SVM and PCA

Swapnoneel Kayal

25 May-31 May 2021

1 SVM : Support Vector Machines

The original SVM algorithm was invented by Vladimir N. Vapnik and the current standard incarnation (soft margin) was proposed by Corinna Cortes and Vapnik in 1993 and published in 1995.

1.1 What's SVM

A support vector machines (SVM) is a supervised learning model used for classification. It differs from logistic regression in the sense that it exploits the geometrical properties of the data; unlike the statistical approach of logistic regression, it tries to draw the best possible decision boundary for classification, and then uses this boundary for future predictions.

1.2 Hypothesis Function

The hypothesis function for an SVM is the same as that for logistic regression:

$$h_\theta(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^T\boldsymbol{x}\right) = \frac{1}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}}$$

1.3 Cost Function

The cost function is modified for an SVM. Here, we want a decision boundary which can distinguish between the two classes quite well, and we wouldn't like to accept a boundary that barely distinguishes data points. For $y=1$, we want $\boldsymbol{\theta}^T\boldsymbol{x} \gg 0$, and for $y=0$, we want $\boldsymbol{\theta}^T\boldsymbol{x} \ll 0$.

For regularized logistic regression, we had the following cost function:

$$\begin{aligned} J(\boldsymbol{\theta}) &= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h_\theta\left(\boldsymbol{x^{(i)}}\right)\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(\boldsymbol{x^{(i)}}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \\ &= \frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\left(-\log\frac{1}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x^{(i)}}}}\right) + \left(1-y^{(i)}\right)\left(-\log\left(1-\frac{1}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x^{(i)}}}}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \end{aligned}$$

For an SVM, we modify the cost function as follows:

$$J(\boldsymbol{\theta}) = C\sum_{i=1}^{m}\left[y^{(i)}cost_1\left(\boldsymbol{\theta}^T\boldsymbol{x^{(i)}}\right) + \left(1-y^{(i)}\right)cost_0\left(\boldsymbol{\theta}^T\boldsymbol{x^{(i)}}\right)\right] + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

Here, C represents the inverse of the regularization parameter λ. The two component cost functions above are:

$$cost_1(z) = \max\left(0, 1 - \boldsymbol{\theta}^T \boldsymbol{x}\right)$$
$$cost_0(z) = \max\left(0, -1 + \boldsymbol{\theta}^T \boldsymbol{x}\right)$$

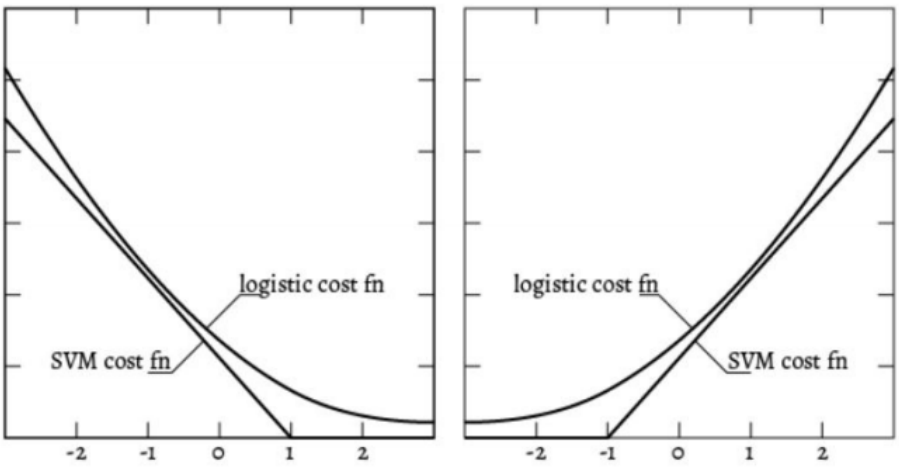


Figure 5.1: The two component cost functions of an SVM

1.4 Working of an SVM

If we ignore θ_0 and assume we are using a large value of C, the objective of an SVM can be expressed as follows:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \frac{1}{2}\|\boldsymbol{\theta}\| \\ \text{s.t.} \quad & \boldsymbol{\theta}^T \boldsymbol{x}^{(i)} \geq 1 \qquad \text{if } y^{(i)} = 1 \\ \text{and} \quad & \boldsymbol{\theta}^T \boldsymbol{x}^{(i)} \leq -1 \qquad \text{if } y^{(i)} = 0 \end{aligned}$$

Now, $\boldsymbol{\theta}^T \boldsymbol{x}^{(i)}$ is the scalar product of the vectors $\boldsymbol{\theta}$ and $\boldsymbol{x}^{(i)}$, and is thus also the projection of $\boldsymbol{x}^{(i)}$ along $\boldsymbol{\theta}$, multiplied by $\|\boldsymbol{\theta}\|$. Thus, the optimization of the above expression would mean maximizing the projection magnitudes of $\boldsymbol{x}^{(i)}$ along $\boldsymbol{\theta}$, so that $\|\boldsymbol{\theta}\|$ can be minimized. Thus, the SVM algorithm finds a hyperplane in the dataset which can clearly distinguish between data point classes, by maximizing the distance of the hyperplane from the data points of both classes. θ_0 appears just so that the optimal hyperplane is not constrained to pass through the origin. Another point to be noted is, the larger C is, the lesser is the model's regularization, and the more it tries to fit to the training dataset (i.e., to classify each training instance correctly). Thus, C must be chosen carefully to avoid overfitting.

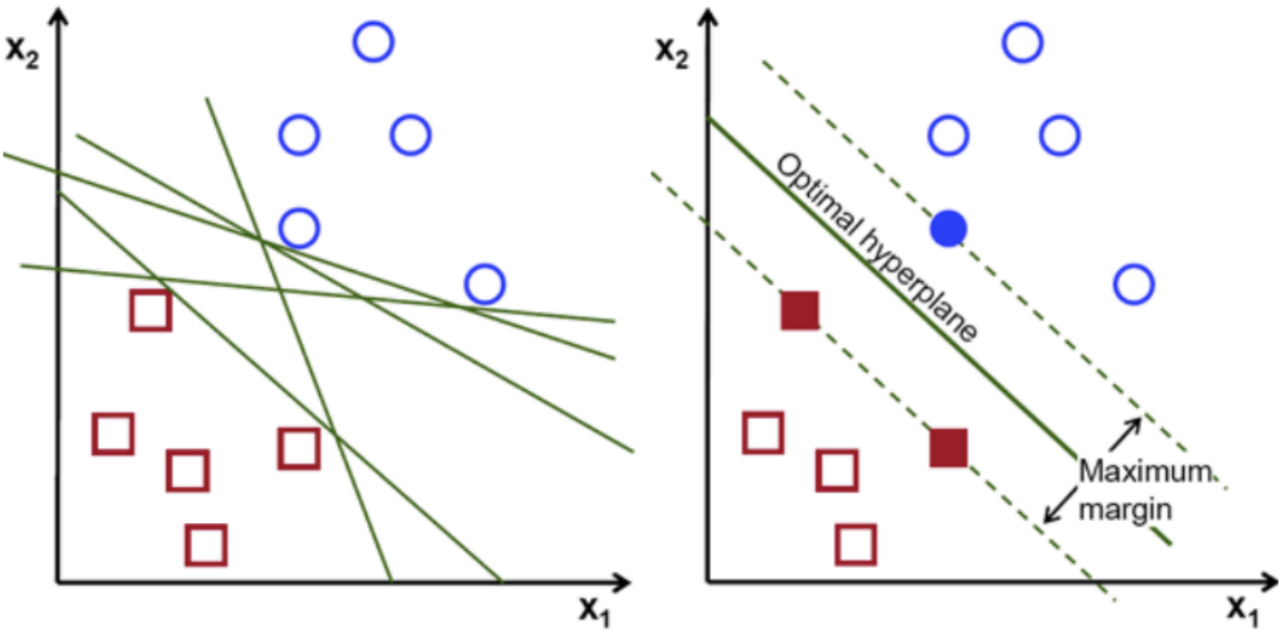


Figure 5.2: Hyperplane optimization by an SVM

1.5 SVMs with Kernels

Kernel functions are used with SVMs to make non-linear decision boundary formation very effective. A kernel function takes a data point and a landmark as input and returns a new data point. Thus, a dataset can be transformed into another using a kernel function. The most commonly used kernel is the Gaussian kernel, which we will denote by $sim\,(\boldsymbol{x}, \boldsymbol{y})$, since it is a measure of similarity between the two points $\boldsymbol{x}$ and $\boldsymbol{y}$.

$$sim\,(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{2\sigma^2}\right)$$

Given a dataset $\boldsymbol{X}$, let $\boldsymbol{l^{(1)}} = \boldsymbol{x^{(1)}}$, $\boldsymbol{l^{(2)}} = \boldsymbol{x^{(2)}}$, $\ldots$, $\boldsymbol{l^{(m)}} = \boldsymbol{x^{(m)}}$ be landmarks.
For each training example $\left(\boldsymbol{x^{(i)}}, y^{(i)}\right)$, we calculate:

$$\begin{aligned}
f_0^{(i)} &= 1 \\
f_1^{(i)} &= sim\left(\boldsymbol{x^{(i)}}, \boldsymbol{l^{(1)}}\right) \\
f_2^{(i)} &= sim\left(\boldsymbol{x^{(i)}}, \boldsymbol{l^{(2)}}\right) \\
&\vdots \\
f_m^{(i)} &= sim\left(\boldsymbol{x^{(i)}}, \boldsymbol{l^{(m)}}\right)
\end{aligned}$$

Thus, we obtain a new dataset $\boldsymbol{F}$ of size $m \times (m+1)$, and we use this to train the model.
Our new hypothesis predicts $y^{(i)} = 1$ if $\boldsymbol{\theta}^T \boldsymbol{f^{(i)}} \geq 0$, and $y^{(i)} = 0$ otherwise.
During training too, the cost function should be calculated using a new modified SVM cost function, replacing $\boldsymbol{\theta}^T \boldsymbol{x^{(i)}}$ with $\boldsymbol{\theta}^T \boldsymbol{f^{(i)}}$.

Note that:

- Large C and small σ can cause low bias and high variance.
- Small C and large σ can cause high bias and low variance.

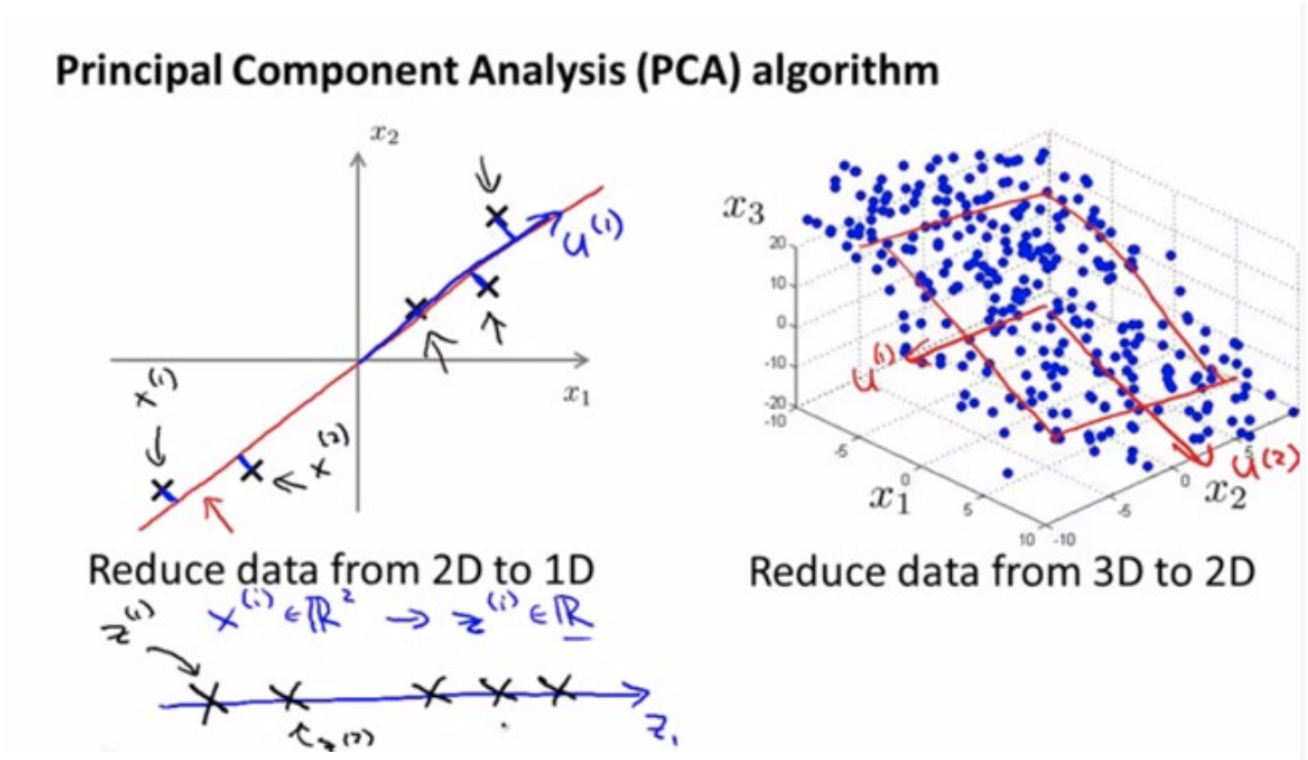
2 PCA : Principal Component Analysis

2.1 What's PCA

The main idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of many variables correlated with each other, either heavily or lightly, while retaining the variation present in the dataset, up to the maximum extent. The same is done by transforming the variables to a new set of variables, which are known as the principal components (or simply, the PCs) and are orthogonal, ordered such that the retention of variation present in the original variables decreases as we move down in the order. So, in this way, the 1st principal component retains maximum variation that was present in the original components. The principal components are the eigenvectors of a covariance matrix, and hence they are orthogonal.

Importantly, the dataset on which PCA technique is to be used must be scaled. The results are also sensitive to the relative scaling. As a layman, it is a method of summarizing data. Imagine some wine bottles on a dining table. Each wine is described by its attributes like colour, strength, age, etc. But redundancy will arise because many of them will measure related properties. So what PCA will do in this case is summarize each wine in the stock with less characteristics.

Intuitively, Principal Component Analysis can supply the user with a lower-dimensional picture, a projection or "shadow" of this object when viewed from its most informative viewpoint



2.2 Some Terminologies

- **Dimensionality:**
  It is the number of random variables in a dataset or simply the number of features, or rather more simply, the number of columns present in your dataset.

- **Correlation:**
  It shows how strongly two variables are related to each other. The value of the same ranges from -1 to +1. Positive indicates that when one variable increases, the other increases as well, while negative indicates the other decreases on increasing the former. And the modulus value of indicates the strength of relation.

- **Orthogonal:**
  Uncorrelated to each other, i.e., correlation between any pair of variables is 0.

- **Eigenvectors:**
  Consider a non-zero vector v. It is an eigenvector of a square matrix A, if Av is a scalar multiple of v. Then, v is the eigenvector and that scalar multiple is the eigenvalue associated with it.

- **Covariance Matrix:**
  This matrix consists of the covariances between the pairs of variables. The (i,j)th element is the covariance between i-th and j-th variable.

2.3 Properties of Principal Component

Technically, a principal component can be defined as a linear combination of optimally-weighted observed variables. The output of PCA are these principal components, the number of which is less than or equal to the number of original variables. Less, in case when we wish to discard or reduce the dimensions in our dataset. The PCs possess some useful properties which are listed below:

- The PCs are essentially the linear combinations of the original variables, the weights vector in this combination is actually the eigenvector found which in turn satisfies the principle of least squares.

- The PCs are orthogonal

- The variation present in the PCs decrease as we move from the 1st PC to the last one, hence the importance.

The least important PCs are also sometimes useful in regression, outlier detection, etc.

2.4 Implementing PCA on a 2-D Dataset

- **Step 1: Normalize the data**
  First step is to normalize the data that we have so that PCA works properly. This is done by subtracting the respective means from the numbers in the respective column. So if we have two dimensions X and Y, all X become x^- and all Y become y^-. This produces a dataset whose mean is zero.

- **Step 2: Calculate the covariance matrix**
  Since the dataset we took is 2-dimensional, this will result in a 2x2 Covariance matrix. Note that Var[X_1]=Cov[X_1,X_1] and Var[X_2]=Cov[X_2,X_2].

  $$\Rightarrow \text{Matrix(Covariance)} = \begin{bmatrix} Var[\text{X}_1] & Cov[\text{X}_1,X_2] \\ Cov[\text{X}_2,X_1] & Var[\text{X}_2] \end{bmatrix}$$

- **Step 3: Calculate the eigenvalues and eigenvectors**
  Next step is to calculate the eigenvalues and eigenvectors for the covariance matrix. The same is possible because it is a square matrix. λ is an eigenvalue for a matrix **A** if it is a solution of the characteristic equation: $\mathbf{det(\lambda I - A) = 0}$ where, **I** is the identity matrix of the same dimension as **A** which is a required condition for the matrix subtraction as well in this case and **'det'** is the determinant of the matrix. For each eigenvalue λ, a corresponding eigen-vector **v**, can be found by solving: $\mathbf{(\lambda I - A)v = 0}$

- **Step 4: Choosing components and forming a feature vector**
  We order the eigenvalues from largest to smallest so that it gives us the components in order or significance. Here comes the dimensionality reduction part. If we have a dataset with n variables, then we have the corresponding n eigenvalues and eigenvectors. It turns out that the eigenvector corresponding to the highest eigenvalue is the principal component of the dataset and it is our call as to how many eigenvalues we choose to proceed our analysis with. To reduce the dimensions, we choose the first p eigenvalues and ignore the rest. We do lose out some information in the process, but if the eigenvalues are small, we do not lose much.

  Next we form a feature vector which is a matrix of vectors, in our case, the eigenvectors. In fact, only those eigenvectors which we want to proceed with. Since we just have 2 dimensions in the running example, we can either choose the one corresponding to the greater eigenvalue or simply take both.

  $\Rightarrow$ Feature Vector = (eig_1,eig_2)

- **Step 5: Forming Principal Components:**
  This is the final step where we actually form the principal components using all the math we did till here. For the same, we take the transpose of the feature vector and left-multiply it with the transpose of scaled version of original dataset.

  $$\text{NewData} = FeatureVector^T \times ScaledData^T$$

  Here, *NewData* is the Matrix consisting of the principal components, *FeatureVector* is the matrix we formed using the eigenvectors we chose to keep and *ScaledData* is the scaled version of original. 'T' in the superscript denotes transpose of a matrix which is formed by interchanging the rows to columns and vice versa.

  If we go back to the theory of eigenvalues and eigenvectors, we see that, essentially, eigenvectors provide us with information about the patterns in the data. In particular, in the running example of 2-D set, if we plot the eigenvectors on the scatterplot of data, we find that the principal eigenvector (corresponding to the largest eigenvalue) actually fits well with the data. The other one, being perpendicular to it, does not carry much information and hence, we are at not much loss when deprecating it, hence reducing the dimension.

All the eigenvectors of a matrix are perpendicular to each other. So, in PCA, what we do is represent or transform the original dataset using these orthogonal (perpendicular) eigenvectors instead of representing on normal $\mathbf{x}$ and $\mathbf{y}$ axes. We have now classified our data points as a combination of contributions from both $\mathbf{x}$ and $\mathbf{y}$. The difference lies when we actually disregard one or many eigenvectors, hence, reducing the dimension of the dataset. Otherwise, in case, we take all the eigenvectors in account, we are just transforming the co-ordinates and hence, not serving the purpose.

2.5 Applications of Principal Component Analysis

PCA is predominantly used as a dimensionality reduction technique in domains like facial recognition, computer vision and image compression. It is also used for finding patterns in data of high dimension in the field of finance, data mining, bioinformatics, psychology, etc.