Kernel Methods

COMP9417, 22T2

Kernel Methods

Primal vs. Dual Algorithms

Primal vs. Dual Algorithms

The *dual* view of a problem is simply just another way to view a problem mathematically.

Primal vs. Dual Algorithms

The *dual* view of a problem is simply just another way to view a problem mathematically.

Instead of pure parameter based learning (i.e minimising a loss function etc.), dual algorithms introduce **instance-based** learning.

Primal vs. Dual Algorithms

The *dual* view of a problem is simply just another way to view a problem mathematically.

Instead of pure parameter based learning (i.e minimising a loss function etc.), dual algorithms introduce **instance-based** learning.

This is where we 'remember' mistakes in our data and adjust the corresponding weights accordingly.

We then use a *similarity function* or **kernel** in our predictions to weight the influence of the training data on the prediction.

In the primal problem, we typically learn parameters:

$$\mathbf{w} \in \mathbb{R}^p$$

meaning we learn parameters for each of the p features in our dataset.

In the primal problem, we typically learn parameters:

$$\mathbf{w} \in \mathbb{R}^p$$

meaning we learn parameters for each of the p features in our dataset.

In the dual problem, we typically learn parameters:

$$\alpha_i \qquad \text{for } i \in [1, n]$$

meaning we learn parameters for each of the n **data-points**.

In the primal problem, we typically learn parameters:

$$\mathbf{w} \in \mathbb{R}^p$$

meaning we learn parameters for each of the p features in our dataset.

In the dual problem, we typically learn parameters:

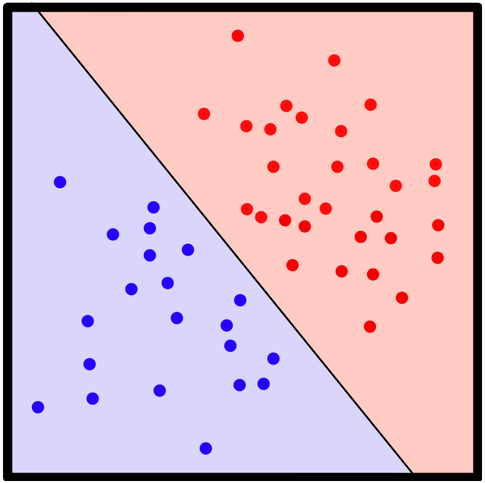
$$\alpha_i \qquad \text{for } i \in [1, n]$$

meaning we learn parameters for each of the n **data-points**.

α_i represents the *importance* of a data point (x_i, y_i).

What do we mean by importance?

What do we mean by importance?



The Dual/Kernel Perceptron

Recall the *primal* perceptron:

```
$converged \leftarrow 0$
**while** not $converged$ **do**
    $converged \leftarrow 1$
    **for** $x_i \in X, y_i \in y$ **do**
        **if** $y_i w \cdot x_i \leq 0$ **then**
            $w \leftarrow w + \eta y_i x_i$
            $converged \leftarrow 0$
        **end if**
    **end for**
**end while**
```

The Dual/Kernel Perceptron

Recall the *primal* perceptron:

```
$converged \leftarrow 0$
**while** not $converged$ **do**
    $converged \leftarrow 1$
    **for** $x_i \in X, y_i \in y$ **do**
        **if** $y_i w \cdot x_i \leq 0$ **then**
            $w \leftarrow w + \eta y_i x_i$
            $converged \leftarrow 0$
        **end if**
    **end for**
**end while**
```

If we define the number of iterations the perceptron makes as $K \in \mathbb{N}^+$ and assume $\eta = 1$. We can derive an expression for the final weight vector $w^{(K)}$:

The Dual/Kernel Perceptron

Recall the *primal* perceptron:

```
$converged \leftarrow 0$
while not $converged$ do
    $converged \leftarrow 1$
    for $x_i \in X, y_i \in y$ do
        if $y_i w \cdot x_i \leq 0$ then
            $w \leftarrow w + \eta y_i x_i$
            $converged \leftarrow 0$
        end if
    end for
end while
```

If we define the number of iterations the perceptron makes as $K \in \mathbb{N}^+$ and assume $\eta = 1$. We can derive an expression for the final weight vector $w^{(K)}$:

$$w^{(K)} = \sum_{i=1}^{N} \sum_{j=1}^{K} \mathbf{1}\{y_i w^{(j)} x_i \leq 0\} y_i x_i$$

We can simply our expression and take out the indicator variable:

$$
\begin{aligned}
w^{(K)} &= \sum_{i=1}^{N} \sum_{j=1}^{K} \mathbf{1}\{y_i w^{(j)} x_i \leq 0\} y_i x_i \\
&= \sum_{i=1}^{N} \alpha_i y_i x_i
\end{aligned}
$$

where α_i is the number of times the perceptron makes a mistake on a data point (x_i, y_i).

If we sub in $w^{(K)} = \sum_{i=1}^{N} \alpha_i y_i x_i$. We get the algorithm for the **dual** perceptron.

If we sub in $w^{(K)} = \sum_{i=1}^{N} \alpha_i y_i x_i$. We get the algorithm for the **dual** perceptron.

```
$converged \leftarrow 0$
while not $converged$ do
    $converged \leftarrow 1$
    for $x_i \in X, y_i \in y$ do
        if $y_i \sum_{j=1}^{N} \alpha_j y_j x_j \cdot x_i \leq 0$ then
            $\alpha_i \leftarrow \alpha_i + 1$
            $converged \leftarrow 0$
        end if
    end for
end while
```

Gram Matrix

The Gram matrix represents the *inner product* of two vectors.
For a dataset X we define $G = X^T X$. That is:

Gram Matrix

The Gram matrix represents the *inner product* of two vectors.
For a dataset X we define $G = X^T X$. That is:

$$G = \begin{bmatrix} \langle x_1, x_1 \rangle & \langle x_1, x_2 \rangle & \cdots & \langle x_1, x_n \rangle \\ \langle x_2, x_1 \rangle & \langle x_2, x_2 \rangle & \cdots & \langle x_2, x_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle x_n, x_1 \rangle & \langle x_n, x_2 \rangle & \cdots & \langle x_n, x_n \rangle \end{bmatrix}$$

$$G_{i,j} = \langle x_i, x_j \rangle$$

Transformations

Transformations

How do we go about solving **non-linearly separable** datasets with linear classifiers?

Transformations

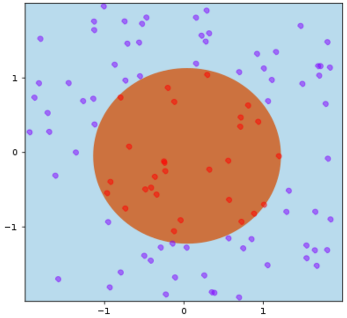
How do we go about solving **non-linearly separable** datasets with linear classifiers?

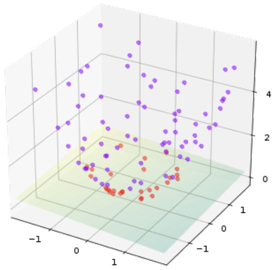
Project them to higher dimensional spaces through a transformation $\phi : \mathbb{R}^p \to \mathbb{R}^k$.

Transformations

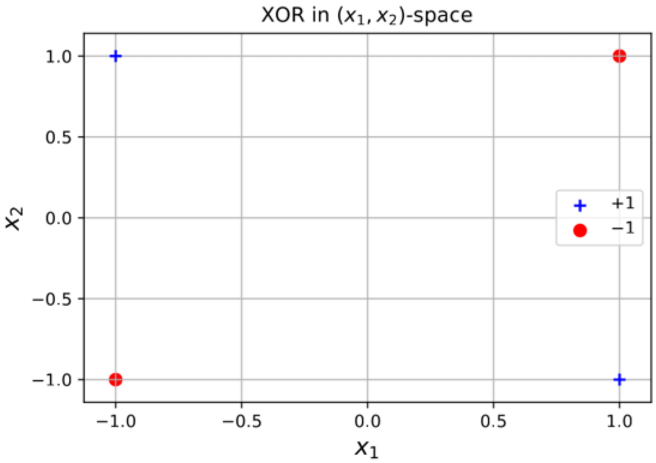
How do we go about solving **non-linearly separable** datasets with linear classifiers?

Project them to higher dimensional spaces through a transformation $\phi : \mathbb{R}^p \to \mathbb{R}^k$.





Let's revisit the XOR.



A solution:

A solution:

For our input vectors in the form $\mathbf{x} = [x_1, x_2]^T$, use a transformation:

$$\phi(\mathbf{x}) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{bmatrix}$$

For our dataset,

For our dataset,

$$\phi\left(\begin{bmatrix}1\\1\end{bmatrix}\right)=\begin{bmatrix}1\\\sqrt{2}\\\sqrt{2}\\1\\1\\\sqrt{2}\end{bmatrix}\phi\left(\begin{bmatrix}-1\\-1\end{bmatrix}\right)=\begin{bmatrix}1\\-\sqrt{2}\\-\sqrt{2}\\1\\1\\\sqrt{2}\end{bmatrix}\quad\phi\left(\begin{bmatrix}-1\\1\end{bmatrix}\right)=\begin{bmatrix}1\\-\sqrt{2}\\\sqrt{2}\\1\\1\\-\sqrt{2}\end{bmatrix}\quad\phi\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)=\begin{bmatrix}1\\\sqrt{2}\\-\sqrt{2}\\1\\1\\-\sqrt{2}\end{bmatrix}$$

For the negative class:

$$\phi\left(\begin{bmatrix}1\\1\end{bmatrix}\right)_{2,6} = \begin{bmatrix}\sqrt{2}\\\sqrt{2}\end{bmatrix}$$

$$\phi\left(\begin{bmatrix}-1\\-1\end{bmatrix}\right)_{2,6} = \begin{bmatrix}-\sqrt{2}\\\sqrt{2}\end{bmatrix}$$

For the positive class:

$$\phi\left(\begin{bmatrix}-1\\1\end{bmatrix}\right)_{2,6} = \begin{bmatrix}-\sqrt{2}\\-\sqrt{2}\end{bmatrix}$$

$$\phi\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)_{2,6} = \begin{bmatrix}\sqrt{2}\\-\sqrt{2}\end{bmatrix}$$

For the negative class:

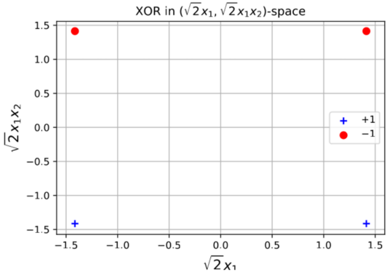
$$\phi\left(\begin{bmatrix}1\\1\end{bmatrix}\right)_{2,6} = \begin{bmatrix}\sqrt{2}\\\sqrt{2}\end{bmatrix}$$

$$\phi\left(\begin{bmatrix}-1\\-1\end{bmatrix}\right)_{2,6} = \begin{bmatrix}-\sqrt{2}\\\sqrt{2}\end{bmatrix}$$

For the positive class:

$$\phi\left(\begin{bmatrix}-1\\1\end{bmatrix}\right)_{2,6} = \begin{bmatrix}-\sqrt{2}\\-\sqrt{2}\end{bmatrix}$$

$$\phi\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)_{2,6} = \begin{bmatrix}\sqrt{2}\\-\sqrt{2}\end{bmatrix}$$



We may have a problem, recall the **dual perceptron**.

```
$converged \leftarrow 0$
while not $converged$ do
    $converged \leftarrow 1$
    for $x_i \in X, y_i \in y$ do
        if $y_i \sum_{j=1}^{N} \alpha_j y_j x_j \cdot x_i \leq 0$ then
            $\alpha_i \leftarrow \alpha_i + 1$
            $converged \leftarrow 0$
        end if
    end for
end while
```

We may have a problem, recall the **dual perceptron**.

```
$converged \leftarrow 0$
while not $converged$ do
    $converged \leftarrow 1$
    for $x_i \in X, y_i \in y$ do
        if $y_i \sum_{j=1}^{N} \alpha_j y_j \phi(x_j) \cdot \phi(x_i) \leq 0$ then
            $\alpha_i \leftarrow \alpha_i + 1$
            $converged \leftarrow 0$
        end if
    end for
end while
```

Recall the transformation $\phi : \mathbb{R}^p \rightarrow \mathbb{R}^k$.

Recall the transformation $\phi : \mathbb{R}^p \to \mathbb{R}^k$. For an arbitrarily large k,

$$G = \begin{bmatrix} \langle \phi(x_1), \phi(x_1) \rangle & \langle \phi(x_1), \phi(x_2) \rangle & \cdots & \langle \phi(x_1), x_n \rangle \\ \langle \phi(x_2), \phi(x_1) \rangle & \langle \phi(x_2), \phi(x_2) \rangle & \cdots & \langle \phi(x_2), \phi(x_n) \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \phi(x_n), \phi(x_1) \rangle & \langle \phi(x_n), \phi(x_2) \rangle & \cdots & \langle \phi(x_n), \phi(x_n) \rangle \end{bmatrix}$$

the Gram matrix becomes far too complex to compute.

The Kernel Trick

The Kernel Trick

An absolute mathemagical idea which allows us to calculate the values of the Gram matrix for cheap.

Recall the transformation to the XOR data:

$$\phi(\mathbf{x}) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{bmatrix}$$

The Kernel Trick

An absolute mathemagical idea which allows us to calculate the values of the Gram matrix for cheap.

Recall the transformation to the XOR data:

$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{bmatrix} \begin{bmatrix} 1 \\ \sqrt{2}y_1 \\ \sqrt{2}y_2 \\ y_1^2 \\ y_2^2 \\ \sqrt{2}y_1y_2 \end{bmatrix}$$

$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = 1 + 2x_1 y_1 + 2x_2 y_2 + x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 x_2 y_1 y_2$$

$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = 1 + 2x_1y_1 + 2x_2y_2 + x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2$$
$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = 1 + 2(x_1y_1 + x_2y_2) + (x_1y_1 + x_2y_2)^2$$

$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = 1 + 2x_1y_1 + 2x_2y_2 + x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2$$
$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = 1 + 2(x_1y_1 + x_2y_2) + (x_1y_1 + x_2y_2)^2$$
$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = (1 + \mathbf{x} \cdot \mathbf{y})^2$$

$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = 1 + 2x_1y_1 + 2x_2y_2 + x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2$$
$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = 1 + 2(x_1y_1 + x_2y_2) + (x_1y_1 + x_2y_2)^2$$
$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = (1 + \mathbf{x} \cdot \mathbf{y})^2$$

Say we define a *kernel*: $k(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x} \cdot \mathbf{y})^2$

$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = 1 + 2x_1y_1 + 2x_2y_2 + x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2$$
$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = 1 + 2(x_1y_1 + x_2y_2) + (x_1y_1 + x_2y_2)^2$$
$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = (1 + \mathbf{x} \cdot \mathbf{y})^2$$

Say we define a *kernel*: $k(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x} \cdot \mathbf{y})^2$

So our Gram matrix is:

$$G = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \cdots & k(x_n, x_n) \end{bmatrix}$$

$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = 1 + 2x_1y_1 + 2x_2y_2 + x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2$$
$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = 1 + 2(x_1y_1 + x_2y_2) + (x_1y_1 + x_2y_2)^2$$
$$\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = (1 + \mathbf{x} \cdot \mathbf{y})^2$$

Say we define a *kernel*: $k(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x} \cdot \mathbf{y})^2$

So our Gram matrix is:

$$G = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \cdots & k(x_n, x_n) \end{bmatrix}$$

Why is this useful?

Support Vector Machines