CS 412 — Introduction to Machine Learning (UIC)	March 06, 2025
Lecture 15	
Instructor: Aadirupa Saha	*Scribe(s): Rithish Reddy Chichili*

Overview

In the last lecture we covered the following topics:

- Soft-Margin SVM
- SVM Regression
- Perceptron

This lecture mainly focuses on the Perceptron algorithm, its theoretical properties, and its extensions to handle non-linearly separable data. The topics covered include:

- Perceptron Mistake Bounds
- Perceptron w/o perfect linear separator
- Kernel Perceptron

1 Online Classification Learning Setting

Before defining the Perceptron algorithm, we introduce the basic setting for **Online Classification Learning**.

1.1 Dataset Definition

We assume we have a dataset $\mathcal{D}$ consisting of N training examples:

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$$

where; each x_i is a feature vector in d-dimensional space:

$$x_i \in \mathbb{R}^d$$

Each label y_i belongs to $\{+1, -1\}$ (binary classification problem).

1.2 Goal of the Perceptron

The **goal** is to learn a function $f(x)$ that classifies points correctly:

$$f(x) = \text{sign}(w^T x + b)$$

where: w is the weight vector, b is the bias term, sign$(\cdot)$ outputs either $+1$ or -1 based on whether the input is positive or negative.

2 Perceptron Algorithm

The **Perceptron Algorithm** is an online learning algorithm that updates the weight vector whenever a misclassification occurs.

2.1 Initialization

We start by initializing the weight vector:

$$w_1 = 0 \in \mathbb{R}^d$$

This means that initially, we assume no prior knowledge about the classification boundary.

Algorithm Steps

For each time step $t = 1, 2, \ldots$, the perceptron follows these steps:

Algorithm 2.1:

Input: Data point from $\mathbb{R}^d$

Output: Weight w_{t+1}

Initialize a new data point $x_t \in \mathbb{R}^d$

$\hat{y}_t = \text{sign}(w_t^T x_t + b)$

if $\hat{y}_t == y_t$:

$w_{t+1} = w_t$

else:

$w_{t+1} = w_t + y_t x_t$

return w_{t+1}

Given for every data point(Input), update weights only if expected output is not equal to actual output.

3 Perceptron Learning Algorithm: Update Rule

In the previous section, we introduced the perceptron algorithm. Here, we describe the **update rule** when the perceptron makes a mistake.

3.1 Perceptron Update Rule

At each time step t, the algorithm makes a prediction:

$$\hat{y}_t = \text{sign}(w_t^T x_t + b)$$

If $\hat{y}_t = y_t$, the prediction is correct, and no update is needed. If $\hat{y}_t \neq y_t$, the prediction is incorrect, meaning the perceptron made a mistake. In this case, the weight update follows:

$$w_{t+1} = \begin{cases} w_t + x_t, & \text{if } y_t = +1 \\ w_t - x_t, & \text{if } y_t = -1 \end{cases}$$

This update rule ensures that misclassified points move the decision boundary in the correct direction.

4 Mistake Bound for Linearly Separable Data

4.1 Definition of Linear Separability

We now analyze the number of mistakes the perceptron makes before converging when the data is **linearly separable**.
Let the dataset $\mathcal{D}$ consist of N training samples:

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$$

where:$x_i \in \mathbb{R}^d$ represents feature vectors.$y_i \in \{+1, -1\}$ represents binary class labels.
A dataset is **linearly separable** if there exists a weight vector $w^* \in \mathbb{R}^d$ such that:

$$y_n(w^* x_n) \geq 0, \quad \forall n \in N$$

Here, w^* is called the **perfect separator**.

4.2 Definition of the Margin

If a dataset is linearly separable, we define the **margin** as:

$$\gamma = \min_{x_n \in \mathcal{D}} \frac{|w^* x_n|}{||x_n||}$$

The margin measures how well-separated the two classes are.A larger margin means fewer mistakes by the perceptron.

5 Perceptron Mistake Bound

In the previous section, we introduced the concept of a **linearly separable dataset** and the definition of the **margin** γ. Now, we discuss the **mistake bound theorem**, which establishes an upper limit on the number of mistakes the perceptron algorithm makes before converging.

Theorem 5.1: Perceptron Mistake Bound Theorem

The theorem states that if the dataset $\mathcal{D}$ is linearly separable with margin γ, then the number of mistakes the perceptron algorithm makes is at most:

$$M \leq \frac{1}{\gamma^2}$$

subject to:

$$\forall n, \quad ||x_n||_2 \leq 1, \quad ||w^*||_2 \leq 1$$

where:$||x_n||_2 \leq 1$ ensures that all input feature vectors are normalized.$||w^*||_2 \leq 1$ ensures that the optimal weight vector is bounded.
This bound shows that the perceptron **converges** in a finite number of updates.

Exercise 5.1: Proof of Generalization of the Theorem

In some cases, we relax the assumptions and obtain a more **general mistake bound**.
Practice Question: A more general statement of Theorem 1 assumes:

$$||w^*||_2 \leq 1, \quad ||x_n||_2 \leq 1, \quad \forall n \in [N]$$

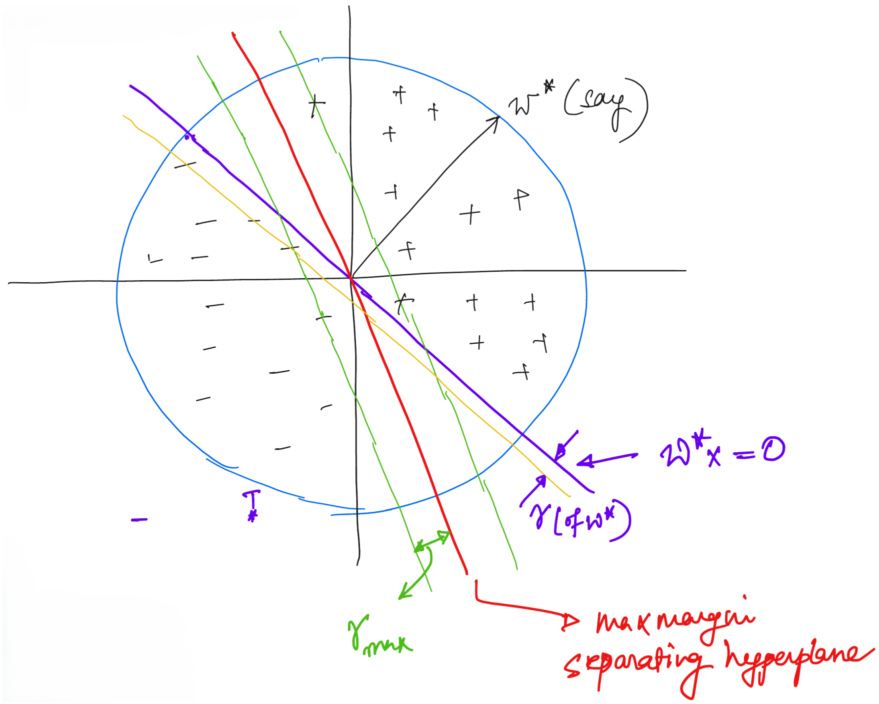
Using this assumption, the number of mistakes is bounded by:**(Try to prove this)**

$$M \leq \frac{||w^*||_2^2 \max ||x_n||_2^2}{\gamma^2}$$

This bound accounts for datasets where the feature vectors may have varying magnitudes.

Graphical Representation of the Margin

To illustrate the concept of **margin-based separation**, we present the following diagram:



Key Insights from the Diagram: The hyperplane $w^*x = 0$ is the **optimal separating boundary**.The **margin** γ represents the **minimum distance** between the closest point and the hyperplane.The **larger the margin**, the **fewer mistakes** the perceptron will make.The red, green, and purple lines represent **different possible hyperplanes**. The **maximum margin classifier** is the most robust.

Remarks on Perceptron Mistake Bound

Remark 1: Tightening the Bound

From the previous theorem, we introduced the perceptron mistake bound:

$$M \leq \frac{1}{\gamma^2}$$

where γ is the margin of the dataset. However, in practice, the margin γ can be upper-bounded by $\gamma_{\max}$, leading to:

$$\frac{1}{\gamma_{\max}^2} \leq \frac{1}{\gamma^2}$$

Thus, the theorem holds for any separating hyperplane $w^T x = 0$, but the **tightest bound** is achieved for the **maximum-margin classifier**.

Remark 2: The Role of Hinge Loss

When the dataset is not perfectly separable, we introduce **hinge loss**:

$$\xi_n = \max\left[0, \gamma - y_n(w^* x_n)\right]$$

where:ξ_n represents the **distance beyond the margin** for a given data point.In a **perfectly separable dataset**, $\xi_n = 0, \forall n$.Hinge loss is widely used in **Support Vector Machines (SVMs)**.

Proof of Perceptron Mistake Bound Theorem

Now, we analyze why the perceptron mistake bound holds by **examining weight updates over time**.

5.1 Update Rule and Alignment with w^*

Consider a time step t where the perceptron makes a mistake. The weight update rule states:

$$w_{t+1} = w_t + y_t x_t$$

Taking the **dot product** with the optimal weight vector w^*, we obtain:

$$w_{t+1}^T w^* = w_t^T w^* + y_t x_t^T w^*$$

Since we assume that the dataset is separable, we know:

$$y_t x_t^T w^* \geq \gamma$$

Thus, substituting this into our equation:

$$w_{t+1}^T w^* \geq w_t^T w^* + \gamma$$

Interpretation: This means that after each mistake, w_{t+1} is **more aligned** with w^*, since the inner product increases by at least γ. This ensures **convergence over time**.

5.2 Proof of Claim 1: Growth of $||w||$

To bound the number of mistakes, we analyze the norm of the weight vector.

$$||w_{t+1}||^2 = ||w_t + y_t x_t||^2$$

Expanding this:

$$||w_{t+1}||^2 = ||w_t||^2 + ||x_t||^2 + 2y_t w_t^T x_t$$

Since $||x_t||^2 = 1$ and $y_t w_t^T x_t < 0$ (because the perceptron made a mistake), we get:

$$||w_{t+1}||^2 \leq ||w_t||^2 + 1$$

Summing over all M mistakes:

$$||w_{M+1}||^2 \leq M$$

Further Implications of Claim 1

5.3 Extending Claim 1

From the previous analysis, we established:

$$w_{t+1}^T w^* \geq w_t^T w^* + \gamma$$

By applying this iteratively over multiple updates:

$$w_{t_M+1}^T w^* \geq w_{t_M}^T w^* + \gamma$$

Summing up over M mistakes:

$$w_{M+1}^T w^* \geq w_1^T w^* + M\gamma$$

Since we initialize with $w_1 = 0$, we obtain:

$$w_{M+1}^T w^* \geq M\gamma$$

This directly ties the number of mistakes M to the **margin** γ.

5.4 Bounding $||w||$: Claim 2

We now establish a second claim that helps us bound the norm of w.
Claim 2: The norm of the weight vector satisfies:

$$||w_{t+1}||^2 \leq ||w_t||^2 + 1$$

Proof: Expanding the norm squared after the weight update:

$$||w_{t+1}||^2 = ||w_t + y_t x_t||^2$$

Expanding using the squared norm property:

$$||w_{t+1}||^2 = ||w_t||^2 + ||x_t||^2 + 2y_t w_t^T x_t$$

Since $||x_t||^2 = 1$ and we know that $2y_t w_t^T x_t < 0$ due to a mistake, we get:

$$||w_{t+1}||^2 \leq ||w_t||^2 + 1$$

Summing over all M mistakes:

$$||w_{M+1}||^2 \leq M$$

5.5 Final Combination of Bounds

Now, combining Claim 1 and Claim 2, we obtain:

$$M\gamma \leq ||w_{M+1}|| \cdot ||w^*||$$

Using our previous bound:

$$||w_{M+1}|| \leq \sqrt{M}$$

Since $||w^*|| = 1$, we get:

$$M\gamma \leq \sqrt{M}$$

Rearranging:

$$\sqrt{M} \leq \frac{1}{\gamma}$$

Squaring both sides:

$$M \leq \frac{1}{\gamma^2}$$

Thus, we have **proved** the perceptron mistake bound theorem.

Final Steps in Proof of Perceptron Mistake Bound

5.6 Bounding $||w||$ Further

We continue from the previous derivation, where we established:

$$||w_{t_M+1}||^2 \leq ||w_{t_M}||^2 + 1$$

Applying this iteratively over M mistakes:

$$||w_{t_M+1}||^2 \leq ||w_{t_M-1}||^2 + 1 + 1$$

Continuing the expansion:

$$||w_{t_M+1}||^2 \leq ||w_1||^2 + M$$

Since we initialized with $w_1 = 0$, we obtain:

$$||w_{t_M+1}||^2 \leq M$$

Taking the square root on both sides:

$$||w_{t_M+1}|| \leq \sqrt{M}$$

5.7 Final Combination of Claims 1 and 2

From our earlier derivation, we had established:

$$M\gamma \leq w_{t_M+1}^T w^*$$

Applying the Cauchy-Schwarz inequality:

$$w_{t_M+1}^T w^* \leq ||w_{t_M+1}|| \cdot ||w^*||$$

Since $||w^*|| = 1$, we simplify:

$$M\gamma \leq ||w_{t_M+1}||$$

Using our previous bound:

$$||w_{t_M+1}|| \leq \sqrt{M}$$

we substitute:

$$M\gamma \leq \sqrt{M}$$

5.8 Final Derivation of the Mistake Bound

Dividing both sides by γ:

$$\sqrt{M} \leq \frac{1}{\gamma}$$

Squaring both sides:

$$M \leq \frac{1}{\gamma^2}$$

Thus, we have proven that the **number of mistakes the perceptron makes is upper-bounded by**:

$$M \leq \frac{1}{\gamma^2}$$

6 Handling Non-Linearly Separable Data

In previous sections, we analyzed the perceptron algorithm under the assumption that the dataset is **linearly separable**. However, in real-world scenarios, data may not be perfectly separable by a single hyperplane.

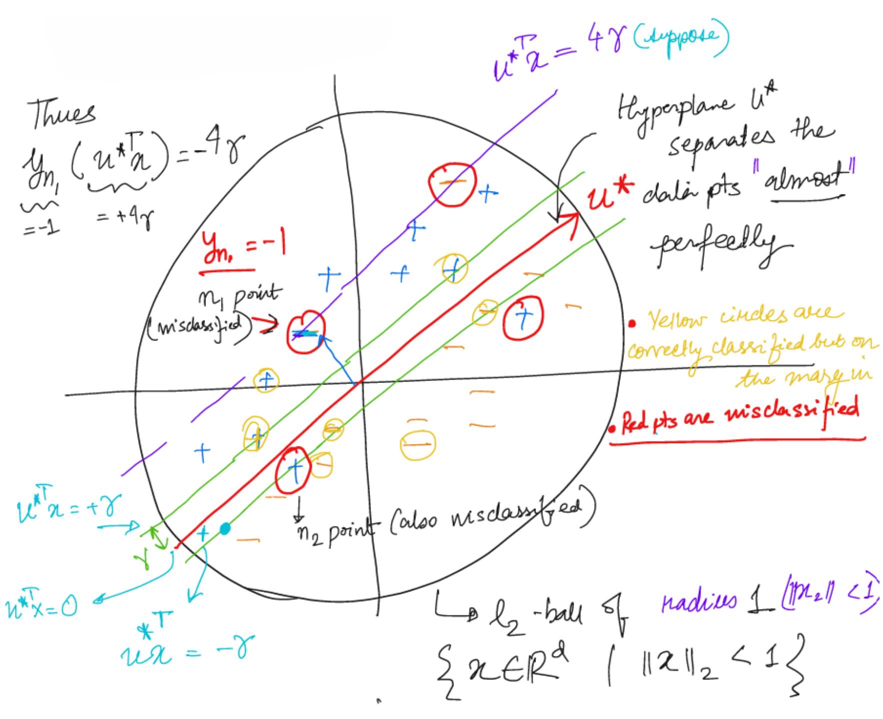
6.1 Understanding the Non-Linearly Seperable Case

Definition: If a dataset contains overlapping points from different classes, no single hyperplane can separate them perfectly. Instead, we aim to find a **best possible separating hyperplane**.
Let u^* be the optimal separating hyperplane, which "almost" separates the data:

$$u^{*T} x = 4\gamma$$

where:γ represents the margin.$u^* x = 0$ defines the **decision boundary**.Some points lie on the correct side but within the margin, while others are misclassified.

Graphical Interpretation



Key Observations from the Diagram: The **green hyperplane** represents a possible separating boundary.The **red circles** denote misclassified points.**Yellow points** are correctly classified but lie within the margin.The margin defines an L_2-ball of radius 1:

$$\{x \in \mathbb{R}^d \mid ||x||_2 \leq 1\}$$

6.2 Misclassification Error in Terms of Hinge Loss

Since some points are misclassified, we introduce **hinge loss** to quantify the error:

$$\xi_n = \max\Big[0, \gamma - y_n(u^* x_n)\Big], \quad \forall n \in [N]$$

where:ξ_n represents how far a point deviates from correct classification.If $u^* x_n$ satisfies $y_n(u^* x_n) \geq \gamma$, then $\xi_n = 0$ (correct classification).If $y_n(u^* x_n) < \gamma$, then $\xi_n > 0$, indicating misclassification.

6.3 Example Calculation

Consider a misclassified point x_1, where:

$$y_n(u^* x_n) = -4\gamma$$

Then, the hinge loss is:

$$\xi_n = \max\Big[0, \gamma - (-4\gamma)\Big] = 5\gamma$$

This shows that the hinge loss increases as the classification margin decreases.

Further Analysis of Hinge Loss and Mistake Bound

6.4 Detailed Computation of Hinge Loss

In the previous section, we introduced the concept of **hinge loss**, which accounts for points that are either misclassified or fall inside the margin.
The hinge loss for a given point x_n is defined as:

$$\xi_n = \max\Big[0, \gamma - y_n(u^* x_n)\Big]$$

We now analyze this for specific points.
Case 1: Consider the first misclassified point x_1, where:

$$y_1(u^* x_1) = -4\gamma$$

Then, the hinge loss is:

$$\xi_1 = \max\Big[0, \gamma - (-4\gamma)\Big] = \max[0, 5\gamma] = 5\gamma$$

Case 2: For another misclassified point x_2, we have:

$$y_2(u^* x_2) = -\gamma$$

Then, its hinge loss is:

$$\xi_2 = \max\Big[0, \gamma - (-\gamma)\Big] = \max[0, 2\gamma] = 2\gamma$$

Remark: If the classifier u^* were a **perfect separator** with margin γ, then:

$$\xi_n = 0, \quad \forall n \in [N]$$

which implies no misclassification.

6.5 Mistake Bound for linearly Non-Separable Data

We now extend the perceptron mistake bound to the case where the data is not perfectly separable.

Theorem 6.1: Mistake Bound for linearly Non-Separable Case

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ be a dataset where:

- $x_n \in \mathbb{R}^d$, $y_n \in \{+1, -1\}$.
- $||x_n|| \leq 1$, meaning all data points are bounded.
- There exists a hyperplane $u^* \in \mathbb{R}^d$ such that:

  $$\xi_n = \max\Big[0, \gamma - y_n(u^* x_n)\Big]$$

  represents the hinge loss for each point.

Implication: The hinge loss measures how well-separated the data points are with respect to the classifier $u^* x = 0$.

Bounding the Number of Mistakes

The number of mistakes made by the perceptron algorithm depends on the hinge loss across all datapoints:

$$\sum_{n=1}^{N} \xi_n$$

The bound on the total number of mistakes is given by:

$$M \leq \frac{1}{\gamma^2} + 2\sum_{n=1}^{N} \frac{\xi_n}{\gamma}$$

where: $\frac{1}{\gamma^2}$ is the mistake bound in the separable case. $\sum \frac{\xi_n}{\gamma}$ accounts for errors due to misclassified points.

General Perceptron Mistake Bound for linearly Non-Separable Data

6.6 Total Hinge Loss and Perceptron Mistake Bound

Previously, we established the mistake bound for separable data. However, for **linearly non-separable cases**, the number of mistakes made by the perceptron algorithm can be related to the total hinge loss over the dataset.
Key insights: If the dataset is perfectly separable, the total hinge loss is:

$$\sum_{t=1}^{M} \xi_t = 0$$

leading to an upper bound of:

$$M \leq \frac{1}{\gamma^2}$$

If the dataset is **not** perfectly separable, the hinge loss contributes additional errors:

$$M \leq \frac{1}{\gamma^2} + 2\sum_{n=1}^{N} \frac{\xi_n}{\gamma}$$

6.7 Interpretation of the Mistake Bound

The mistake bound provides an upper limit on the number of times the perceptron algorithm updates its weight vector before it stops making mistakes.
Perfect separation ($\xi_n = 0$): The perceptron will correctly classify all points eventually. **Linearly Non-separable case** ($\xi_n > 0$): The additional hinge loss term accounts for mistakes.
Thus, the total hinge loss over all N points in $\mathcal{D}$ affects the total mistake bound.

6.8 Key Observations

If a datapoint x_n was misclassified by u^* (i.e., $\xi_n > 0$), then it is also likely to be misclassified by the perceptron algorithm. The perceptron algorithm will eventually learn to classify correctly if $\xi_n = 0$ for all n.

6.9 Practice Problem

Exercise 6.1: Proof of Mistake Bound for Linearly Non-Seperable Case Theorem

Exercise: Complete the proof of Mistake Bound for Linearly Non-Seperable Case Theorem by showing that:

$$M \leq \frac{1}{\gamma^2} + 2\sum_{n=1}^{N} \frac{\xi_n}{\gamma}$$

Hint:

- Consider the weight update rule at each iteration.
- Use Claim 1: If the perceptron makes a mistake at time step t, then;

$$w_{t+1}^T u^* \geq w_t^T u^* + \gamma - \xi_t$$

- Use Claim 2: Bounding the norm of w;

$$||w_{t+1}||^2 \leq ||w_t||^2 + 1$$

- Analyze the cumulative sum over M mistakes.

6.10 Key Takeaways

The mistake bound increases as a function of hinge loss. If the dataset is nearly separable, the number of mistakes remains close to $\frac{1}{\gamma^2}$. If hinge loss is large, mistakes increase.

7 Kernel Perceptron

7.1 Motivation for Non-Linear Classification

The perceptron algorithm works well for linearly separable data, but what if the data is **not** linearly separable?
Question: How can we modify the perceptron algorithm in another way to handle **non-linearly separable** data?

7.2 Feature Transformation

One approach is to **map** data from a low-dimensional space $\mathbb{R}^d$ to a higher-dimensional space $\mathbb{R}^D$ where a separating hyperplane exists.
Let:

$$\varphi : \mathbb{R}^d \to \mathbb{R}^D, \quad \text{where } D >> d$$

We seek a hyperplane that separates the transformed data:

$$w^*\varphi(x) = 0$$

Key Insight: Instead of working directly in high dimensions, we use the **kernel trick** to compute inner products efficiently.

$$K_\varphi(x_i, x_j) = \varphi(x_i)^T\varphi(x_j)$$

This avoids explicitly computing $\varphi(x)$, reducing computational complexity.

8 Kernel Trick: Transforming to Higher Dimensions

8.1 Motivation for Feature Mapping

The perceptron algorithm in its standard form can only handle linearly separable data. However, real-world data is often **not linearly separable** in the given feature space.
Solution: We map the input data from a low-dimensional space $\mathbb{R}^d$ to a higher-dimensional space $\mathbb{R}^D$, where a linear separation becomes possible.

$$\varphi : \mathbb{R}^d \to \mathbb{R}^D, \quad \text{where } D >> d$$

This transformation allows us to find a new hyperplane in the transformed space:

$$w^T\varphi(x) = 0$$

8.2 Formal Definition of Feature Mapping

Consider a dataset:

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$$

where $x_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$.
The goal is to find a **non-linear mapping** $\varphi(x)$ such that the transformed data is linearly separable by a hyperplane:

$$w^T\varphi(x) = 0$$

8.3 Kernel Function and Inner Product Trick

Instead of explicitly computing the feature transformation $\varphi(x)$, we use the **kernel function**:

$$K_\varphi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$$

where:

$$K_{\varphi}(x_1, x_2) = \varphi(x_1)^T \varphi(x_2)$$

This allows us to work in the higher-dimensional space **without explicitly computing** $\varphi(x)$, which avoids computational inefficiency.

8.4 Key Question: Can We Use the Classifier Without Knowing φ?

A key insight from the kernel trick is that we can compute **decision boundaries** in the transformed space without explicitly calculating $\varphi(x)$. The decision rule is:

$$\hat{y} = \text{sign}\Big(\sum_{t=1}^{T} y_t K_{\varphi}(x_t, x)\Big)$$

This means that **even if we do not explicitly know** $\varphi(x)$, we can still classify data using just the **kernel function**.

Kernel Perceptron Algorithm

8.5 Weight Vector Representation

In the standard perceptron, the weight vector is updated using:

$$w_{T+1} = \sum_{t=1}^{T} y_t x_t + w_1$$

where the sum represents all misclassified points that contributed to the final weight vector. However, in the **kernel perceptron**, we extend this idea to **feature space**:

$$w_{T+1} = \sum_{t=1}^{T} y_t \varphi(x_t) + w_1$$

where $\varphi(x)$ is a feature transformation that maps the input to a higher-dimensional space.

8.6 Prediction Rule in Kernel Perceptron

The perceptron predicts a label $\hat{y}(x)$ based on:

$$\hat{y}(x) = \text{sign}\Big(w_{T+1}^T \varphi(x)\Big)$$

Substituting the weight representation:

$$\hat{y}(x) = \text{sign}\Big(\sum_{t=1}^{T} y_t \varphi(x_t)^T \varphi(x)\Big)$$

Since computing $\varphi(x)$ explicitly can be computationally expensive, we leverage the **kernel function**:

$$\hat{y}(x) = \text{sign}\Big(\sum_{t=1}^{T} y_t K_\varphi(x_t, x)\Big)$$

where $K_\varphi(x_t, x) = \varphi(x_t)^T \varphi(x)$ computes the inner product in the **higher-dimensional feature space**.

8.7 Intuition Behind Kernel Perceptron

In classical perceptron, the decision boundary is learned directly in the **original feature space**.In **kernel perceptron**, the decision boundary is **implicitly computed** in a transformed high-dimensional space without explicitly computing $\varphi(x)$.This makes it feasible to learn non-linear decision boundaries **efficiently**.
Example Scenario: Suppose we receive a new data point $x \in \mathbb{R}^d$: 1. **Step 1**: Map the input x to a high-dimensional feature space using $\varphi(x)$. 2. **Step 2**: Compute the prediction using the **kernel function** instead of explicitly computing $\varphi(x)$.
Key Benefit: We never need to explicitly compute $\varphi(x)$, making kernel perceptron an **efficient** method for handling non-linearly separable data.

Final Remarks on Kernel Perceptron

8.8 Efficient Classification using Kernel Function

Once the **Kernel Perceptron** has been trained, predicting the label of a new point x can be efficiently performed using:

$$\hat{y}(x) = \text{sign}\left(\sum_{t=1}^{T} y_t K_\varphi(x_t, x)\right)$$

where $K_\varphi(x_t, x)$ is the **kernel function**, which computes the similarity between a training example x_t and the new data point x.

8.9 Key Observation

The **remarkable advantage** of the kernel perceptron is:

We can make predictions without explicitly computing $\varphi(x)$, thanks to the kernel function!

Mathematically, this means:

$$\hat{y}(x) = \text{sign}\left(\sum_{t=1}^{T} y_t \varphi(x_t)^T \varphi(x)\right) = \text{sign}\left(\sum_{t=1}^{T} y_t K_\varphi(x_t, x)\right)$$

Thus, we can use **Kernel Methods** to implicitly work in **high-dimensional spaces** without computing the transformation $\varphi(x)$ explicitly.

8.10 Practical Benefits of Kernel Perceptron

Avoids Curse of Dimensionality: Instead of computing high-dimensional transformations $\varphi(x)$, we only evaluate kernel functions.**Efficient Training and Prediction**: Predictions rely only on support vectors (misclassified points).**Handles Non-Linear Data**: By using appropriate kernels (e.g., polynomial or Gaussian), the perceptron can classify non-linearly separable data.

Next Lecture:

- **Perceptron as single-layer NN**
- **Winnow's Algorithm**
- **Learning from expert advice**

References

- **Probabilistic Machine Learning-An Introduction by Kevin P. Murphy (Sections 10.2.5,13.2)**
- **Machine Learning Basics by Yingyu Liang (Lecture 3: Perceptron)**