Dimensionality Reduction and Data Fusion

Narges Mehran
narges.mehran@aau.at

Peer Seminar
Lecturers: Prof. Hellwagner and Prof. Hitz
28.06.2019
Key questions

• How to transform a high-dimensional data set into a small set?

• Which methods can be used to combine and reduce the high-dimensional data?

• Are these methods among the learning-based schemes?

• Do I need some pre-processing techniques before data fusion exploitation?
Data fusion

- Humans, who rely on their senses as the vision, smell, taste, voice and physical movement, are a principal example of data-fusion system.

- A major tool is to remove the dependencies among the collected data.

- In computer science, it is also required to combine various data sets into a unified (fused) data set which includes all data points.
Data fusion (cont.)

• To store, analyze and summarize the vast amounts of generated data, one may reduce the dimension of data by dimensionality-reduction data fusion.

• This transformation finds a subspace whose vectors are a combination of the old subspace and projects a \( t \)-dimensional space onto an \( k \)-dimensional subspace of the original features, where \( k << t \).

• The raw data sets collected from our implementations are not available before running the algorithm.
Principal component analysis

• PCA is mathematically defined as an orthogonal linear transformation that transforms the data to a new coordinate system,

• PCA
  o helps to find relevant structure in data,
  o helps to throw away things that won’t matter

• The projection of the data comes to lie on the new coordinate system,
  o the greatest variance on the first coordinate (called the first principal component),
  o the second greatest variance on the second coordinate,
  o and so on
Transforming the data set to the new space

Data in the old space

Data in new space after PCA Transformation
Projection of the data

Data in the old space

Projection on the line with higher variance

Projection on the line with lower variance
PCA Calculation

- Vectors of data $X$: $X_1, X_2, X_3, \ldots, X_J$
  - Dimension of every vector of data $X$: $I \times I$

- Matrix dimension: $I \times J \rightarrow I$ is the number of samples, $J$ shows the attribute for every sample.

- First step is to calculate the average of samples and normalizing them:

$$
\mu_j = \frac{1}{I} \left( \sum_{i=1}^{I} X_{ij} \right) \quad \Rightarrow \quad X = [X_1 - \mu_1, X_2 - \mu_2, \ldots, X_J - \mu_J,]$

\[
\mu_j = \frac{1}{I} \left( \sum_{i=1}^{I} X_{ij} \right) \quad \Rightarrow \quad X = [X_1 - \mu_1, X_2 - \mu_2, \ldots, X_J - \mu_J,]
\]
PCA Calculation (cont.)

• Second step is to calculate the principal components of the new subspace:
  1) Calculating the co-variance matrix:
     • \( C = \frac{1}{I} (X^T X) \)
     • \( V_i C = \lambda_i C \)
  2) Calculating by the singular value decomposition: \( \text{SVD} (X) = [U, \Sigma, V] = U \Sigma V^T \)
     • \( \Sigma \) is the diagonal matrix
     • \( U \) and \( V \) are unitary matrices

• Third step is choosing a few number of eigenvectors of \( V \) and projecting \( X \) on this new subspace.
  • \( X_k = X . V_k \)
Randomized-SVD algorithm

- For reducing the size of information and combining features with different qualities, Truncated-SVD is exploited; Truncated-SVD has the ability to extract only the most important information from the data matrix by using just the first several components estimated from the original matrix of data set.

- Randomized-SVD implements a type of Truncated Singular Value Decomposition (Truncated-SVD) that only computes the $k$-largest singular values with a randomized algorithm, where $k$ is a user-specified parameter.

- Randomized-SVD is similar to PCA, but differs in that it works on sample matrix $X$ directly instead of their covariance matrices. When the column-wise (per-feature) means of $X$ is subtracted from the feature values, Randomized-SVD on the resulting matrix is equivalent to PCA.
Randomized-SVD algorithm (cont.)

• Given an $m \times n$ matrix $X$, a target number $k$ of singular vectors, this procedure computes an approximate factorization $UV$, where $U$ and $V$ are orthonormal matrices whose columns are eigenvectors of $X X^*$ and $X^* X$ respectively, and is nonnegative and diagonal matrix containing the eigenvalues of $X$. $X^*$ in the diagonal being sorted in descending order.

• By considering the problem of finding the $k$ principal components of the SVD of an $m \times n$ input matrix, randomized algorithms involve $O(mn \log(k))$ floating-point operations (flops) in distinction to $O(mnk)$ for classical algorithms.

• Randomized-SVD can generate a structure from an unstructured input data matrix by using a subsampled random Fourier Transform (SRFT) and QR decomposition:
Randomized-SVD algorithm (cont.)

![Algorithm 1 Randomized-SVD’s Pseudo Code](image)

Reaching the fused data in a nutshell

- Using the Randomized SVD algorithm as a data fusion model as follows:

  \[ x = \frac{x - \text{mean}}{\text{std. dev}} \]

  Second, Randomized-SVD, applied to the training samples \( X \), produces a low-rank approximation \( X_k \)

  \[ X \approx X_k = U_k \cdot \Sigma_k \cdot V_k^T \]

  \( U_k \Sigma_k \) is a transformed training set with \( k \) features. To also transform the original set \( X \), we multiply it with \( V_k \) (the normalized eigenvectors of a new subspace)

  \[ X_{\text{fused}} = X \cdot V_k \]
A Data Fusion Diagram

• [https://scikit-learn.org/stable/](https://scikit-learn.org/stable/)

**Scikit-learn**, an open source library in Python, can be exploited for:

### Classification
- Identifying to which category an object belongs to.
- **Applications**: Spam detection, Image recognition.
- **Algorithms**: SVM, nearest neighbors, random forest, ...

### Regression
- Predicting a continuous-valued attribute associated with an object.
- **Applications**: Drug response, Stock prices.
- **Algorithms**: SVR, ridge regression, Lasso, ...

### Clustering
- Automatic grouping of similar objects into sets.
- **Applications**: Customer segmentation, Grouping experiment outcomes.
- **Algorithms**: k-Means, spectral clustering, mean-shift, ...

### Dimensionality Reduction
- Reducing the number of random variables to consider.
- **Applications**: Visualization, Increased efficiency
- **Algorithms**: PCA, feature selection, non-negative matrix factorization.

### Model Selection
- Comparing, validating and choosing parameters and models.
- **Goal**: Improved accuracy via parameter tuning
- **Modules**: grid search, cross validation, metrics.

### Preprocessing
- Feature extraction and normalization.
- **Application**: Transforming input data such as text for use with machine learning algorithms.
- **Modules**: preprocessing, feature extraction.

---

---
import numpy as np

class PCA:
    def __init__(self, n_components=None):
        self.n_components = n_components

    def fit_transform(self, X):
        X = array2d(X)
        n_samples, n_features = X.shape
        X -= self.mean
        U, S, V = linalg.svd(X, full_matrices=False)
        self.explained_variance_ = (S ** 2) / n_samples
        self.explained_variance_ratio_ = (self.explained_variance_ / self.explained_variance_.sum())
        if self.whiten:
            self.components_ = V / S[:, np.newaxis] * np.sqrt(n_samples)
        else:
            self.components_ = V

        Cov_ = np.cov(X.T)
        print("nCovariance matrix is:")
        print(Cov_

        if self.n_components == 'mle':
            if n_samples < n_features:
                raise ValueError("n_components='mle' is only supported "
                                "if n_samples > n_features")
            self.n_components = int(np.round(n_features))
        elif (self.n_components is not None
              and 0 < self.n_components
              and self.n_components < n_features):
            # number of components for which the cumulated explained variance
            # percentage is superior to the desired threshold
            ratio_cumsum = self.explained_variance_ratio_.cumsum()
            self.n_components = np.sum(ratio_cumsum < self.n_components) + 1

        if self.n_components is not None:
            self.components_ = self.components_[:, :self.n_components]
            self.explained_variance_ = \
            self.explained_variance_[:self.n_components]
            self.explained_variance_ratio_ = \
            self.explained_variance_ratio_[:self.n_components]

        return (U, S, V)
An implementation of a randomized algorithm for principal component analysis

A. Szlam et al. 2014

```python
random_state = check_random_state(random_state)
n_random = n_components + n_oversamples
n_samples, n_features = M.shape

if n_iter == 'auto':
    # Checks if the number of iterations is explicitly specified
    # Adjust n_iter. 7 was found a good compromise for PCA. See $5299
    n_iter = 7 if n_components < 0.1 * min(M.shape) else 4

if transpose == 'auto':
    transpose = n_samples < n_features
if transpose:
    # this implementation is a bit faster with smaller shape[1]
    M = M.T

Q = randomized_range_finder(M, n_random, n_iter,
                             power_iteration_normalizer, random_state)

# project M to the (k + p) dimensional space using the basis vectors
B = safe_sparse_dot(Q.T, M)

# compute the SVD on the thin matrix: (k + p) wide
Uhat, s, V = linalg.svd(B, full_matrices=False)

del B
U = np.dot(Q, Uhat)

if flip_sign:
    if not transpose:
        U, V = svd_flip(U, V)
    else:
        # In case of transpose u_based_decision=False
        # to actually flip based on u and not v.
        U, V = svd_flip(U, V, u_based_decision=False)

if transpose:
    # transpose back the results according to the input convention
    return V[:,n_components, :], s[n_components], U[:,n_components].T
else:
    return U[:, n_components], s[n_components], V[:, n_components, :]
```
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD

X_std = StandardScaler().fit_transform(X)

sklearn_X = TruncatedSVD(n_components=1)
sklearn_transf = sklearn_X.fit(X_std)
Ref.

• http://cs229.stanford.edu/notes/cs229-notes10.pdf
Thank you