Lecture 12

Support Vector Machines and Singular Value Decomposition

12.1 Support Vector Machines

Support Vector Machines, or SVMs, have emerged as an increasingly popular algorithm for effectively learning a method to categorize objects based on a set of properties. SVMs were originally introduced by Vapnik [97], although his work was in Russian and was not translated to English for several years. Recently, more attention has been paid to them, and there are several excellent books and tutorials [14, 51, 98, 99].

We first develop the notion of *learning* a categorization function, then describe the SVM algorithm in more detail, and finally provide several examples of its use.

12.1.1 Learning Models

In traditional programming, a function is given a value x as input, and computes a value $y = h(x)$ as its output. In this case, the programmer has an intimate knowledge of the function $h(x)$, which he explicitly defines in his code.

An alternative approach involves *learning* the function $h(x)$. There are two methods to generate this function. In *unsupervised learning*, the program is given a set of objects $\mathbf{x}$ and is asked to privide a function $h(\mathbf{x}$ to categorize them without any *a priori* knowledge about the objects.

Support Vector Machines, on the other hand, represent one approach to *supervised learning*. In supervised learning, the program is given a *training set* of pairs $(\mathbf{x}, \mathbf{y})$, where $\mathbf{x}$ are the objects being classified and $\mathbf{y}$ are a matching set of *categories*, one for each x_i. For example, an SVM attempting to categorize handwriting samples might be given a training set which maps several samples to the correct character. From these samples, the supervised learner induces the function $y = h(x)$ which attempts to return the correct category for each sample. In the future, new samples may be categorized by feeding them into the pre-learned $h(x)$.

In learning problems, it is important to differentiate between the types of possible output for the function $y = h(x)$. There are three main types:

Binary Classification $h(x) = y \in \pm 1$ (The learner classifies objects into one of two groups.)

Discrete Classification $h(x) = y \in \{1, 2, 3, ..., n\}$ (The learner classifies objects into one of n groups.)

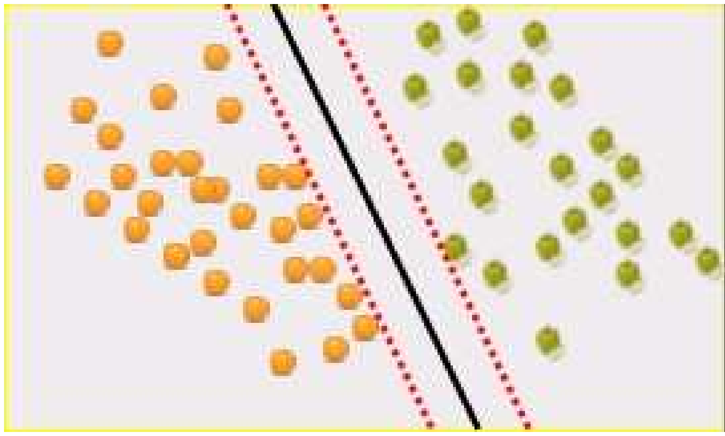

Figure 12.1: An example of a set of linearly separable objects.

Continuous Classification $h(x) = y \in \Re^n$ (The learner classifies objects in the space of n-dimensional real numbers.)

As an example, to learn a linear regression, x could be a collection of data points. In this case, the $y = h(x)$ that is learned is the best fit line through the data points.

All learning methods must also be aware of *overspecifying* the learned function. Given a set of training examples, it is often inappropriate to learn a function that fits these examples *too* well, as the examples often include noise. An overspecified function will often do a poor job of classifying new data.

12.1.2 Developing SVMs

A Support Vector Machine attempts to find a linear classification of a set of data. For dimension $d > 2$ this classification is represented by a *separating hyperplane*. An example of a separating hyperplane is shown in Figure 12.1. Not that the points in Figure 12.1 are *linearly separable*, that is, there exists a hyperplane in $\Re^2$ such that all oranges are on one side of the hyperplane and all apples are on the other.

In the case of a linearly separable training set, the *perceptron* model [80] is useful for finding a linear classifier. In this case, we wish to solve the equation

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

subject to the constraints that $h(\mathbf{x_1}) \leq -1$ and $h(\mathbf{x_2}) \geq +1$, where $\mathbf{x_1}$ are the objects in category 1 and $\mathbf{x_2}$ are the objects in category 2. For example, in Figure 12.1, we would wish to satisfy:

$$\begin{aligned} h(orange) &\leq -1 \\ h(apple) &\geq +1 \end{aligned}$$

The SVM method for finding the *best* separating hyperplane is to solve the following linear program:

$$\min_{w,b} \frac{\|\mathbf{w}\|^2}{2} \quad \text{subject to} \tag{12.1}$$

$$y_i(w_i^T x_i + b) \geq 1, i = 1, ..., n \tag{12.2}$$

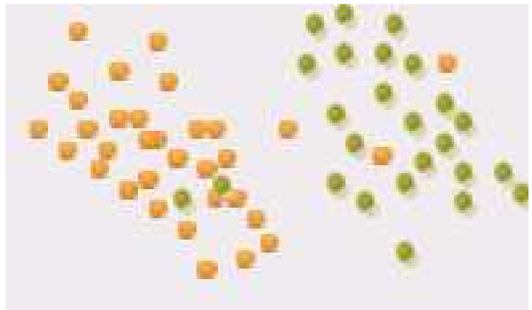

Figure 12.2: An example of a set of objects which are not linearly separable.

This method works well for training sets which are linearly separable. However, there are also many cases where the training set is *not* linearly separable. An example is shown in Figure 12.2. In this case, it is impossible to find a separating hyperplane between the apples and oranges.

There are several methods for dealing with this case. One method is to add *slack* variables, ε_i to the linear program:

$$\min_{w,b,\varepsilon} \frac{\|\mathbf{w}\|^2}{2} + c\sum_i \varepsilon_i \qquad \text{subject to}$$
$$y_i(w_i^T x_i + b) \geq 1 - \varepsilon_i, i = 1, ..., n$$

Slack variables allow some of the x_i to "move" in space to "slip" onto the correct side of the separating hyperplane. For instance, in Figure 12.2, the apples on the left side of the figure could have associated ε_i which allow them to move to the right and into the correct category.

Another approach is to non-linearly distort space around the training set using a function $\Phi(x_i)$:

$$\min_{w,b} \frac{\|\mathbf{w}\|^2}{2} \qquad \text{subject to}$$
$$y_i(w_i^T \Phi(x_i) + b) \geq 1, i = 1, ..., n$$

In many cases, this distortion moves the objects into a configuration that is more easily separated by a hyperplane. As mentioned above, one must be careful not to overspecify $\Phi(x_i)$, as it could create a function that is unable to cope easily with new data.

Another way to approach this problem is through the dual of the linear program shown in Equations (12.1) and (12.2) above. If we consider those equations to be the primal, the dual is:

$$\max_{\alpha} \frac{\alpha^{\mathbf{T}}\mathbf{1} - \alpha^{\mathbf{T}}\mathbf{H}\alpha}{2} \qquad \text{subject to}$$
$$\mathbf{y}^{\mathbf{T}}\alpha = 0$$
$$\alpha \geq 0$$

Note that we have introduced *Lagrange Multipliers* α_i for the dual problem. At optimality, we have

$$w = \sum_i y_i \alpha_i x_i$$

This implies that we may find a separating hyperplane using

$$H_{ij} = \mathbf{y_i}(\mathbf{x_i^T}\mathbf{x_j})\mathbf{y_j}$$

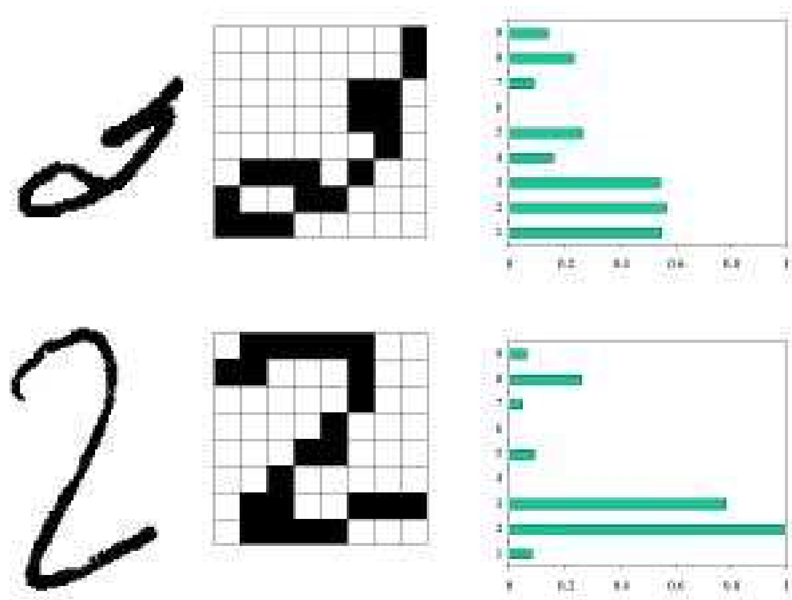

Figure 12.3: An example of classifying handwritten numerals. The graphs on the right show the probabilities of each sample being classified as each number, 0-9.

This dual problem also applies to the slack variable version using the constraints:

$$\mathbf{y}^{\mathbf{T}}\alpha = 0$$
$$\mathbf{c} \geq \alpha \geq 0$$

as well as the distorted space version:

$$H_{ij} = \mathbf{y_i}(\Phi(\mathbf{x_i})^T \Phi(\mathbf{x_j}))\mathbf{y_j}$$

12.1.3 Applications

The type of classification provided by Support Vector Machines is useful in many applications. For example, the post office must sort hundreds of thousands of hand-written envelopes every day. To aid in this process, they make extensive use of handwriting recognition software which uses SVMs to automatically decipher handwritten numerals. An example of this classification is shown in Figure 12.3.

Military uses for SVMs also abound. The ability to quickly and accurately classify objects in a noisy visual field is essential to many military operations. For instance, SVMs have been used to identify humans or artillery against the backdrop of a crowded forest of trees.

12.2 Singular Value Decomposition

Many methods have been given to decompose a matrix into more useful elements. Recently, one of the most popular has been the *singular value decomposition*, or SVD. This decomposition has been known since the early 19th century [94].

The SVD has become popular for several reasons. First, it is stable (that is, small perturbations in the input $\mathbf{A}$ result in small perturbations in the singular matrix $\mathbf{\Sigma}$, and vice versa). Second, the

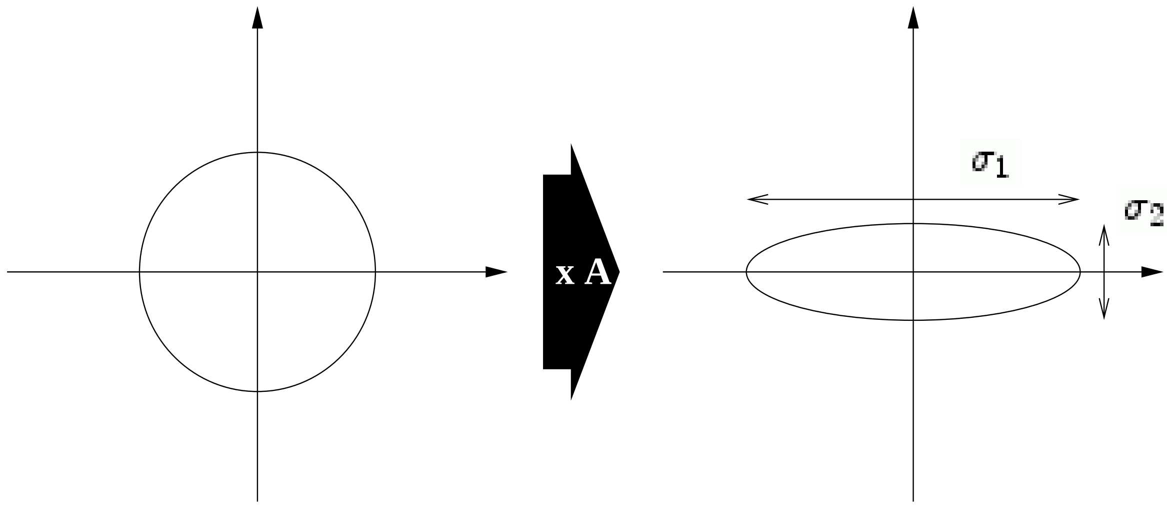

Figure 12.4: The deformation of a circle using a matrix $\mathbf{A}$.

singular values σ_i provide an easy way to approximate $\mathbf{A}$. Finally, there exist fast, stable algorithms to compute the SVD [42].

The SVD is defined by

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

Both $\mathbf{U}$ and $\mathbf{V}$ are orthagonal matrices, that is, $\mathbf{U^TU} = \mathbf{I}$ and $\mathbf{V^TV} = \mathbf{I}$. $\mathbf{\Sigma}$ is the *singular matrix*. It is non-zero except for the diagonals, which are labeled with σ_i:

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \dots & \\ & & & \sigma_n \end{pmatrix}$$

There are several interesting facts associated with the SVD of a matrix. First, the SVD is well-defined for any matrix $\mathbf{A}$ of size m x n, even for $m \neq n$. In physical space, if a matrix $\mathbf{A}$ is applied to a unit hypercircle in n dimensions, it deforms it into a hyperellipse. The diameters of the new hyperellipse are the singular values σ_i. An example is shown in Figure 12.4.

The singular values σ_i also have a close relation to its eigenvalues λ_i. The following table enumerates some of these relations:

Matrix Type	**Relationship**
Symmetric Positive Definite	$\sigma_i = \lambda_i$
Symmetric	$\sigma_i = \lvert\lambda_i\rvert$
General Case	$\sigma_i^2 = i^{th}$ eigenvalue of $\mathbf{A^TA}$

These relationships often make it much more useful as well as more efficient to utilize the singular value decomposition of a matrix rather than computing $\mathbf{A^TA}$, which is an intensive operation.

The SVD may also be used to *approximate* a matrix $\mathbf{A}$ with n singular values:

$$\mathbf{A} = \sum_{i=1}^{n} \sigma_i u_i v_i^T$$

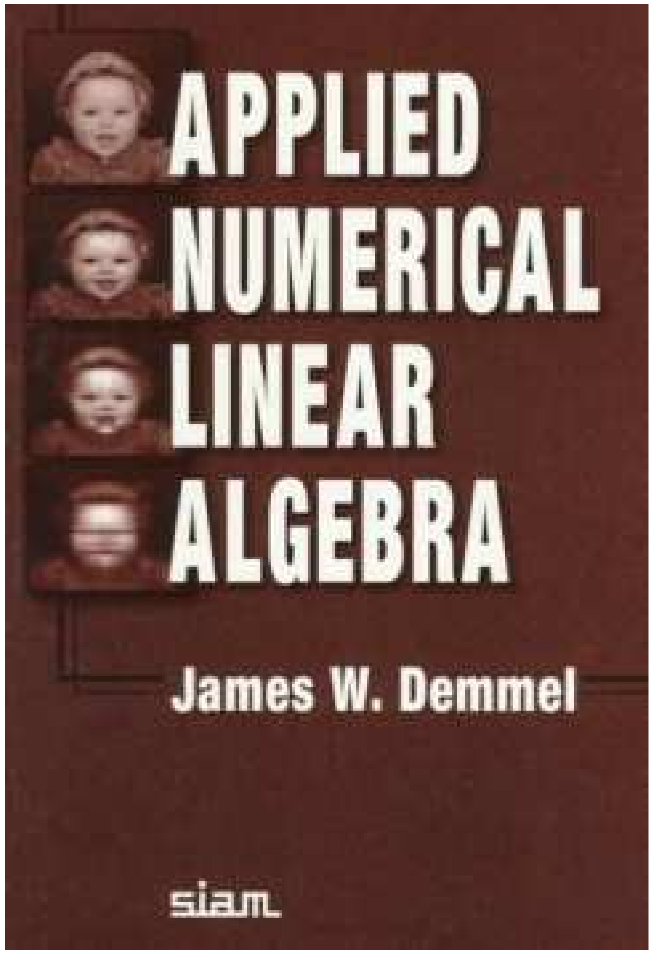

Figure 12.5: The approximation of an image using its SVD.

$$\approx \sum_{i=1}^{p} \sigma_i u_i v_i^T, p < n$$

where u_i is the i^{th} row of $\mathbf{U}$ and v_i^T is the i^{th} column of $\mathbf{V}$. This is is also known as the "rank-p approximation of $\mathbf{A}$ in the 2-norm or F-norm."

This approximation has an interesting application for image compression. By taking an image as a matrix of pixel values, we may find its SVD. The rank-p approximation of the image is a compression of the image. For example, James Demmel approximated an image of his daugher for the cover of his book *Applied Numerical Linear Algebra*, shown in Figure 12.5. Note that successive approximations create horizontal and vertical "streaks" in the image.

The following MATLAB code will load an image of a clown and display its rank-p approximation:

```
>> load clown;
>> image(X);
>> colormap(map);
>> [U,S,V] = svd(X);
>> p=1; image(U(:,1:p)*S(1:p,1:p)*V(:,1:p)');
```

The SVD may also be used to perform *latent semantic indexing*, or clustering of documents based on the words they contain. We build a matrix $\mathbf{A}$ which indexes the documents along one axis and the words along the other. $A_{ij} = 1$ if word j appears in document i, and 0 otherwise. By taking the SVD of $\mathbf{A}$, we can use the singular vectors to represent the "best" subset of documents for each cluster.

Finally, the SVD has an interesting application when using the FFT matrix for parallel computations. Taking the SVD of one-half of the FFT matrix results in singular values that are approximately one-half zeros. Similarly, taking the SVD of one-quarter of the FFT matrix results in singular values that are approximately one-quarter zeros. One can see this phenomenon with the following MATLAB code:

```
>> f = fft(eye(100));
>> g = f(1:50,51:100);
>> plot(svd(g),'*');
```

These near-zero values provide an opportunity for compression when communicating parts of the FFT matrix across processors [30].