OST
Ostschweizer
Fachhochschule

Lab 11 (solution)

FTP MachLe MSE
HS 2023

Dimensionality Reduction

MSE MachLe
WÜRC

Lernziele/Kompetenzen

- You know the main motivations and applications and drawbacks of *dimensionality reduction* of a high dimensional dataset.
- You can explain by an example what is meant by the term *curse of dimensionality* and the consequences of this fact.
- You can apply *Principal Component Analysis* (**PCA**) to high dimensional datasets and explain how PCA works. You know different versions of PCA like incremental PCA, vanilla PCA, randomized PCA and kernel PCA. You can explain the differences between these methods and know to apply them accordingly using `scikit learn`.
- You know the *kernel trick*, it's advantages and when it can be applied to a given technique. You can list at least four different kernels $k(x, x')$ that are frequently used in Machine Learning.
- You know the four axioms that define a *metrics* and can name four different metrices that are commonly used in Machine Learning.
- You know the manifold hypothesis and different manifold techniques like **MDS** (*Multidimensional scaling*), **LLE** (*local linear embedding*), **t-SNE** (*t-distributed stochastic neighbor embedding*) and **Isomap** (*Isometric mapping*) and can apply them to high dimensional datasets in `scikit learn`.
- You can successfully apply dimensionality reduction techniques for the *visualization* of high dimensional datasets, for *noise reduction* in datasets and for the *elimination of meaningless features used for classification*.

---

1. Applications of dimensionality reduction techniques [A, I]

a) The main motivations for dimensionality reduction are:

- To speed up a subsequent training algorithm (in some cases it may even remove noise and redundant features, making the training algorithm perform better).
- To visualize the data and gain insights on the most important features.
- Simply to save space (compression).
- To reduce noise.

b) The main applications are:

- removal of noise and redundant features for classification tasks
- visualization of high dimensional data
- data compression
- noise reduction, removal

c) The main drawbacks are:

- Some information is lost, possibly degrading the performance of subsequent training algorithms.
- It can be computationally intensive.
- It adds some complexity to your Machine Learning pipelines.
- Transformed features are often hard to interpret.

2. Curse of Dimensionality [A,I]

The *curse of dimensionality* refers to the fact that many problems that do not exist in low-dimensional space arise in high-dimensional space. In Machine Learning, one common manifestation is the fact that randomly sampled highdimensional vectors are generally very sparse, increasing the risk of overfitting and making it very difficult to identify patterns in the data without having plenty of training data.

3. General questions [A,II]

a) Once a dataset's dimensionality has been reduced using one of the algorithms we discussed, it is almost always impossible to perfectly reverse the operation, because some information gets lost during dimensionality reduction. Moreover, while some algorithms (such as PCA) have a simple reverse transformation procedure that can reconstruct a dataset relatively similar to the original, other algorithms (such as t-SNE) do not.

b) For dimensionality reduction of high dimensional, nonlinear datasets, mainfold methods that use the local structure of the dataset should be used:

- *Kernel PCA* is generally well suited in reducing the dimensionality of high dimensional, nonlinear datasets. By applying the kernel trick, a nonlinear mapping is applied to the input data, actually increasing the dimensionality even more. The kernel however can be evaluated in dataspace. The problem complexity is given by the number of data points.
- *Local Linear Embedding* (LLE) reduces dimensionality while trying to preserve the distances between close instances only.
- *Isomap* creates a graph by connecting each instance to its nearest neighbors, then reduces dimensionality while trying to preserve the geodesic distances between the instances.
- *t-Distributed Stochastic Neighbor Embedding* (t-SNE) reduces dimensionality while trying to keep similar instances close and dissimilar instances apart. It is mostly used for visualization, in particular to visualize clusters of instances in high-dimensional space (e.g., to visualize the MNIST images in 2D).

- *Linear Discriminant Analysis* (LDA) is actually a classification algorithm. During training it learns the most discriminative axes between the classes. These axes can be used to define a hyperplane onto which to project the data. The projection will keep classes as far apart as possible, so LDA is a good technique to reduce dimensionality before running another classification algorithm such as an SVM classifier.

c) Intuitively, a dimensionality reduction algorithm performs well if it eliminates a lot of dimensions from the dataset without losing too much information. One way to measure this is to apply the reverse transformation and measure the reconstruction error. However, not all dimensionality reduction algorithms provide a reverse transformation.

Alternatively, if you are using dimensionality reduction as a preprocessing step before another Machine Learning algorithm (e.g., a Random Forest classifier), then you can simply measure the performance of that second algorithm; if dimensionality reduction did not lose too much information, then the algorithm should perform just as well as when using the original dataset.

d) It can absolutely make sense to chain two different dimensionality reduction algorithms. A common example is using PCA to quickly get rid of a large number of useless dimensions, then applying another much slower dimensionality reduction algorithm, such as LLE. This two-step approach will likely yield the same performance as using LLE only, but in a fraction of the time.

e) In which cases would you use incremental PCA, randomized PCA or kernel PCA?

- *Regular PCA* is the default, but it works only if the dataset fits in memory.
- *Incremental PCA* is useful for large datasets that don't fit in memory, but it is slower than regular PCA, so if the dataset fits in memory you should prefer regular PCA. Incremental PCA is also useful for online tasks, when you need to apply PCA on the fly, every time a new instance arrives.
- *Randomized PCA* is useful when you want to considerably reduce dimensionality and the dataset fits in memory; in this case, it is much faster than regular PCA. Finally, Kernel PCA is useful for nonlinear datasets.

4. PCA and scaling importance [A,II]

The Jupyter notebook solution can be found on moodle:
`Lab10_A4_PCA_ScalingImportance_Wine_solution.ipynb`

5. Stochastic Neighbour Embedding [A,II]

The Jupyter notebook solution can be found on moodle:
`Lab11_A5_MNIST_tSNE_solution.ipynb`

6. Feature Engineering using PCA [A,II]

The Jupyter notebook solution can be found on moodle:
`Lab11_A6_MNIST_tSNE_solution.ipynb`

7. Kernels and the Kernel Trick [A, II]

a) The kernel trick avoids the explicit mapping that is needed to get linear learning algorithms to learn a nonlinear function or decision boundary. For all $\mathbf{x}$ and $\mathbf{x}'$ in the input space $\mathcal{X}$, certain functions $k(\mathbf{x}, \mathbf{x}')$ can be expressed as an inner product in another space $\mathcal{V}$. The function $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is often referred to as a *kernel* or a *kernel function*. The word 'kernel' is used in mathematics to denote a weighting function for a weighted sum or integral.

Certain problems in machine learning have additional structure than an arbitrary weighting function k. The computation is made much simpler if the kernel can be written in the form of a 'feature map' $\varphi\colon \mathcal{X} \to \mathcal{V}$ which satisfies

$$k(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle_{\mathcal{V}} \tag{1}$$

The key restriction is that $\langle \cdot, \cdot \rangle_{\mathcal{V}}$ must be a proper inner product. On the other hand, an explicit representation for φ is not necessary, as long as $\mathcal{V}$ is an inner product space. The alternative follows from MERCER's theorem: an implicitly defined function φ exists whenever the space $\mathcal{X}$ can be equipped with a suitable measure ensuring the function k satisfies MERCER's condition. MERCER's theorem is similar to a generalization of the result from linear algebra that associates an inner product to any positive-definite matrix.

Some algorithms that depend on arbitrary relationships in the native space $\mathcal{X}$ would, in fact, have a linear interpretation in a different setting: the range space of φ. The linear interpretation gives us insight about the algorithm. Furthermore, there is often no need to compute φ directly during computation, as is the case with *support vector machines*. Some cite this running time shortcut as the primary benefit. Researchers also use it to justify the meanings and properties of existing algorithms.

b) The most common kernels in Machine Learning are:

$$\begin{aligned}
\text{Gaussian}: &\quad k(\mathbf{x}, \mathbf{x}') = \exp\left(-\beta \cdot \|\mathbf{x} - \mathbf{x}'\|^2\right) \\
\text{Laplacian}: &\quad k(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \cdot |\mathbf{x} - \mathbf{x}'|_1\right) \\
\text{Polynomial}: &\quad k(\mathbf{x}, \mathbf{x}') = \left(1 + \mathbf{x} \cdot \mathbf{x}^T\right)^p \\
\text{Sigmoid}: &\quad k(\mathbf{x}, \mathbf{x}') = \tanh\left(\alpha \mathbf{x} \cdot \mathbf{x}^T + \delta\right)
\end{aligned}$$