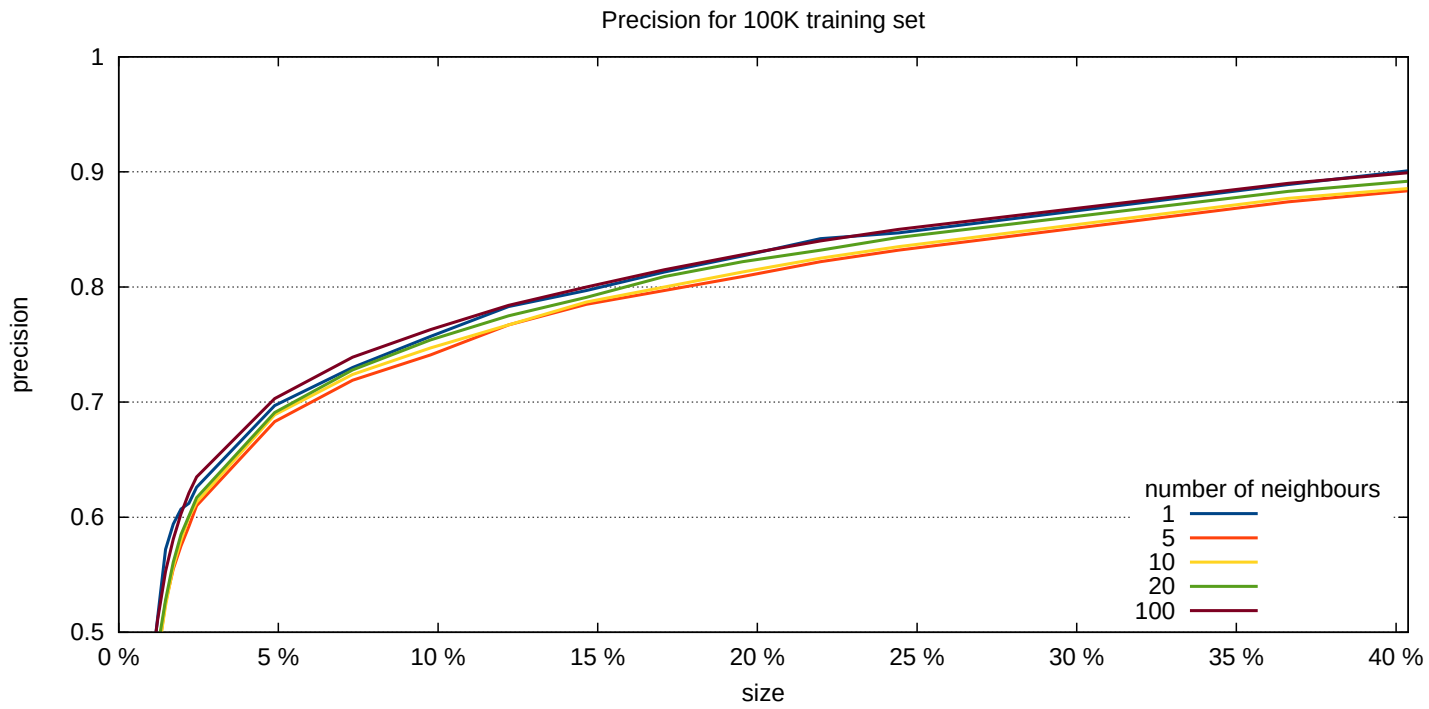

Figure B.3: Average precision of k nearest neighbours search for different dimensionality reductions and number of neighbours. The x-axis denotes size (number of dimensions) in percentage of the original dimensionality (4096). Because of the compressibility (section 4.1.4) of the tested data only first 40.38% dimensions is shown.

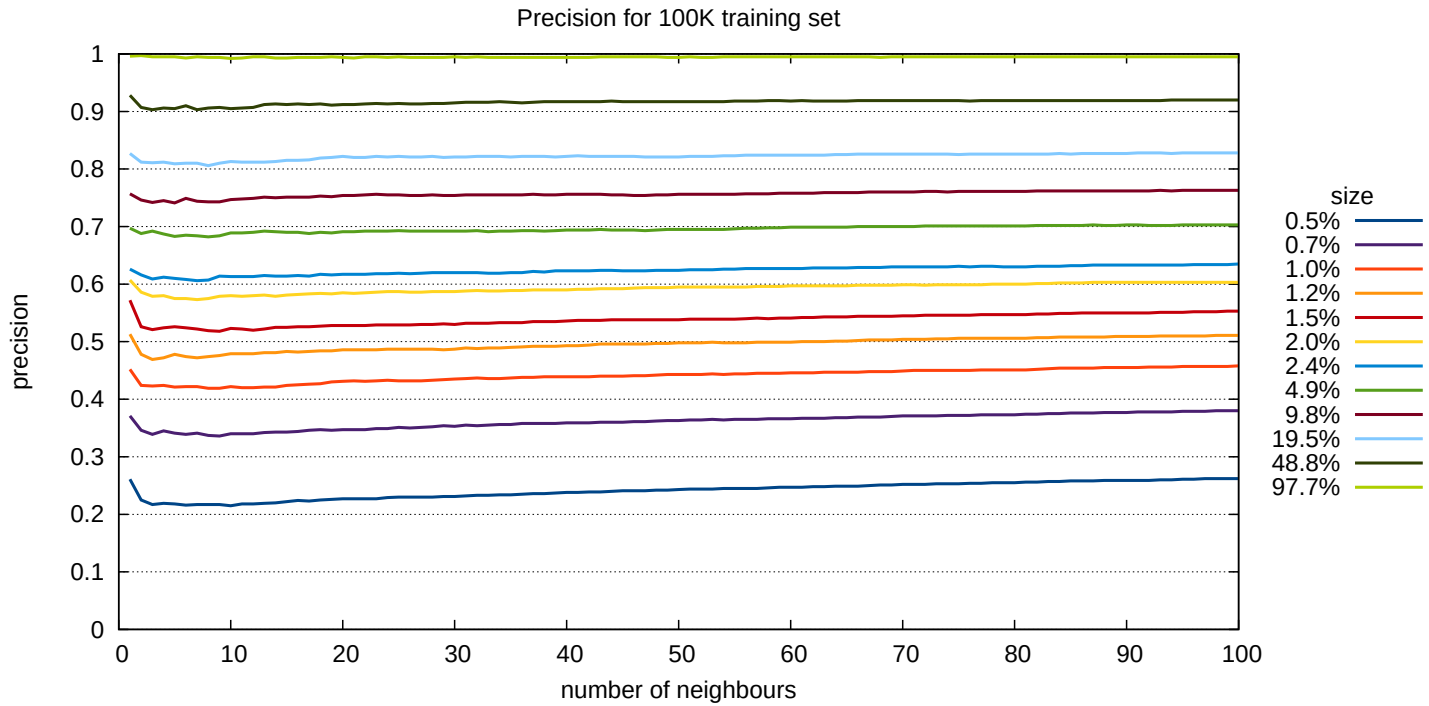

Figure B.4: Average precision of k nearest neighbours search for different number of neighbours and various dimensionality reductions. Each line represents reduction to a particular size.

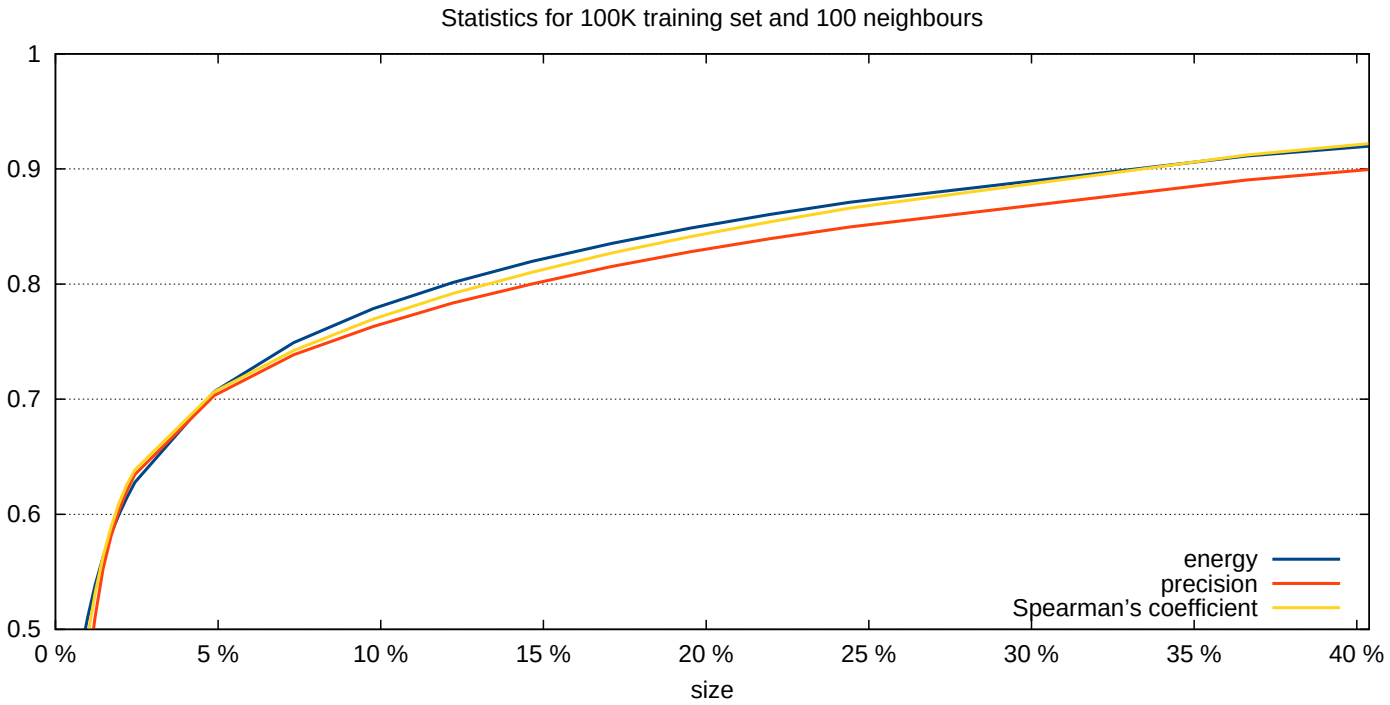

Figure B.5: Comparsion of PCA energy, precision and Spearman's coefficient for different dimensionality reductions.

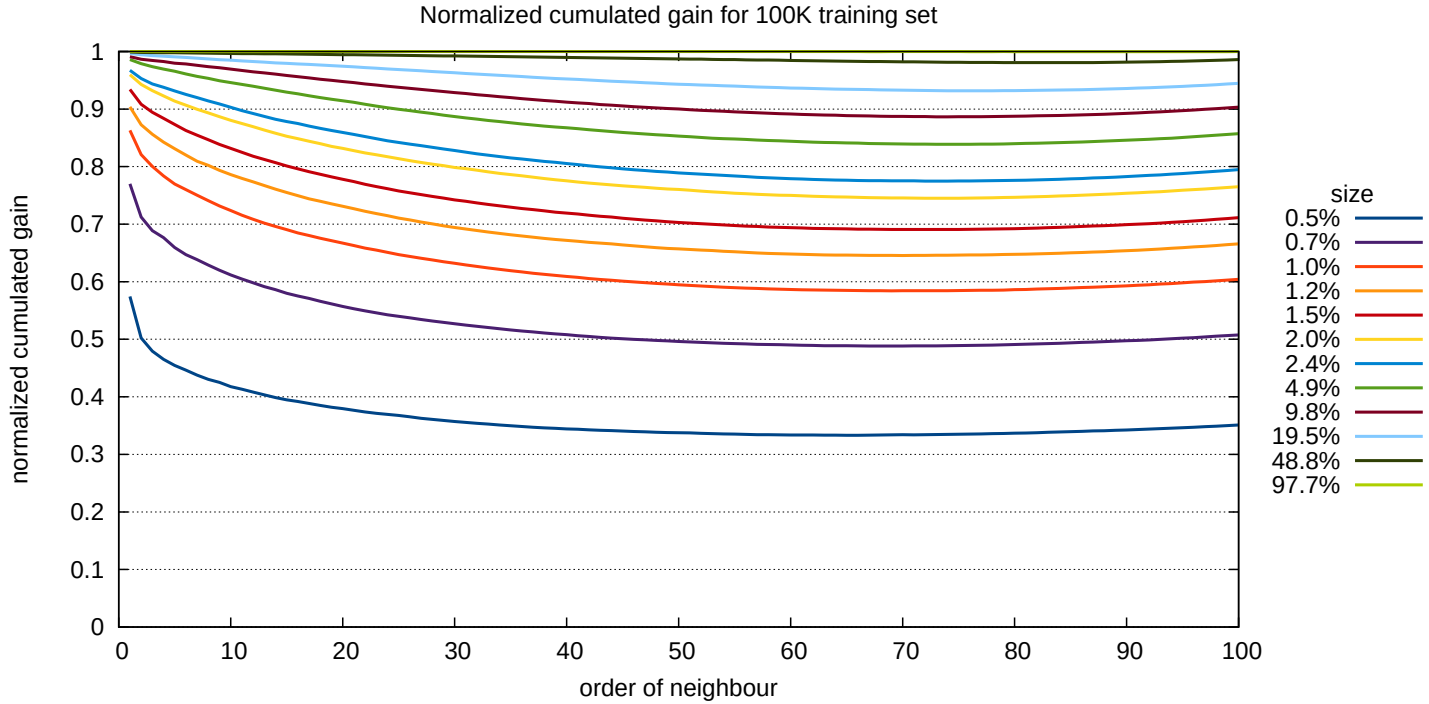

Figure B.6: Normalized discounted cumulated gain for first 100 neighbours and various dimensionality reductions.

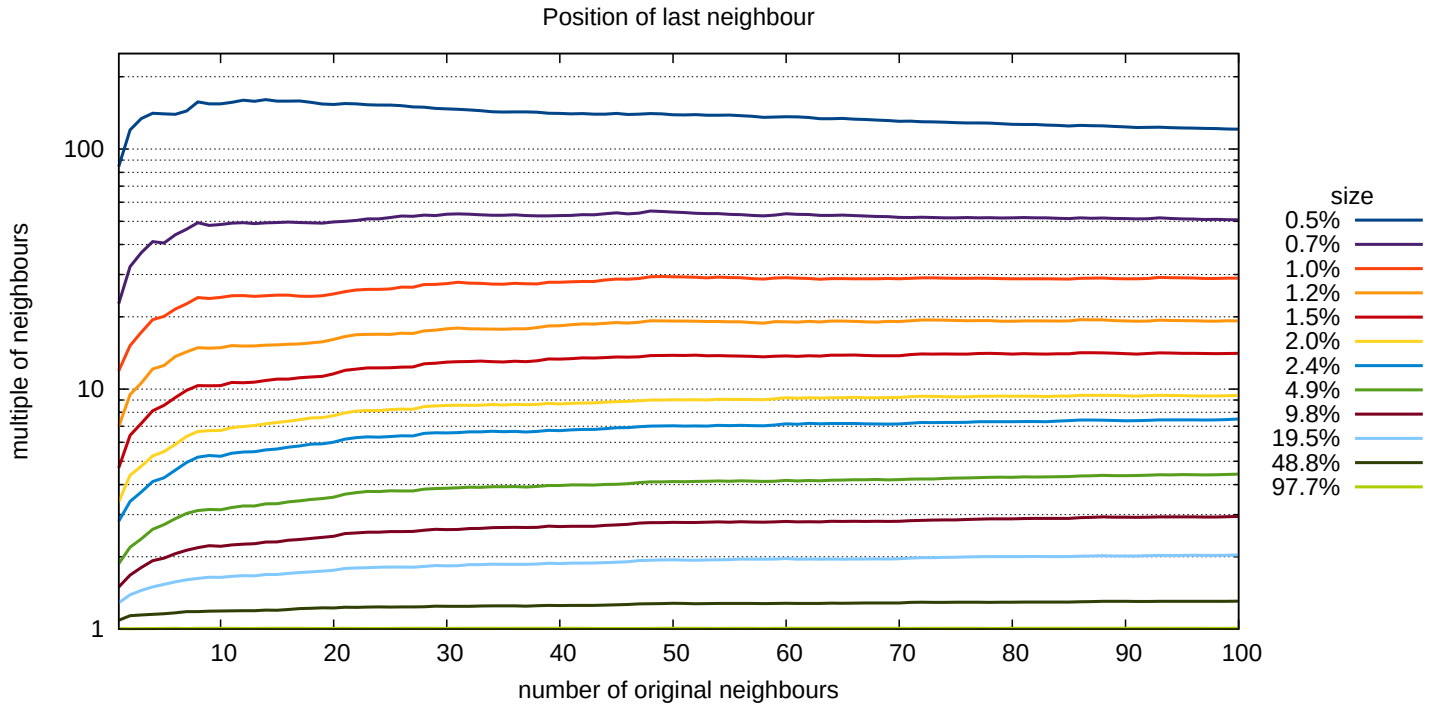

Figure B.7: Average position of the last neighbour in k nearest neighbours search for different number of neighbours and various dimensionality reductions. The x-axis specify k for k-NN search on original data. The y-axis shows the average number of neighbours (in the multiple of the number of the original neighbours) that were necessary to be retrieved to covr all the original neighbours. Each line represents reduction to a particular size.

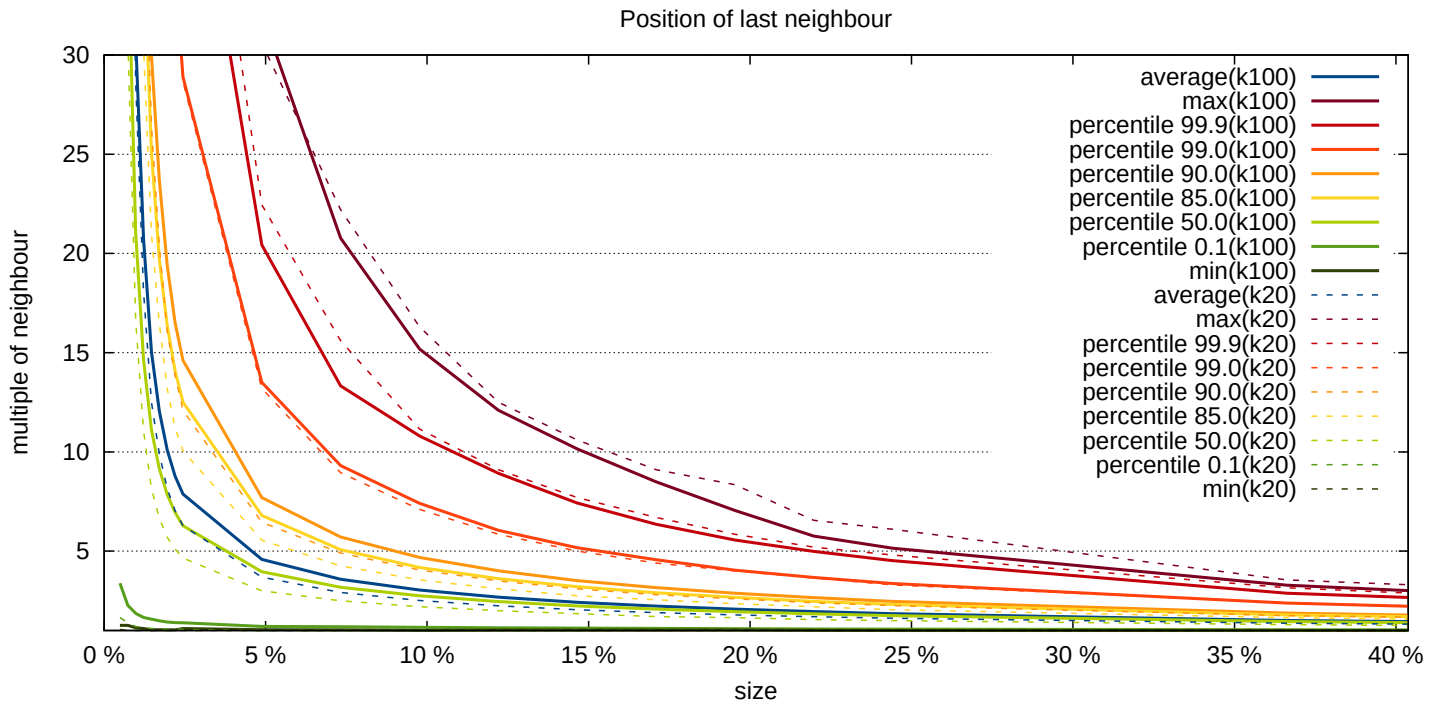

Figure B.8: Distribution of the last neighbours in 20 and 100 nearest neighbours search for different dimensionality reductions. On the reduced data as many neighobours were retrieved as was needed to retrieve 100 nearest neighbours from original data.

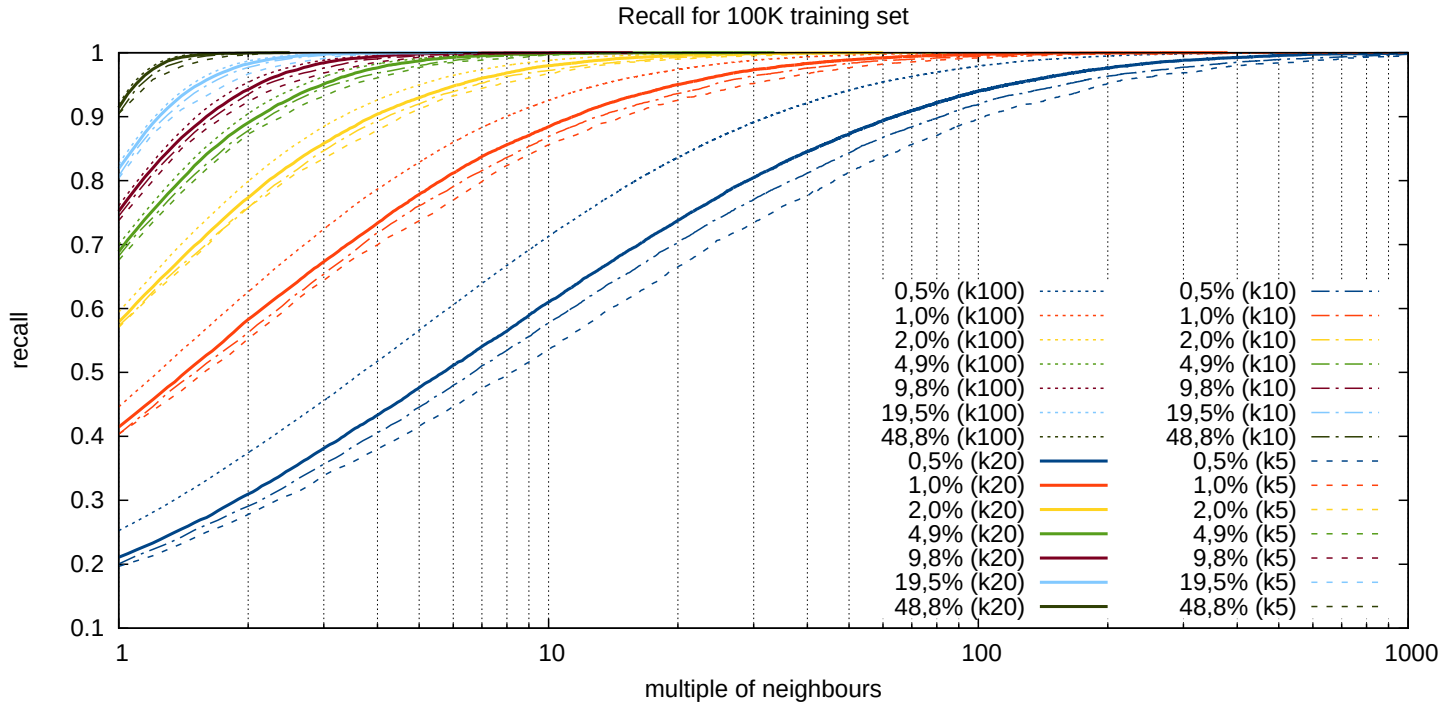

Figure B.9: Average recall of 5, 10, 20 and 100 nearest neighbours search on original data for different number of neighbours on reduced data and various dimensionality reductions. The x-axis denotes number of neighbours on reduced data asultiple of the number of the original neighbours. Each line line represents reduction to a particular size.