Dual and Kernel Perceptron

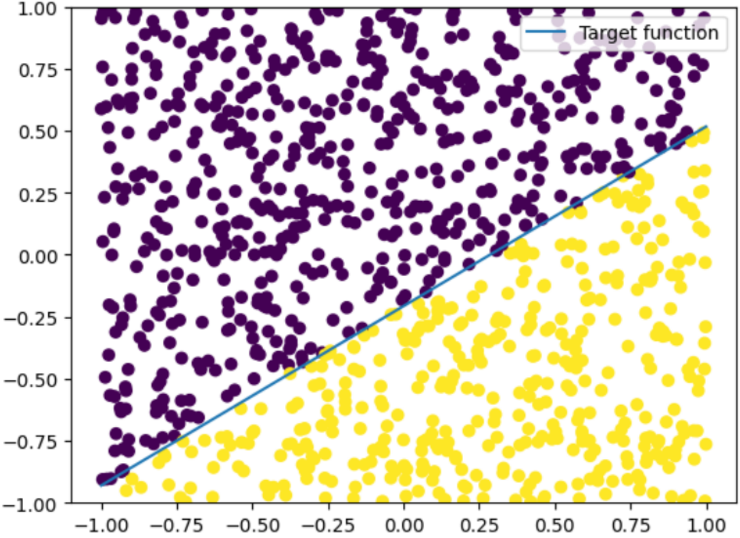
Ella Kim

PRIMES CS Circle

December 2023

Table of Contents

Graph



Target function
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

Perceptron Review

- Weight vector w, initially set to all-0 vector
- Initial hypothesis: $h(x) = \text{sign}(w \cdot x)$

Perceptron Review

- Weight vector w, initially set to all-0 vector
- Initial hypothesis: $h(x) = \text{sign}(w \cdot x)$
- Given an example $x \in \mathbb{R}^n$ and its label $c(x) = \text{sign}(v \cdot x)$,
  - If $h(x) = c(x)$ then no update is performed
  - If $h(x) \neq c(x)$ then w is updated by setting w_{new} to $w + c(x)x$
    Example: False positive: $h(x) = 1$, $c(x) = -1 \implies w_{\text{new}} = w - x$
    $(w - x) \cdot x = w \cdot x - x \cdot x < w \cdot x$

Motivation

What if the data's not linearly separable?

- We can try representing it in a higher dimension!
- But this can be computationally expensive :(

To deal with this, we can use kernel functions.

Definition

Definition

A **kernel** is a function $K(x, y)$ such that for some mapping $\phi : \mathbb{R}^n \to \mathbb{R}^N$, $K(x, y) = \phi(x) \cdot \phi(y)$.

Definition

Definition

A **kernel** is a function $K(x, y)$ such that for some mapping $\phi : \mathbb{R}^n \to \mathbb{R}^N$, $K(x, y) = \phi(x) \cdot \phi(y)$.

Suppose $x, y \in \mathbb{R}^n$ and $\phi(x), \phi(y) \in \mathbb{R}^N$, $n < N$. If N is very large, just writing down $\phi(x)$ and $\phi(y)$ or computing $\phi(x) \cdot \phi(y)$ can take an enormous amount of time.

Definition

Definition

A **kernel** is a function $K(x, y)$ such that for some mapping $\phi : \mathbb{R}^n \to \mathbb{R}^N$, $K(x, y) = \phi(x) \cdot \phi(y)$.

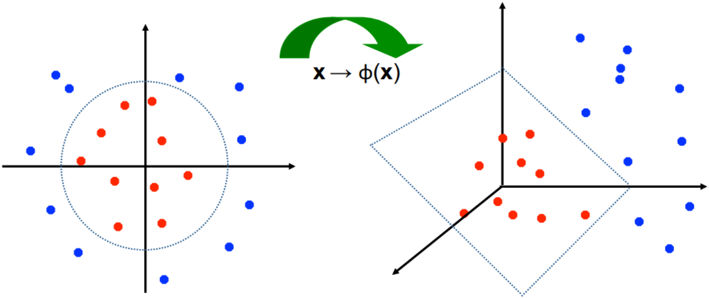
Suppose $x, y \in \mathbb{R}^n$ and $\phi(x), \phi(y) \in \mathbb{R}^N$, $n < N$. If N is very large, just writing down $\phi(x)$ and $\phi(y)$ or computing $\phi(x) \cdot \phi(y)$ can take an enormous amount of time.

Since $K(x, y)$ is a function of x and y (which are in n), we can potentially compute the desired value much faster by using this kernel function!

Example

Say the decision boundary for our data is some kind of ellipse with equation $x_1^2 + x_2^2 + \sqrt{2}x_1x_2$.

This equation isn't linear in x_1, x_2. To deal with this, create a mapping $\phi : \mathbb{R}^2 \to \mathbb{R}^3$, for $x = (x_1, x_2)$, $\phi(x) = \phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$.



Example

Say the decision boundary for our data is some kind of ellipse with equation $x_1^2 + x_2^2 + \sqrt{2}x_1 x_2$.

This equation isn't linear in x_1, x_2. To deal with this, create a mapping $\phi : \mathbb{R}^2 \to \mathbb{R}^3$, for $x = (x_1, x_2)$, $\phi(x) = \phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1 x_2)$.

Suppose we have $a = (a_1, a_2)$ and $b = (b_1, b_2)$ and we want to compute $K(a, b)$.

$$\begin{aligned}
\phi(a) \cdot \phi(b) &= (a_1^2, a_2^2, \sqrt{2}a_1 a_2) \cdot (b_1^2, b_2^2, \sqrt{2}b_1 b_2) \\
&= a_1^2 b_1^2 + a_2^2 b_2^2 + 2a_1 a_2 b_1 b_2 \\
&= (a_1 b_1 + a_2 b_2)^2 \\
&= (a \cdot b)^2 \\
&= K(a, b)
\end{aligned}$$

Dual Perceptron

- Formulates our Perceptron algorithm in a slightly different way
- Replaces hypothesis vector with a new collection of examples where the algorithm has made a mistake
- Allows for algorithm to only depend on taking inner products between examples in $\mathbb{R}^n$
- We can apply kernel functions!

Dual Perceptron

Suppose we are at an intermediate step in dual Perceptron.

- Algorithm has made k mistakes so far, on examples $x_{i_1}, x_{i_2}, \ldots, x_{i_k} \in \mathbb{R}^n$
- Corresponding labels $c(x_{i_1}), c(x_{i_2}), \ldots, c(x_{i_k}) \in \{-1, 1\}$

Hypothesis vector

$$w = \sum_{j=1}^{k} c(x_{i_j}) x_{i_j}$$

Dual Perceptron

Hypothesis vector

$$w = \sum_{j=1}^{k} c(x_{i_j}) x_{i_j}$$

$w \cdot x$

$$w \cdot x = \left(\sum_{j=1}^{k} c(x_{i_j}) x_{i_j} \right) \cdot x = \sum_{j=1}^{k} c(x_{i_j}) x_{i_j} \cdot x$$

This means that we only ever need to compute inner products between examples $x_{i_j}, x \in \mathbb{R}^n$ in order to able to compute $w \cdot x$.

Kernelized Dual Perceptron

- Running dual Perceptron over higher dimensional $\mathbb{R}^N$
- Using kernel function $\phi : \mathbb{R}^n \to \mathbb{R}^N$ for inner product computations

Kernelized Dual Perceptron

Suppose we are at an intermediate step in kernelized dual Perceptron.

- Algorithm has made k mistakes so far, on examples $x_{i_1}, x_{i_2}, \ldots, x_{i_k} \in \mathbb{R}^n$
- Label examples according to $c(x) = \text{sign}(v \cdot \phi(x))$ (note that v is a N-dimensional vector)

Hypothesis vector

$$w = \sum_{j=1}^{k} c(x_{i_j}) \phi(x_{i_j})$$

Kernelized Dual Perceptron

Hypothesis vector

$$w = \sum_{j=1}^{k} c(x_{i_j})\phi(x_{i_j})$$

$w \cdot \phi(x)$

$$w \cdot \phi(x) = \left(\sum_{j=1}^{k} c(x_{i_j})\phi(x_{i_j})\right) \cdot \phi(x) = \sum_{j=1}^{k} c(x_{i_j})\phi(x_{i_j}) \cdot \phi(x)$$

$$= \sum_{j=1}^{k} c(x_{i_j})K(x_{i_j}, x)$$

Kernelized Dual Perceptron

$w \cdot \phi(x)$

$$w \cdot \phi(x) = \sum_{j=1}^{k} c(x_{i_j}) K(x_{i_j}, x)$$

This can be computed efficiently if K can be computed efficiently! Note that we never have to explicitly write down the high-dimensional vector w while running kernelized dual Perceptron.

Acknowledgements

- I would like to thank Lali and Surya for being wonderful mentors.
- I would also like to thank the MIT PRIMES program for this amazing opportunity.