ADVANCED LEARNING FOR TEXT AND GRAPH DATA
MASTER DATA SCIENCE
Lab 1: Dimensionality Reduction

Fragkiskos Malliaros, Antoine Tixier and Michalis Vazirgiannis

February 4, 2016

1 Description

The goal of this lab is to study several *dimensionality reduction* techniques, both unsupervised and supervised, and to examine how they can be applied on real data. We give a brief introduction of the basic dimensionality reduction techniques that will be used in this lab, and then we describe the tasks that need to be performed.

2 Unsupervised Dimensionality Reduction Techniques

The goal of a dimensionality reduction technique is to transform the data from a higher dimensional space into a lower dimensional one, such that uninformative variance in the data is discarded, or such that a subspace in which the data lives is detected. Let $\mathbf{X} = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}),\ \mathbf{x_i} \in \mathbb{R}^n$ be our dataset. The goal of a dimensionality reduction technique is to find a credible mapping of the m vectors to $\mathbb{R}^k, k \ll n$, maintaining as much as possible the variation/distances of the data. There are many reasons why dimensionality reduction is a useful tool in data mining and machine learning. For example, in many cases, not all the measured variables are important for understanding the underlying phenomena of interest.

We should note that, dimensionality reduction techniques differ from the so-called *feature selection* techiques, which retain a subset of the initial features of the dataset. On the other hand, dimensionality reduction techniques map the data into a new representation space, creating an alternative, smaller set of variables – defined as functions over all features – to represent the data. Additionally, in contrast to feature selection techniques, most of the dimensionality reduction methods do not take into account the class labels associated with the instances of the data (e.g., labels that can be used in the classification task). In the second part of the lab, we will see how this can be done with the LDA method.

In this lab, we will focus on the following two unsupervised dimensionality reduction methods:

- SVD (Singular Value Decomposition)
- PCA (Principal Component Analysis).

Next, we will briefly present each one of these methods and we describe the tasks that need to be done in the lab.

2.1 SVD (Singular Value Decomposition)

Singular Value Decomposition (SVD) is a matrix decomposition method that leads to a low-dimensional representation of a high-dimensional matrix. Let $\mathbf{A}$ be a $m \times n$ matrix. Then the Singular Value Decomposition of matrix $\mathbf{A}$ is defined as

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}}$$

where

- $\mathbf{U}$: $m \times m$ matrix has as columns the eigenvectors of $\mathbf{A}\mathbf{A}^{\mathbf{T}}$
- Σ : $m \times n$ is a diagonal matrix with the singular values of $\mathbf{A}$ in the diagonal (= square roots of $\mathbf{A}\mathbf{A}^{\mathbf{T}}$ eigenvalues)
- $\mathbf{V}$: $n \times n$ matrix has as columns the eigenvectors of $\mathbf{A}^{\mathbf{T}}\mathbf{A}$.

SVD allows an exact representation of any matrix, and also makes it easy to eliminate the less important parts of that representation in order to produce an approximate representation with any desired number of dimensions – leading to the concept of *low rank approximation* of a matrix. Let $\mathbf{A}$ be a $m \times n$ matrix, where $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}}$. Then, a r rank approximation of $\mathbf{A}$ is given by

$$\mathbf{Y} = \mathbf{U}_{m\times r}\text{diag}(\sigma_1, \ldots, \sigma_r){\mathbf{V}^{\mathbf{T}}}_{r\times n}.$$

How to perform dimensionality reduction with SVD?

The goal of dimensionality reduction is to find an approximation $\mathbf{Y}$ of $\mathbf{A}$ using r dimensions instead of the original n, $r < n$. This can be done using the properties of SVD: $\mathbf{Y}_{m\times n} = \mathbf{U}_{m\times r}\mathbf{\Sigma}_{r\times r}{\mathbf{V}^{\mathbf{T}}}_{r\times n}$. In practice, however, the purpose is not to actually reconstruct the original matrix, but to use the reduced dimensionality representation $\mathbf{U}_{m\times r}\mathbf{\Sigma}_{r\times r}{\mathbf{V}^{\mathbf{T}}}_{r\times n}$ in order to analyze the data.

The quality of low rank approximation $\mathbf{Y}$ of a matrix $\mathbf{A}$ can be evaluated using the *Frobenious norm* $\| \mathbf{A} - \mathbf{Y} \|_{\mathbf{F}}$, where

$$\| \mathbf{X} \|_{\mathbf{F}} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |x_{ij}|^2} = \sqrt{\sum_{i=1}^{\min(m,n)} \sigma_i^2}.$$

The best low rank approximation is the one that minimizes the Frobenius norm $\| \mathbf{A} - \mathbf{Y} \|_{\mathbf{F}}$.

Practical issues: How many singular values should we retain?

A useful rule of thumb is to retain enough singular values to make up 90% of the energy in $\mathbf{\Sigma}$. That is, the sum of the squares of the retained singular values should be at least 90% of the sum of the squares of all singular values.

2.1.1 Tasks to be Done in the Lab

Fill in the Python script `my_svd.py` under the directory `Code/SVD`, in order to implement and apply the SVD decomposition on a matrix $\mathbf{X}$ that corresponds to an image. Then, you will need to reconstruct the original matrix (image) $\mathbf{X}$ using the top (largest) $k = \{10, 20, 50, 100, 200\}$ singular values. Compare (visually) these approximations to the original image. What is the error of each approximation? What do you observe? Hint: examine the distribution of the singular values of the original matrix $\mathbf{X}$.

2.2 Principal Component Analysis (PCA)

Principal components analysis (PCA) is a very popular technique for dimensionality reduction. Given a set of data on n dimensions, PCA aims to find an optimal linear subspace of dimension $d < n$ such that the variance of the observations projected onto that subspace is maximized. The linear subspace can be specified by d orthogonal vectors that form a new coordinate system, called the *principal axes* or *principal directions*. The principal axes are linear transformations of the original dimensions, so there can be no more than n of them. However, the hope is that only $d < n$ principal axes are needed to approximate the space spanned by the n original dimensions. Thus, for a given set of data vectors $x_i, i \in 1 \ldots t$, the d principal axes are those orthonormal axes onto which the variance retained under projection is maximal. This transformation is defined in such a way that the first principal axis lies in the direction of maximum variability of the data (explains as much of the variance as possible), and each succeeding axis in turn accounts for the highest possible variance under the constraint that it is orthogonal to (i.e., uncorrelated with) the preceding axes.

Recall that the variance of a random variable is given by the following formula: $V(X) = \sigma^2 = E[(X - \mu)^2]$. PCA can be computed using the eigenvalue decomposition of the covariance matrix as follows.

1. Suppose that the data is organized into an $m \times n$ matrix $\mathbf{A}$, initially subtract mean values from columns: $\mathbf{C} = \mathbf{A} - \mathbf{M}$
2. Calculate the covariance matrix $\mathbf{W} = \mathbf{C^T C}$
3. Find eigenvalues and eigenvectors of the covariance matrix $\mathbf{W}$
4. *Principal Directions*: the k eigenvectors $\mathbf{U}_{[1\ldots k]}$ that correspond to the k largest eigenvalues
5. Project the data to the new space: $\mathbf{CU}_{[1\ldots k]}$

Can we compute the principal components using the SVD decomposition? Recall that the square root of the eigenvalues of the covariance matrix $\mathbf{C^T C}$ corresponds to the singular values of $\mathbf{C}$. That way, in step 2, we can compute the SVD decomposition of $\mathbf{C}$, and the principal axes will correspond to the singular vectors that correspond to the k largest singular values.

As we have already discussed, PCA transforms the data into a lower dimensional space preserving the variance. The fraction of variance that is preserved in the transformed data can be captured by the following formula:

$$var = \frac{\sum_{i=0}^{k} \lambda_i}{\sum_{j=0}^{n} \lambda_j},$$

where n is the number of dimensions in the original dataset, k is the number of dimensions in the new representation space, and λ are the eigenvalues of the covariance matrix sorted in descending order.

2.2.1 Tasks to be Done in the Lab

Fill in the Python script `my_pca.py` under the directory `Code/PCA`, in order to implement the PCA technique. Then, PCA will be applied to a dataset in order to project the data into a k-dimensional space defined by the first k principal axes. Here, we will analyze the *Wine* dataset, that corresponds to the chemical analysis of wines grown in the same region in Italy but derived from three different cultivars (three different types of wine; thus, three classes). This dataset is composed of 178 observations, described in terms of 13 constituents. Thus, the number of dimensions of the dataset is 13. Notice that the last feature takes on values several orders of magnitude higher than the others. Therefore, without further action, most of the variability in the dataset will be summarizable by only one direction. In such cases (and more generally when the features are measured in different units), it is best practice to first scale the data (divide columnwise by the standard deviations) before running the analysis. Visualize different combinations of features for the dataset. Then, project the data to the first 2 and first 3 principal directions. What do you observe?

3 Supervised Dimensionality Reduction and Classification with Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a method used in statistics, pattern recognition and machine learning to find a linear combination of features which characterizes or separates two or more classes of objects or events. The resulting combination can be used as a linear classifier or as a dimensionality reduction method. More precisely, one way to view a linear classification model is in terms of dimensionality reduction. Let's consider the case of two classes, $K = 2$, and suppose we take the d-dimensional input vector $\mathbf{x}$ and project it down to one dimension using $y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$. Then, setting a threshold on $y(\mathbf{x})$, we can classify $\mathbf{x}$ to the first class if $y(\mathbf{x}) \geq 0$, and to class two otherwise. In general, the projection onto one dimension leads to a considerable loss of information, and classes that are well separated in the original d-dimensional space may become strongly overlapping in one dimension. However, by adjusting the components of the weight vector $\mathbf{w}$, we can select a projection that maximizes the class separation.

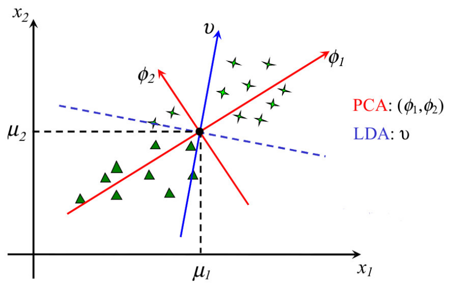

Figure 1: LDA vs. PCA.

LDA is also closely related to Principal Component Analysis (PCA). Both methods look for linear combinations of variables which best explain the data. LDA explicitly attempts to model the difference between the classes of data, while PCA does not take into account any difference in class. Figure 1 illustrates the difference between PCA and LDA. In PCA we process the data as a whole and do not consider any division into classes. The axes are optimal for representing the data as they indicate where the maximum variation actually lies. The LDA axis is optimal for distinguishing between the different classes. In general the number of axes that can be computed by the LDA method is one less than the number of classes of the problem. In the Figure 1, the ϕ_1 axis is less effective for separating the classes than the LDA axis v. The ϕ_2 axis is clearly useless for classification.

LDA computes an optimal transformation (projection) by minimizing the within-class distance and maximizing the between-class distance simultaneously, thus achieving maximum class discrimination. The optimal transformation in LDA can be readily computed by applying an eigendecomposition on the so-called scatter matrices.

Next we describe the process of computing the LDA axes in the general case, where the number of classes is $K > 2$. The classes are denoted as $C_1, C_2, \ldots, C_K$. Let $\mathbf{x} \in \mathbb{R}^d$ be an input vector (i.e., instance of our dataset), and let n be the total number of instances (i.e., the input matrix $\mathbf{X}$ has dimensions $n \times d$). We introduce $d' > 1$ linear features $y_k = \mathbf{w}^T\mathbf{x}$, where $k = 1, 2, \ldots, d'$. These features can be grouped to form a vector $\mathbf{y}$. In a similar way, the weight vectors $\mathbf{w}_k$ can be considered to be the columns of a matrix $\mathbf{W}$:

$$\mathbf{y} = \mathbf{W}^T\mathbf{x}. \tag{1}$$

The goal is to find matrix $\mathbf{W}$. The first step of LDA is to find two scatter matrices, referred to as *between class* and *within class* scatter matrices. The within class matrix $\mathbf{S}_w$ of size $d \times d$ is defined as the within-class *covariance matrix* and is considered for all the K classes in the dataset as follows:

$$\mathbf{S}_w = \sum_{j=1}^{K} \sum_{\mathbf{x} \in C_j} (\mathbf{x} - \mathbf{m}_j)(\mathbf{x} - \mathbf{m}_j)^T, \tag{2}$$

where $\mathbf{m}_j = \frac{1}{n_j} \sum_{\mathbf{x} \in C_j} \mathbf{x}, j = 1, , 2 \ldots, K$ is the mean vector (centroid) of the j-th class (n_j is the number of instances in class j). Note that the outer sum is for the different classes, while the inner sum is for the instances of each class and is equal to the covariance matrix per class.

The between class scatter matrix $\mathbf{S}_b$ of size $d \times d$ is defined as follows:

$$\mathbf{S}_b = \sum_{j=1}^{K} n_j (\mathbf{m}_j - \mathbf{m})(\mathbf{m}_j - \mathbf{m})^T, \tag{3}$$

where $\mathbf{m} = \frac{1}{n} \sum_{\mathbf{x} \in \mathbf{X}} \mathbf{x}$ is the mean of the total data. A different way to see the between class scatter matrix is by considering the total covariance matrix of the data, defined as:

$$\mathbf{S}_t = \sum_{\mathbf{x} \in \mathbf{X}} (\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^T, \tag{4}$$

where we sum up over all data instances. The total covariance $\mathbf{S}_t$ can be decomposed into the sum of the within class covariance matrix, plus the between class covariance matrix as $\mathbf{S}_t = \mathbf{S}_w + \mathbf{S}_b$.

The main objective of LDA is to find a projection matrix $\mathbf{W}$ that minimizes the within class separability while simultaneously maximizing the between class separability. This forms the following optimization problem:

$$\mathbf{W}_{LDA} = \arg\max_{\mathbf{W}} \frac{|\mathbf{W}^T \mathbf{S}_b \mathbf{W}|}{|\mathbf{W}^T \mathbf{S}_w \mathbf{W}|}. \tag{5}$$

This ratio is known as Fisher's criterion. To get an intuition of what is means, note that the determinant of the covariance matrix tells us how much variance a class has. For example, for the co-variance matrix in the PCA (diagonal) projection, the value of the determinant is just the product of the diagonal elements which are the individual variable variances. The determinant has the same value under any orthonormal projection. So Fisher's criterion tries to find the projection that maximizes the variance of the class means and minimizes the variance of the individual classes.

It is known that the solution to the optimization problem in Eq. (5) can be obtained by solving the following generalized eigenvalue problem:

$$\mathbf{S}_b \mathbf{w}_j = \lambda \mathbf{S}_w \mathbf{w}_j \Leftrightarrow \mathbf{S}_w^{-1} \mathbf{S}_b \mathbf{w}_j = \lambda \mathbf{w}_j, \quad j = 1, \ldots, K-1, \tag{6}$$

where $\mathbf{w}_j$ are vectors that correspond to columns of the projection matrix $\mathbf{W}$, i.e., $\mathbf{W} = [\mathbf{w}_1 | \mathbf{w}_2 | \ldots | \mathbf{w}_{K-1}]$. Therefore, the projection matrix $\mathbf{W}$ can be obtained by the eigenvectors that correspond to the $K-1$ largest eigenvalues of the $\mathbf{S}_w^{-1} \mathbf{S}_b$ matrix. Additionally, the new representation of the data $\mathbf{X}_{LDA}$ can be obtained by projecting the data $\mathbf{X}$ to the new space defined by $\mathbf{W}$ (i.e., dot product of $\mathbf{X}$ and $\mathbf{W}$).

Algorithm 1 provides the pseudocode of the LDA method.

Algorithm 1 Linear Discriminant Analysis (LDA)

```
Input: Training data $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\}$, where $\mathbf{x_i} = (x_1, x_2, \ldots, x_d)$, $i = 1, \ldots, n$ and their class
    labels $\mathbf{Y}$
Output: Projected data $\mathbf{X}_{LDA}$ (of dimension $n \times (K-1)$)

1: Compute the within class scatter matrix as shown in Eq. 2
2: Compute the between class scatter matrix as shown in Eq. 3
3: Compute matrix $\mathbf{W}$ solving the eigenvalue problem of Eq. 6 and getting the eigenvectors that
   correspond to the $K-1$ largest eigenvalues ($K$ is the number of classes)
4: Project the data to the new space defined by $\mathbf{W}$ and get $\mathbf{X}_{LDA}$
5: Project also the mean vectors of each class $\mathbf{m}_j, j = 1, \ldots, K$ to the new space (this will be used later
   for classification) and get $\mathbf{m}_j^{LDA}, j = 1, \ldots, K$
```

How to perform classification?

Algorithm 1 provides the dimensionality reduction obtained by the LDA method. To perform classification of a new data instance $\mathbf{x}$ to one of the possible classes $C_j, j = 1, \ldots, K$, a similar approach is followed. The new instance is projected into the new space defined by matrix $\mathbf{W}$, and then it is assigned to the closest class as defined by the Euclidean distance of $\mathbf{x}$ to the centroid vectors (mean) $\mathbf{m}_j^{LDA}$ of each class C_j (as computed at step 5 of Algorithm 1).

3.1 Pipeline of the Task

Next we briefly describe the pipeline that will be followed in this part of the lab. The goal is to implement LDA and apply it on the 178×13 *Wine* dataset (three different types of wine; thus, three classes) – also used for PCA.

The pipeline of the task is in the `LDA/main.py` Python script. Initially we load the data and randomly split it into training and testing sets.

```
# Load data
my_data = np.genfromtxt('wine_data.csv', delimiter=',')
np.random.shuffle(my_data)
trainingData = my_data[:100,1:] # training data
trainingLabels = my_data[:100,0] # class labels of training data

testData = my_data[101:,1:] # training data
testLabels = my_data[101:,0] # class labels of training data
```

After that, we apply LDA to project the data to the new space. This part is implemented by the `my_LDA()` function in the `LDA/my_LDA.py` file that performs the tasks described in Algorithm 1. The function returns the projection matrix $\mathbf{W}$, the centroid vectors $\mathbf{m}_j^{LDA}, j = 1, \ldots, K$ of each class and the projected data $\mathbf{X}_{LDA}$.

```
# Training the LDA classifier
W, projected_centroid, X_lda = my_LDA(trainingData, trainingLabels)
```

Then, we use the projection matrix $\mathbf{W}$ and the centroid vectors of each class to perform classification of the test data. The process followed is the one described above and implemented in the `predict()` function in the `LDA/predict.py`. The class labels of the new instances are returned (note that we increment their value by one since the original classes are $1, 2$ and 3, while the returned values are $0, 1, 2$).

```
# Perform predictions for the test data
```

```
predictedLabels = predict(testImages, projected_centroid, W)
predictedLabels = predictedLabels+1
```

3.2 Tasks to be Done in the Lab

The goal of this part is to implement the functions `my_LDA()` and `predict()`, as described above. Then, the accuracy of the LDA classifier can be computed for the *Wine* dataset.

References

[1] Lindsay I. Smith. “A Tutorial on Principal Components Analysis”. Cornell University, USA 51 (2002): 52.

[2] Jonathon Shlens. “A Tutorial on Principal Component Analysis”. arXiv preprint arXiv:1404.1100 (2014).

[3] Christopher M. Bishop. “Pattern Recognition and Machine Learning”. Vol. 1. New York: Springer, 2006.

[4] Tom M. Mitchell. “Machine learning”. Burr Ridge, IL: McGraw Hill 45, 1997.

[5] Jure Leskovec, Anand Rajaraman, and Jeff Ullman. “Mining of Massive Datasets”. Cambridge University Press, 2014.

[6] Michael E. Wall, Andreas Rechtsteiner, and Luis M. Rocha. “Singular Value Decomposition and Principal Component Analysis: A Practical Approach to Microarray Data Analysis”. Springer US, 2003. 91-109.