CISC 271 Class 31

Nonlinear Separation – Embeddings and Gram Matrix

Texts: Hastie *et al.*, 2009 [6], pp. 547–550

Main Concepts:

- *Embed vectors in a high–dimensional space*
- *Kernel functions for embedded dot products*
- *Gram matrix from kernel function*

Sample Problem, Machine Inference: How can we use nonlinearly separable data?

In machine learning, binary classification problems do not always have a linearly separable structure. Data vectors that are not linearly separable can sometimes be mapped to linearly separable vectors by means of a non-linear transformation. Nonlinear transformations are common in the SVM literature, in part because the dual representation of the SVM conveniently represents and solves certain kinds of these transformations.

For this class, we will use a special notation for the row of a matrix. Our data are provided as a data matrix $A \in \mathbb{R}^{m \times n}$. We expect that each column of A contains values of a variable, so each column is a vector $\vec{a}_j \in \mathbb{R}^m$. The rows are observations, which we expect to be values of variables. a row is a "vector" in the sense that it is an ordered set of real numbers, but a row is not necessarily a member of a vector space.

For our purposes of data analysis, we will temporarily disregard this ditinction and we will cautiously perform computations on rows. We will write the rows of a matrix $A \in \mathbb{R}^{m \times n}$ as the partition

$$A \stackrel{\text{def}}{=} \begin{bmatrix} \underline{a}_1 \\ \underline{a}_2 \\ \vdots \\ \underline{a}_m \end{bmatrix} \quad \text{with} \quad \underline{a}_i \in \mathbb{R}^n \tag{31.1}$$

We will define the dot product of two rows, $\underline{u} \in \mathbb{R}^n$ and $\underline{v} \in \mathbb{R}^n$, as

$$\begin{aligned} \underline{u} \cdot \underline{v} &\stackrel{\text{def}}{=} \sum_{k=1}^{n} u_k v_k \\ &= \underline{u}\,\underline{v}^T \end{aligned} \tag{31.2}$$

31.1 Linear Separation Using an Embedding

The principal method for managing non-linearly separable data, which is commonly used in machine learning and data analytics, is to map each data vector to a higher-dimensional vector space. In many applications, data vectors that seem to be not linearly separable can be mapped to another vector space in which they *are* linearly separable.

For example, we might be provided with labeled data vectors that are in a 2D vector space. The next higher dimension is 3D. Some forms of mappings that are effective in practice are to copy the entries of $\underline{u}$ into a new size 3 vector, then to put some function of each data vector $\underline{u}$ into the third entry. These forms would map a 2D vector $\underline{u}$ to a 3D vector $\hat{u}$.

These are examples of *embeddings*, which are smooth maps of one dimension into a space of higher dimension. Although it is not exact, we can think of an embedding as being an "inverse projection": a projection maps a higher-dimensional vector space to a lower-dimensional vectors space and an embedding reverse this process.

Suppose we are given a 2D data set that has 24 observations with label +1 and 16 observations with label -1 in the picture-like data set that is illustrated in Figure 31.1. By inspection, we can determine that these sets are not linearly separable – but they do seem to be *non-linearly* separable.

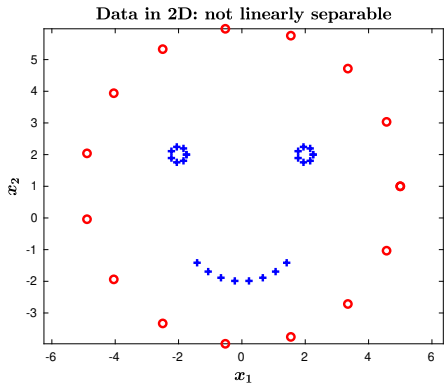

Figure 31.1: Pictorial data in 2D. The observations with label +1 are plotted as a plus sign; observations with label -1 are plotted as open circles.

Suppose that we try an embedding map that, for each row "vector" $\underline{u}_i$, "copies" the row into the first two entries of a 3D row "vector" $\hat{u}_i$ and then computes the squared length of $\underline{u}_i$ as the third entry of $\hat{u}_i$. The idea is that observations that are further from the original are embedded as 3-entry rows that have a larger third entry. This would be the map

$$\vec{u} \hookrightarrow \hat{u} \quad \text{which is} \quad \begin{bmatrix} u_1 & u_2 \end{bmatrix} \hookrightarrow \begin{bmatrix} u_1 & u_2 & u_1^2 + u_2^2 \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & \|\underline{u}\|^2 \end{bmatrix} \tag{31.3}$$

As we can see in Figure 31.2, these 3D data are linearly separable.

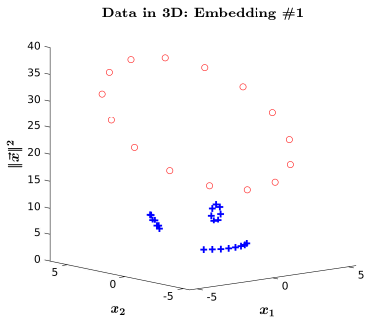

Figure 31.2: Pictorial data in 3D. The observations with label +1 are plotted as a plus sign; observations with label -1 are plotted as open circles. These data are separable by a 3D plane that, from this view, projects to a 2D horizontal line.

We can explore another embedding that will also linearly separate these data. Consider embedding each entry of $\underline{u}$ as the square of the entry of $\hat{u}$, and set $\hat{u}_3$ to be the product of the entries. For technical reasons, let us multiply the third entry of $\vec{u}_2$ by $\sqrt{2}$, so that the second embedding is

$$\underline{u} \hookrightarrow \hat{u} \quad \text{which is} \quad \begin{bmatrix} u_1 & u_2 \end{bmatrix} \hookrightarrow \begin{bmatrix} u_1^2 & u_2^2 & \sqrt{2}u_1u_2 \end{bmatrix} \tag{31.4}$$

The 2D original data, embedded as 3D data using Equation 31.4, are linearly separable as shown in Figure 31.2.

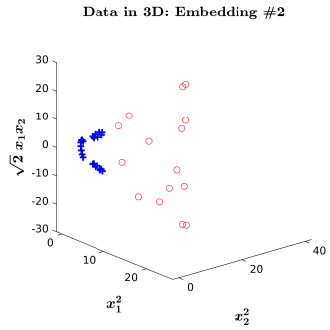

Figure 31.3: Pictorial data in 3D. The observations with label +1 are plotted as a plus sign; observations with label -1 are plotted as open circles. These data are separable by a 3D plane that, from this view, projects to a 2D vertical line.

Consider embedding a row "vector" $\underline{u} \in \mathbb{R}^n$ into a higher-dimensional vectors space, as $\hat{u} \in \mathbb{R}^p$. This would be a transformation of a data matrix $A \in \mathbb{R}^{m \times n}$ to a new matrix $\hat{A} \in \mathbb{R}^{m \times p}$. This requires more memory to store the matrix and, if we perform principal components analysis, might require the computation of a scatter matrix $\hat{S} \in \mathbb{R}^{p \times p}$ in size.

The embedding function of Equation 31.4 is a special case of a *polynomial* embedding, specifically as a quadratic function of the inner product. Recall, from basic algebra, that the number of terms of a polynomial grow combinatorially with the order of the polynomial. Here, the dimension p of the new vector space would grow combinatorially, as would the memory requirements. It is easy to imagine that there might be a combinatorial explosion of memory requirements for even a modest order of polynomial embedding.

31.2 Kernel Functions and the Gram Matrix

Let us carefully examine entry (i, j) of the matrix AA^T. This is the dot product of the i^{th} observation in A and the j^{th} observation in A. We can write the i^{th} row of A as $\underline{a}_i$ and the j^{th} row of A as $\underline{a}_j$.

Using Equation 31.2, the matrix AA^T can be computed. How can we compute the right-transpose product of the embedded matrix, which is $\hat{A}\hat{A}^T$? We must return to the definition of the embedding to better understand this computation.

We can write the embedding function $f : \mathbb{R}^n \to \mathbb{R}^p$, using row-like observations, as $\underline{f}(\underline{u})$. If the embedding function has certain properties, it is possible to perform the necessary computations for finding $\hat{A}\hat{A}^T$ actually embedding the data in a higher-dimensional space.

The insight for the *kernel trick* in PCA with a data matrix A is: the embedding does not need to be computed! In finding the entries of $\hat{A}\hat{A}^T$, only the *dot products* of the embedded vectors are needed. It is possible to compute the dot products without computing the embedding, for certain useful embeddings.

We can see that the matrix AA^T is symmetric and positive semidefinite. The entry (i, j) of this symmetric matrix is $\underline{a}_i \, \underline{a}_j^T$. If we embed the observation $\underline{a}_i$ in a higher-dimensional space as $\hat{a}_i$, and embed the observation $\underline{a}_j$ as $\hat{a}_j$, the dot product of Equation 31.2 would be

$$\hat{a}_i \, \hat{a}_j^T = \underline{f}(\underline{a}_i)[\underline{f}(\underline{a}_j)]^T \tag{31.5}$$

As defined in the extra notes for this class, a *kernel function* is a function that is symmetric and positive semidefinite. For an appropriate kernel function $\kappa(\cdot, \cdot)$ that is defined for the vector space $\mathbb{R}^n$, the mapping in Equation 31.5 can be avoided and the dot product can be computed directly.

For example, the embedding $\underline{f}(\cdot)$ in Equation 31.3 has a kernel function that is

$$\kappa(\underline{u}, \underline{v}) = \underline{u}\,\underline{v}^T + (\underline{u}\,\underline{v}^T)^2 \tag{31.6}$$

The embedding embedding $\underline{f}(\cdot)$ in Equation 31.4 has a kernel function

$$\kappa(\underline{u}, \underline{v}) = (\underline{u}\,\underline{v}^T)^2 \tag{31.7}$$

The kernel function in Equation 31.7 is a *second-order polynomial* of the dot product of two vectors. Computing a high-order polynomial kernel is not complicated, but storing the full embedding may result in a combinatorial explosion of memory use.

For any data matrix A, and any kernel function $\kappa(\cdot, \cdot)$ defined on the observations that are the rows of A, the *Gram matrix* is the matrix K that has the entries

$$K_{ij} \stackrel{\text{def}}{=} \kappa(\underline{a}_i, \underline{a}_j) \tag{31.8}$$

As described in the extra notes, and proved in many textbooks on matrix analysis, the Gram matrix K of a design matrix is symmetric and positive semidefinite.

31.3 Some Kernel Functions for Row Spaces

The kernel function of Equation 31.6 is a specialized instance that is not widely used. Here are some examples of commonly used kernel functions and their interpretations. Here, we will describe the kernel function for a pair of rows as $\kappa(\underline{u}, \underline{v})$ and understand that we can also compute the kernel function for vectors $\kappa(\vec{u}, \vec{v})$.

Linear Kernel: The dot product of rows is the basic kernel function

$$\kappa(\underline{u}, \underline{v}) = \underline{u}\,\underline{v}^T \tag{31.9}$$

Polynomial Kernel: The dot product of vectors can be added to a constant c and raised to a power p, which is kernel function

$$\kappa(\underline{u}, \underline{v}) = (\underline{u}\,\underline{v}^T + c)^l \tag{31.10}$$

Equation 31.10 is usually presented for $c = 1$. An alternative that is sometimes presented is $c = 0$, which is an *exact power* kernel function.

Gaussian Kernel: The square of the norm of the distance between vectors can be used in the unscaled normal distribution function e^{-x^2}. In PCA literature, this is usually presented with a scaling term γ as the kernel function

$$\kappa(\underline{u}, \underline{v}) = \exp\left(-\gamma \|\underline{u} - \underline{v}\|^2\right) \tag{31.11}$$

The term γ is a crucial hyper-parameter in kernel PCA. A small γ causes the Gram matrix to approach the identity matrix. A large γ causes the entries in the Gram matrix to approach a constant value. The variance can be supplied by the user and can sometimes be estimated from the data matrix A.

In `MATLAB`, the Gaussian kernel is

$$\kappa(\underline{u}, \underline{v}) = \exp\left(-\|\underline{u} - \underline{v}\|^2\right) \tag{31.12}$$

Laplacian kernel: The norm of the distance between vectors, rather than the square of the norm, can be used in an exponential function.

$$\kappa(\underline{u}, \underline{v}) = \exp\left(-\gamma \|\underline{u} - \underline{v}\|\right) \tag{31.13}$$

Sigmoid kernel: The inner product of vectors can be used as an argument in the hyperbolic tangent function that is common in neural networks. This must be done with care because not all of the sigmoid hyper-parameters produce a positive semidefinite Gram matrix from a full-rank design matrix X. The sigmoid kernel is

$$\kappa(\vec{u}, \vec{v}) = \tanh(\gamma \underline{u}\underline{v}^T + \beta) \tag{31.14}$$

A sufficient condition for Equation 31.14 to produce a positive semidefinite Gram matrix is that γ is positive and β is negative, which is $\gamma > 0$ and $\beta < 0$. The conditions are so sensitive that `MATLAB` does not currently offer this kernel as an option.

31.4 Extra Notes for The Gram Matrix and Kernel Functions

Definition: Gram matrix of a finite set of vectors

For any non-empty finite set $\mathbb{X} = \{\underline{x} : \underline{x} \in \mathbb{R}^n\}$ of cardinality m_X, any $m \in \mathbb{N}_{++} : m \leq m_X$, any $i \in \mathbb{N}_{++} : i \leq m$, any $j \in \mathbb{N}_{++} : j \leq m$, any $\underline{x}_i \in \mathbb{X}$, any $\underline{x}_j \in \mathbb{X}$, and any positive semidefinite symmetric function $\kappa : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$, the *Gram matrix* of the members $\{\underline{x}_i : 1 \leq i \leq m\}$ is defined as the matrix K that is

$$K_{ij} \stackrel{\text{def}}{=} \kappa(\underline{x}_i, \underline{x}_j) \tag{31.15}$$

Observation: Because the function $\kappa(\cdot, \cdot)$ is positive semidefinite symmetric, the Gram matrix of a set of members of $\mathbb{X}$ is symmetric and positive semidefinite.

In linear algebra, the set $\mathbb{X}$ is typically a vector space $\mathbb{R}^n$ and the kernel function κ is typically an inner product $\langle \cdot, \cdot \rangle$. In machine learning, the set $\mathbb{X}$ can be quite diverse, including strings of symbols in text analysis. The positive semidefinite symmetric function κ may be a measure, especially a metric, on the set $\mathbb{X}$.

Definition: Kernel function

For any non-empty set $\mathbb{X}$, any $n \in \mathbb{N}_+$, a symmetric function $\kappa : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ is a *positive semidefinite kernel function* is defined as:

a function for which the Gram matrix is positive semidefinite, or

$$K \succeq 0 \tag{31.16}$$

Observation: In linear algebra, an inner product on $\mathbb{R}^n$ is a kernel function. For a vector space and a kernel that is an inner product, if the vectors $\vec{x}_i$ are linearly independent then the Gram matrix is positive semidefinite.

End of Extra Notes