CS 217: Artificial Intelligence and Machine Learning **Jan-Apr 2024**

Lecture 16: Unsupervised Learning through Clustering

Lecturer: Swaprava Nath *Scribe(s): SG31 & SG32*

16.1 Support Vector Machines (Recap)

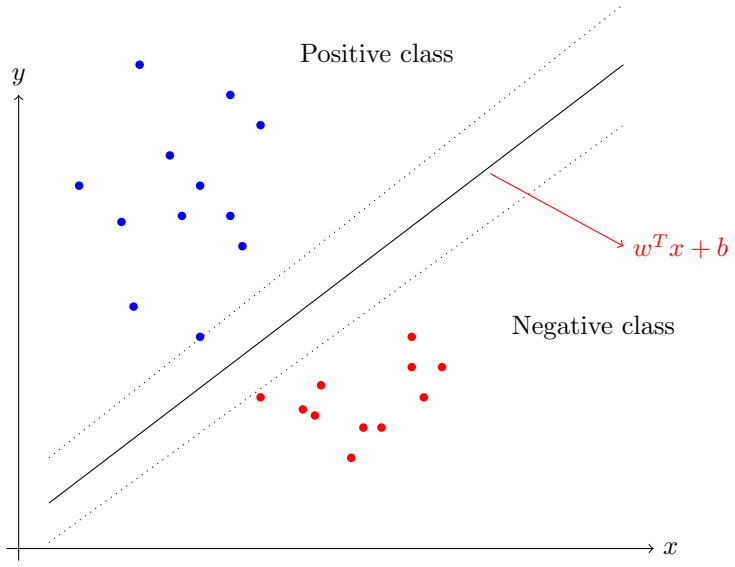

Figure 16.1: Linear SVM separating two classes

We had discussed in the previous lecture about SVM's and saw that the optimization problem can be represented in two forms:

Primal Form:

$$\begin{aligned} \text{Minimize:} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to:} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad \text{for all } i = 1, 2, \ldots, n \end{aligned}$$

Dual Form:

$$\begin{aligned} \text{Maximize:} \quad & \max_{\lambda \geq 0} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum \sum \lambda_i \lambda_j y_i y_j \left(x_i^\top x_j\right) \\ \text{subject to:} \quad & \sum \lambda_i y_i = 0 \\ \text{has solution:} \quad & w^* = \sum \lambda_i^* y_i x_i \end{aligned}$$

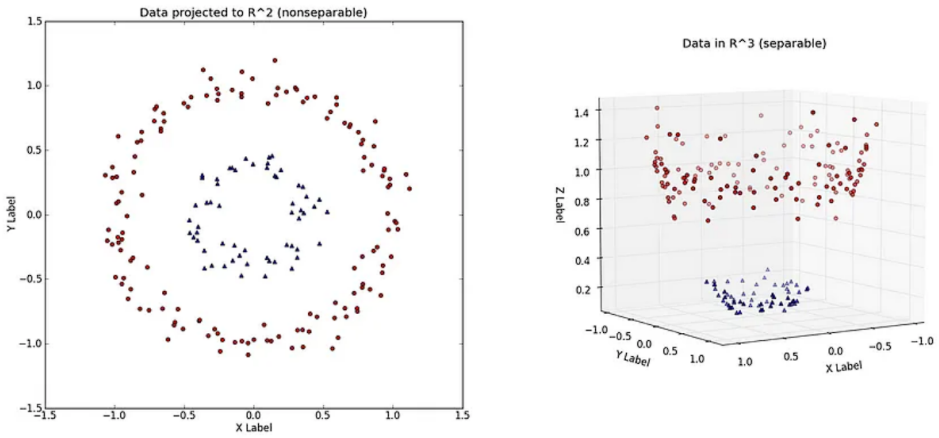

Figure 16.2: Separating non-linearly separable data points. Image taken from [Medium]

Turns out that a dual problem is easier to solve because:

- Less costly computation as less number of variables and less number of constraints.
- Kernelisation to treat points that are linearly non-separable.

In Figure 16.2 we have non-linearly separable data points. In this case, even the soft margin classifier is not very good, so to classify the points using SVM we can do basis transformation i.e we can project the data to a high dimensional space and then we can pass a linear classifier on the transform space.
But here the problem is x_i itself is a high dimensional vector and doing a transformation would result in an even more high dimensional vector due to which first doing transformation and then doing the inner product $\left(\phi(x_i)^\top \phi(x_j)\right)$ in the dual-problem optimization equation would be very computationally expensive. Hence we use the method of **kernelisation**.

16.2 Kernelisation

Kernelisation is used for points which are not linearly separable.

Given : Linearly inseparable data

Goal : Project this data to a high dimensional space and then solve using SVM. (Find w,h in the higher dimension)

Step 1 :

$$\begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \end{bmatrix} \xrightarrow{\text{transform}} \begin{bmatrix} 1 \\ x_i^{(1)} \\ x_i^{(2)} \\ x_i^{(1)} x_i^{(2)} \\ (x_i^{(1)})^2 \\ (x_i^{(2)})^2 \end{bmatrix} = \phi(x_i)$$

Step 2 : Dual of SVM

$$\max_{\lambda \geq 0} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_j \sum_i \lambda_i \lambda_j y_i y_j (\phi(x_i))^T \phi(x_j) \text{ such that } \sum_{i=1}^{n} \lambda_i y_i = 0$$

The objective function has an inner product of the data in the higher dimension.

Original Data		Transformed data		Inner Product of Transformed points
x_i	$\xrightarrow{\phi}$	$\phi(x_i)$	$\rightarrow$	
Size : $n \times d$		Size : $n \times m$		Size : $n \times n$

Now, is the first transformation (ϕ) really necessary ?

Do there exist functions that calculate the inner product in the transform space without explicitly computing the transformed data ?

The answer turns out to be YES in some cases, and it is possible through **Kernel Functions**

Example : Consider the previous case, $\left[1, x_i^1 x_j^1, x_i^2 x_j^2, (x_i^1)^2 (x_j^1)^2, (x_i^2)^2 (x_j^2)^2\right]$ and $(1 + x_i^T x_j)^2$, They have the same terms, Hence we have computed the inner product without calculating the transformation.

Kernel trick : Transformations are equivalent as long as we are calculating dual of SVM

Different Kernel Functions :

1. Linear K(x,z) = $x^T z$
2. Polynomial K(x,z,d) = $(1 + x^T z)^d$
3. Gaussian K(x,z) = $e^{\frac{-||x-z||^2}{2\sigma^2}}$
4. Laplacian / Radial K(x,z) = $e^{\frac{-||x-z||}{2\sigma^2}}$

Some uses of SVM : Handwriting recognition, Protein Structure, Medical Image Classification.

A set of necessary and sufficient conditions govern the kernel functions, like which kernel function to pick (decided by grid search), is defined by **Mercer's Theorem**

Limitations of SVM :

1. Only Binary Classification is possible [One vs. Rest Classifier]
2. No probabilistic interpretation exists
3. It doesn't work well with noisy data

16.3 Unsupervised Learning

Until now, everything we have done comes under supervised learning, where training data is labelled, that is, $D = \{(x_i, y_i)\}_{i \in [n]}$ where x_i is the data and y_i is its label. However, in many situations, we may need to find relations with unlabelled data. This may be due to reasons like-

1. **High cost of labelling data**: While it is easy to find data for just about any situation, it is costly and time consuming to label such data with the output as this requires human intervention.
2. It may also be irrelevant to classify data by labels and make better sense to look at properties of unlabelled data.

For this purpose we introduce the methods of **Unsupervised Learning** where our data is of the form $D = \{(x_i)\}_{i\in[n]}$ where x_i is a d-dimensional vector. Among unsupervised learning methods, we shall now look at clustering.

16.3.1 Clustering

Clustering is the partitioning of data into proper subsets based on a function of their location in d-dimensional space.
Let our data be of the form $D = \{x_1, x_2, \ldots, x_n\}, x_i \in \mathbb{R}^d$.
Goal: Find a "well separated" partition of the data

$$D = D_1 \cup D_2 \cup D_3 \cup \cdots \cup D_k$$

where $1 \leq k \leq n$ is a hyperparameter. We also define the condition for **hard clustering** as-

$$D_i \cap D_j = \Phi \qquad \forall i \forall j, i \neq j$$

For any type of clustering, there exists a **Clustering Function**, $C : [n] \to [k]$, such that for each data point x_i that is mapped to the partition $D_{k'}$, $C(i) = k'$.

16.3.2 k-Means Clustering

We shall now look at a method for clustering called k-means clustering, wherein we characterize clusters by the cluster centres calculated by the algebraic mean of all data points within that cluster.

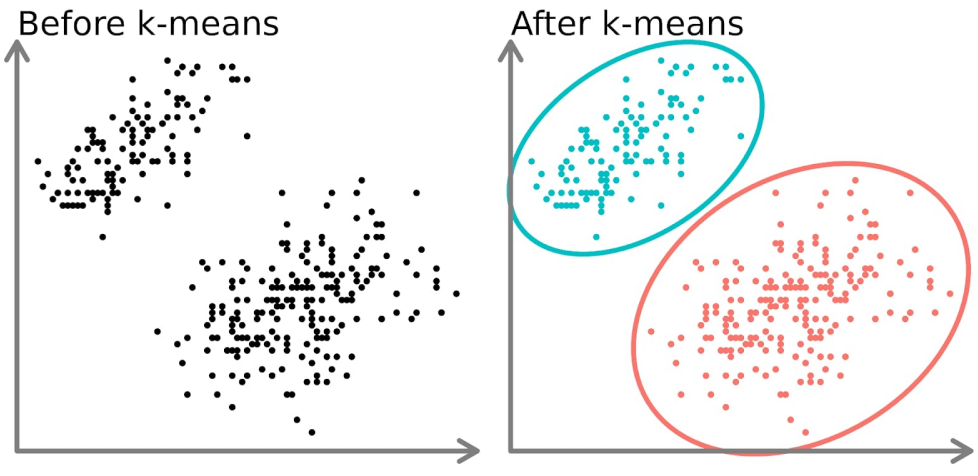

Figure 16.3: Example of k-means clustering

16.3.2.1 Clustering as on Optimization Problem

We can look at the clustering problem as an optimization problem as we want to decrease the total sum of "error" over all data points. Let us define the error for a data point x_i as the square of the L_2-distance between x_i and the centre of the cluster assigned to x_i, thus, $E_{x_i} = ||x_i - m_{C(i)}||_2^2$. To minimize the total error, we want to solve the following problem-

$$\underset{c}{\operatorname{argmin}} \sum_{i=1}^{n} ||x_i - m_{C(i)}||_2^2 \qquad\qquad c \in \mathcal{C}$$

$\mathcal{C}$ represents the set of all possible partitions of D into k sets and thus $|\mathcal{C}| = k^n$. As there are exponentially many possible partitions to this problem, solving this problem for a global optimum is NP-hard. However, we notice that our objective function, which we write as

$$SE(c, m) = \sum_{i=1}^{n} ||x_i - m_{C(i)}||_2^2$$

is a non-convex function and thus may have local minima in addition to the global minimum. We shall now give an algorithm for k-means clustering which converges in polynomial time onto one of these local minima, giving us a "reasonable" partition of our clustering.

16.3.2.2 k-Means Clustering Algorithm

Let us take the input $D = \{x_1, x_2, \ldots, x_n\}$
Initialise the cluster means $\{m_1, m_2, \ldots, m_k\}$ and $c(i)$. We shall denote these by m_i^0 and $c^0(i)$ respectively to denote time $t = 0$. Also, initialize L to store the size of each cluster.
Now we shall repeat the following steps until convergence of the assignments (that is, $C^{t+1}(i) = C^t(i)\ \forall\, i$):

1. $\forall\, i \in [n]$, check if $\exists k' \neq C^t(i)$ such that $||m_{k'}^t - x_i||_2 < ||m_{C^t(i)}^t - x_i||$
2. If such a k' is found, assign $c^{t+1}(i) = k'$ and update

$$m_{k'}^{t+1} = \frac{m_{k'}^t \times L(k') + ||m_{k'}^t - x_i||_2}{L(k') + 1)}$$

$$m_{C^t(i)}^{t+1} = \frac{m_{C^t(i)}^t \times L(C^t(i)) - ||m_{C^t(i)}^t - x_i||_2}{L(C^t(i)) - 1}$$

3. Update cluster sizes as well and repeat from step 1
4. If k' is not found, we are done as assignment does not change.

What remains is to prove that this algorithm will always converge in polynomial time, which we shall do in the next part.

16.3.2.3 Proof of Convergence

Let us assume that the algorithm goes from iteration t to iteration $t+1$: This assumes that we have a sequence of iterations where the algorithm updates the cluster centers and assigns points to clusters iteratively. **Now we need to prove that** $SSE(C_{t+1}, \mu_{t+1}) < SSE(C_t, \mu_t)$**:** Here, SSE stands for "Sum of Squared Errors,"

which is a measure of how well the clusters fit the data points. The goal is to show that in each iteration, the SSE decreases, indicating improvement in clustering quality.

If the SSE decreases in each iteration, eventually the algorithm will reach a point where no further improvement can be made, leading to termination.

The algorithm stops when each point is assigned to the cluster with the closest center, indicating that the clustering is locally optimal at that stage.

The proof is divided into two steps to demonstrate that the SSE decreases from iteration t to iteration $t+1$.

First step involves showing that updating the cluster centers (from μ_t to μ_{t+1}) while keeping the cluster assignments (C_{t+1}) the same leads to a decrease in SSE.

Second step

$$SSE(C_{t+1}, \mu_{t+1}) = \sum_{i=1}^{n} \|x_i - \mu_{t+1} C_{t+1}(i)\|^2$$

This is the formula for calculating the SSE between the data points and the cluster centers in iteration $t+1$ with the updated cluster assignments.

$$= \sum_{k} \sum_{k_0=1}^{k} \sum_{i \in [n], C_{t+1}(i)=k_0} \|x_i - \mu_{t+1} C_{t+1}(i)\|^2$$

Splits the summation over clusters (k) and data points (i) based on the cluster assignments in iteration $t+1$.

$$\leq \sum_{k} \sum_{k_0=1}^{k} \sum_{i \in [n], C_{t+1}(i)=k_0} \|x_i - \mu_t C_{t+1}(i)\|^2$$

This step uses the lemma to show that the SSE with updated cluster centers (μ_{t+1}) is less than or equal to the SSE with the previous cluster centers (μ_t).

$$= \sum_{i=1}^{n} \|x_i - \mu_t C_{t+1}(i)\|^2 = SSE(C_{t+1}, \mu_t)$$

Combines the terms back together to show that the SSE with updated cluster centers and cluster assignments is equal to $SSE(C_{t+1}, \mu_t)$, completing the proof.

References

[Medium] Vivek Yadav, *How Neural Networks Learn Nonlinear Functions and Classify Linearly Non-Separable Data*, Medium, Available at: https://vivek-yadav.medium.com/how-neural-networks-learn-nonlinear-functions-and-classify-linearly-non-separable-data-22328e7e5be1

[K-Means] `https://www.datacamp.com/tutorial/k-means-clustering-r`