07 - Support Vector Machines

Ethan Graham

20th June 2023

General Idea

When we have non-linearly separable data, we map it to a higher dimension. For example: map $1D$ data with the following $x \rightarrow \phi(x) = \begin{pmatrix} x \\ x^2 \end{pmatrix}$ This allows us to separate data in some cases (check slides) The same idea applies to more than one dimension, of course.

Mapping to a higher dimension

We must be careful because the dimension of ϕ grows rapidly with the number of features.

We train polynomial SVM in the same way as linear SVM, except we map to higher dimension. → **run linear svm on $\phi(\mathbf{x})$**

A higher degree polynomial gives us a more flexible boundary, but it also increases the dimensionality of the problem. **The computational complexity grows in** $O(n^3)$ *(where n is the number of features)*

HOG (Histogram of Oriented Gradients)

I think this is outside of scope of course but will double-check

Cover's Theorem

A complex pattern classification problem mapped to a higher dimension is more likely to be linearly separable than a low dimensional space, provided that the space is **not densely populated**

Overally, higher dimesions are better. But the problem that arises is that when we deal with many many samples, the computational complexity increases.

Lagrange Multipliers

To find a constrained minimization i.e. minimize $f(x, y)$ subject to $g(x, y) \leq c$ we use **Lagrange multipliers**. At the constrained minimum we have $\exists \lambda \in$

$\mathbb{R}$, $\nabla f = \lambda \nabla g$ where λ is the Lagrange multiplier.

Lagrangian Formulation of SVM optimization

$L(\mathbf{w}, \Lambda) = \frac{1}{2}\left\|\mathbf{w}\right\|^2 - \sum\limits_{n=1...N} \lambda_n(t_n \tilde{\mathbf{w}} \cdot \phi(\mathbf{x_n}) - 1)$ Where we have $\Lambda = [\lambda_1, ..., \lambda_n]$ (one multiplier per constraint)

Theorem: A solution to the constrained minimization problem must be such that L is *minimized* with respect to the components of $\mathbf{w}$ and *maximized* with respect to the Lagrange multipliers, which are ≥ 0

The Kernel Trick

Let's say that we want to map our x_i points with a function ϕ. The kernel trick allows us to do this **without** having to compute $\phi(x)$ for every point.

We just need to know how $\phi(x)$ compares to each other point $\phi(x')$. Mathematically, this corresponds to doing the inner product $\phi(x)^T\phi(x') = k(x, x')$ the ***kernel function***.

The kernel function can be seen as a *similarity metric*

Example: Linear Kernel

This corresponds to the identity function $\phi(x) = x$. We get $k(x, x') = x^T x'$.

```
SVC(kernel='linear').fit(X,y) # yields a linear decision boundary!
```

Example: Polynomial Kernel

The transformation $\phi(\mathbf{x}) = (x_1, x_2, x_1x_2, x_1^2, x_2^2)$ corresponds to the kernel $k(\mathbf{x}, \mathbf{x'}) = (1 + \mathbf{x^T x'})^2$. This is a lot easier to compute, even if the transformation is complicated.

```
SVC(kernel='poly').fit(X,y) # this will give us the polynomial kernel discussed above
```

Example: RBF Kernel

Sometimes it's possible to give a kernel function for a non-linear transformation that is hard/impossible to compute. Prime example is the *RBF function*. $k(\mathbf{x}, \mathbf{x'}) = e^{-\gamma\left\|\mathbf{x}-\mathbf{x'}\right\|^2}$ which actually yields ***infinite dimension***

```
SVC(kernel='rbf', gamma=0.01).fit(X,y) # uses the RBF kernel
```

The γ parameter allows us to go from a smooth to a rough function. It is super easy to tweak the RBF kernel, and we don't have to worry about the output dimension.

Formulation

$y(\mathbf{x}) = \sum_{n \in S} \lambda_n t_n k(\mathbf{x}, \mathbf{x_n}) + b$

The only subset of points with $\lambda_i \neq 0$ are the support vectors. Here we have $t_i y_i = 1$ Here we have that ϕ is implicit. In practice we only have to compute k

Working with a **gaussian kernel** (RBF) virtually makes the dimension as large as the number of samples. Unfortunately, it is still computationally complex.

Summary

SVM performs well on MNIST as we can linearly separate in high-dimension space when we can't in the input-space. Classifiers can be learned in the feature space without having to actually perform the mapping. The main issue with SVM is the $O(n^3)$ time and space complexity which makes it difficult to exploit large datasets.