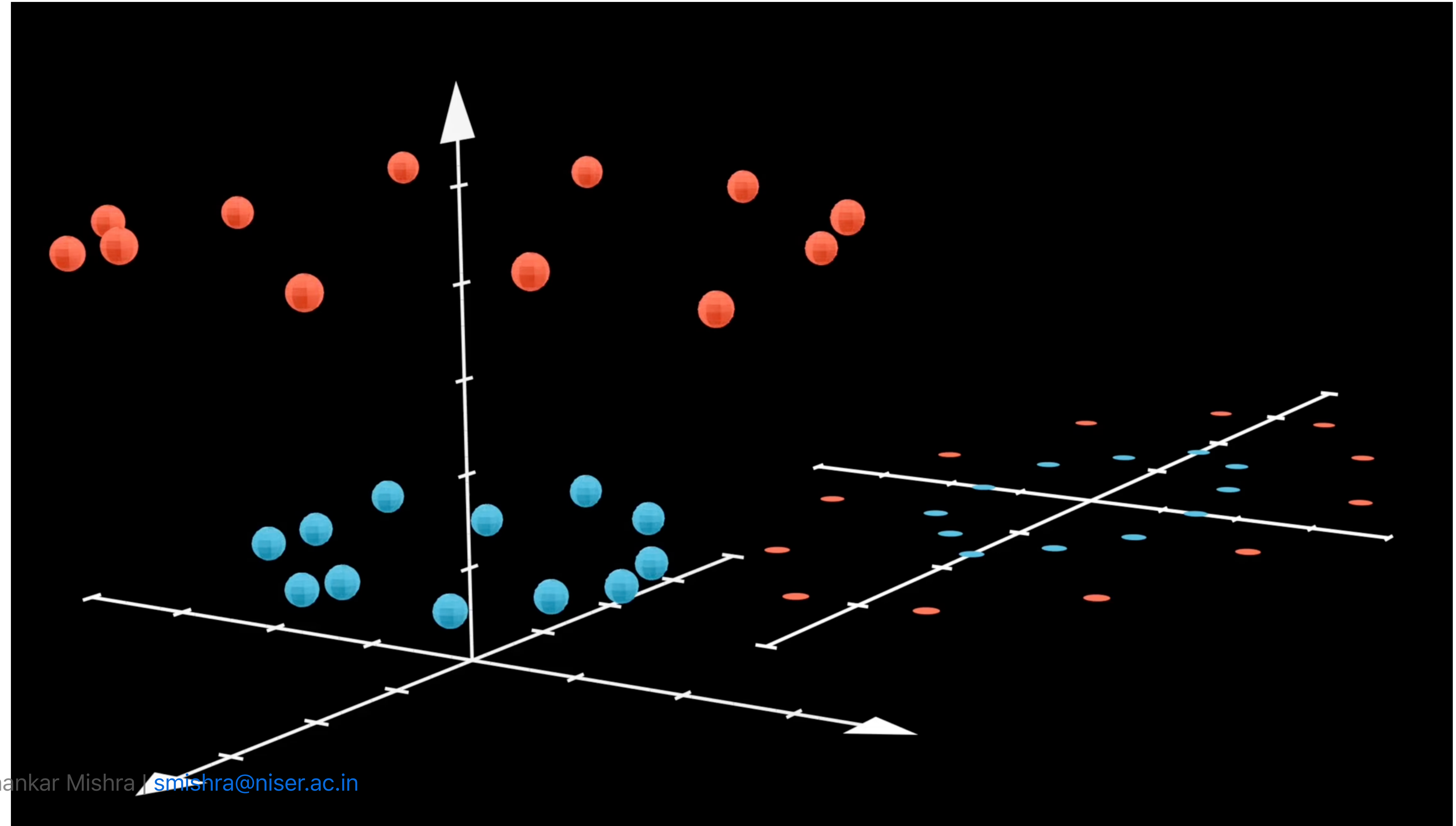
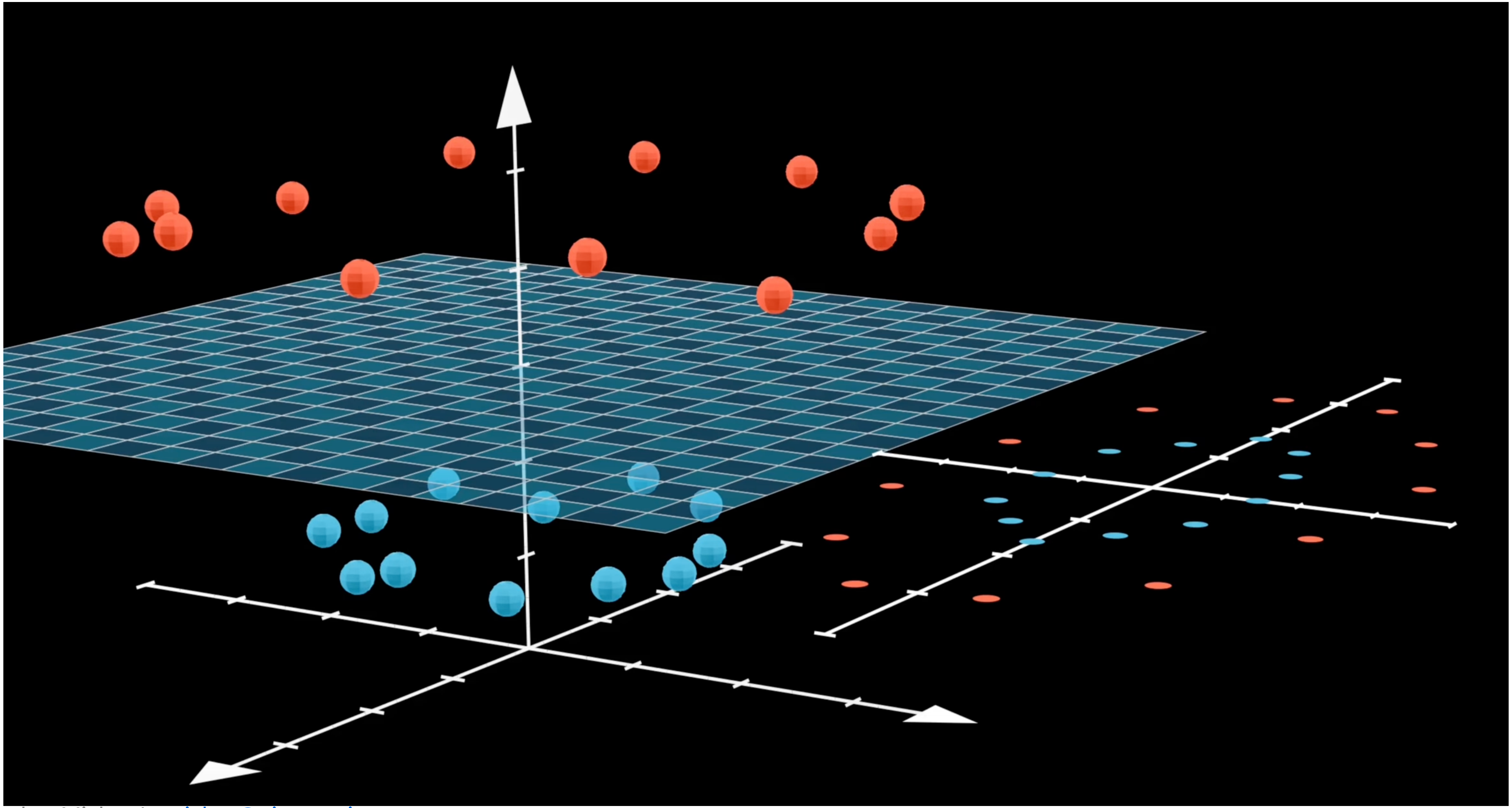
SVM Recap

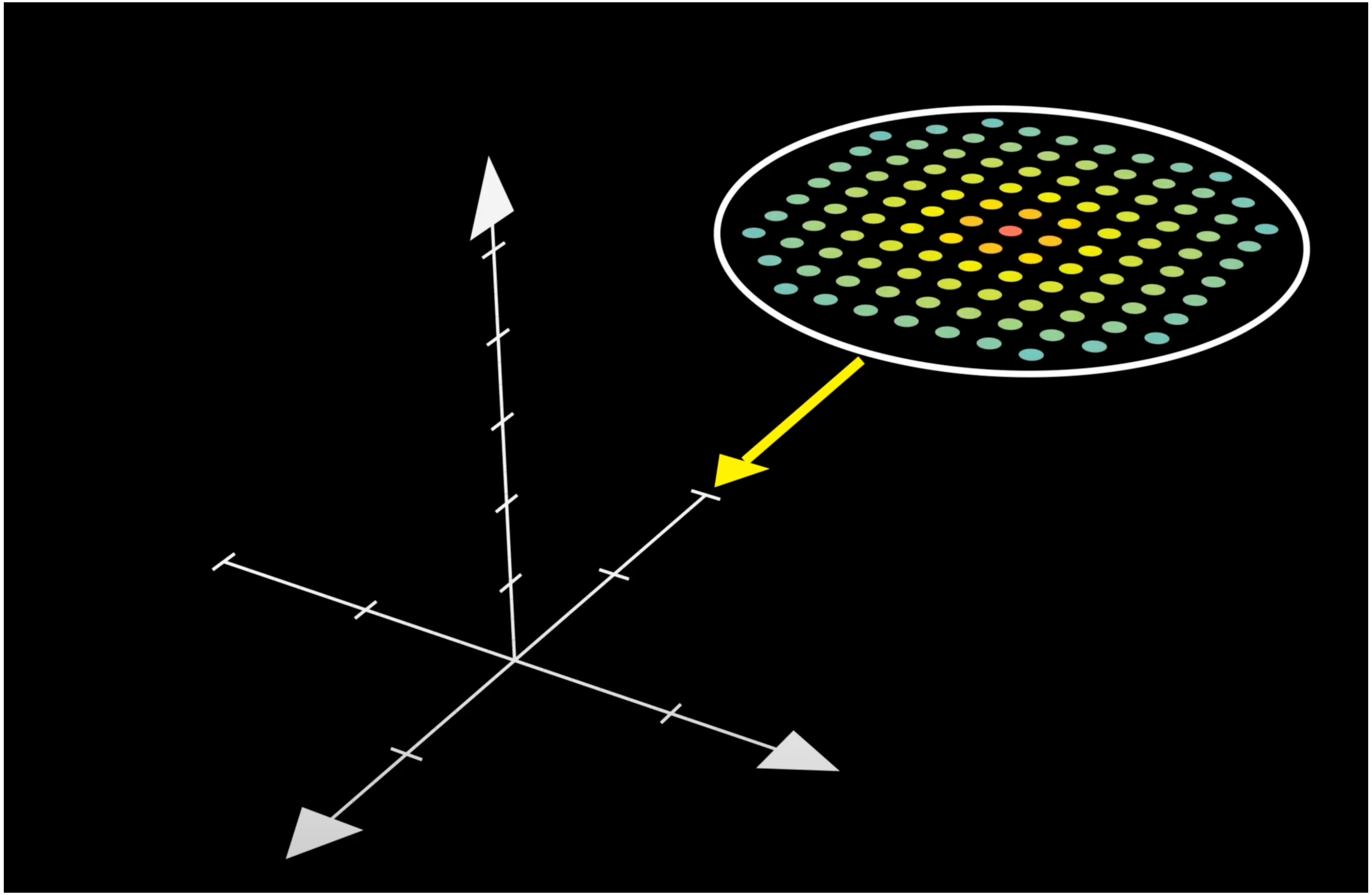
Subhankar Mishra | smishra@niser.ac.in

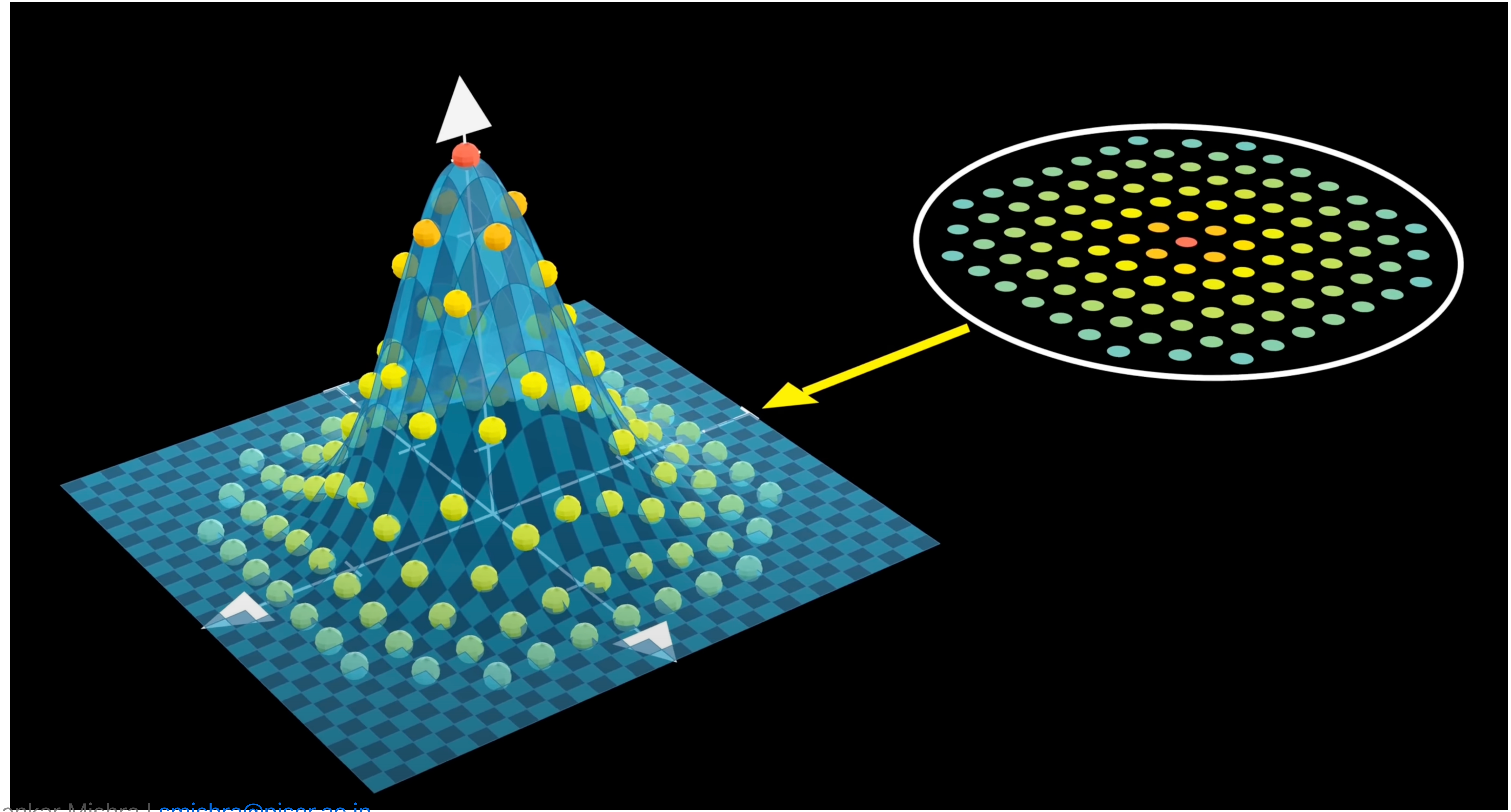




RBF Kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$





3.5 k-Nearest Neighbors (kNN)

A Non-parametric Learning Algorithm

What is k-Nearest Neighbors?

- **Non-parametric learning algorithm**
- **Instance-based learning** (lazy learning)
- Keeps all training examples in memory
- No explicit model building phase
- Makes predictions based on similarity to training examples

Key Characteristic: Unlike other algorithms that discard training data after model building, kNN retains all training data for prediction.

The kNN Learning Principle

Inductive Bias: Nearby Points Have Similar Labels

Consider a test point among training examples (positive and negative). Most likely, you'd predict the label based on nearby points.

kNN's Core Assumption: Examples that are close in feature space should have similar labels.

The Algorithm:

1. **Training**: Simply store all training examples
2. **Prediction**: Find the training example x most similar to test example $\hat{x}$
3. **Decision**: Predict the same label as the nearest neighbor

Simple yet effective! But prone to noise...

The Problem with k=1

Issue: Overfitting to Label Noise

- Single nearest neighbor can be an outlier
- Cannot consider "preponderance of evidence"
- Makes unnecessary errors due to noisy labels

Solution: Use Multiple Neighbors (k > 1)

- Consider k nearest neighbors
- Let them **vote** on the correct class
- Majority wins!

Example: For k=3, if 2 neighbors are positive and 1 is negative, predict positive.

kNN Algorithm Pseudocode

```
Algorithm: KNN-Predict(D, K, x̂)
S ← []                                // empty list
for n = 1 to N do
  S ← S ⊕ ⟨d(xn, x̂), n⟩              // store distance to training example n
end for
S ← sort(S)                           // put lowest-distance objects first
ŷ ← 0
for k = 1 to K do
  ⟨dist, n⟩ ← Sk                      // kth closest data point
  ŷ ← ŷ + yn                          // vote according to label
end for
return sign(ŷ)                        // return +1 if ŷ > 0, −1 if ŷ < 0
```

Note: No explicit training phase - just store the data!

Geometric Intuition

Thinking of Examples as Vectors

The biggest advantage of representing examples as vectors in high-dimensional space is that it allows us to apply **geometric concepts** to machine learning.

Example: Distance between $\langle 2, 3 \rangle$ and $\langle 6, 1 \rangle$:

```
d = √[(2−6)² + (3−1)²] = √[16 + 4] = √20 ≈ 4.24
```

General Euclidean Distance in D dimensions:

```
d(a, b) = √[Σ(ai − bi)²]
```

This geometric thinking enables us to measure similarity and make predictions based on proximity.

Distance Functions

The **closeness** of two points is measured by a distance function:

1. Euclidean Distance

```
d(xi, xj) = √(Σ(xi^(l) − xj^(l))²)
```

2. Cosine Similarity

```
s(xi, xk) = cos(θ(xi, xk)) = (Σxi^(j) ∗ xk^(j)) / (√Σ(xi^(j))² ∗ √Σ(xk^(j))²)
```

Cosine Distance = -1 × cosine similarity

Distance Functions

Other Common Metrics:

- **Manhattan Distance**: Sum of absolute differences
- **Chebyshev Distance**: Maximum difference across dimensions
- **Hamming Distance**: For categorical data

Choosing K: The Bias-Variance Trade-off

The Big Question: How to Choose K?

- **K = 1**: Risk of overfitting to noise
- **K = N**: Always predicts majority class (underfitting)
- **K is a hyperparameter** that controls bias-variance trade-off

Guidelines:

- **Small K**: More sensitive to noise, complex decision boundaries
- **Large K**: Smoother decision boundaries, may miss local patterns
- **Use cross-validation** to find optimal K
- **Try odd values** to avoid ties in voting

kNN's Inductive Bias & Feature Problems

Core Assumptions:

1. **Locality**: Nearby points should have the same label
2. **Feature Equality**: All features are equally important!

kNN's Inductive Bias & Feature Problems

The Feature Scale Problem:

Classifying Cricket Players by Position

- Features: Age (years) and Annual Income (₹ lakhs)
- Age: 18-40 years (small range)
- Income: 50-15,000 lakhs (huge range!)
- Distance dominated by income differences
- Age becomes irrelevant despite being important for position classification

Without normalization: A 25-year-old earning ₹500 lakhs appears "closer" to a 35-year-old earning ₹600 lakhs than to a 24-year-old earning ₹100 lakhs, even though age might be more relevant for playing position!

kNN's Inductive Bias & Feature Problems

Contrast with Decision Trees:

- **Decision Trees:** Find the most useful features
- **kNN:** Uses every feature equally, ignores importance

Implication: kNN struggles with irrelevant features and different scales

kNN: Advantages & Disadvantages

Advantages:

- **Simple** to understand and implement
- **No assumptions** about data distribution
- **Effective** for non-linear problems
- **Locally adaptive:** Decision boundaries follow data density

Disadvantages:

- **Memory intensive:** Stores all training data
- **Computationally expensive:** Distance calculations for every prediction
- **Sensitive** to irrelevant features (curse of dimensionality)
- **Sensitive to feature scale:** Requires preprocessing

kNN: Fails

When kNN Fails:

- **High-dimensional data**: All points become equidistant
- **Irrelevant features**: Noise dominates signal
- **Large datasets**: Memory and computation challenges

Practical Guidelines

When to use kNN:

- Small to medium datasets
- Non-linear relationships
- Local patterns important
- No clear parametric model

Essential Preprocessing:

- **Feature scaling**: Normalize/standardize features
- **Feature selection**: Remove irrelevant features
- **Dimensionality reduction**: PCA for high dimensions

Practical Guidelines

Choosing k:

- **Use cross-validation** to find optimal k
- **Try odd values** to avoid ties in voting
- **Small k:** Complex boundaries, sensitive to noise
- **Large k:** Smooth boundaries, may miss patterns

Summary & Key Takeaways

kNN in a Nutshell:

- **Non-parametric**: No assumptions about data distribution
- **Instance-based**: Uses all training data for prediction
- **Distance-dependent**: Choice of metric is crucial
- **Locally adaptive**: Decision boundaries adapt to data density

Critical Success Factors:

1. **Proper feature scaling** (normalize/standardize)
2. **Appropriate k selection** (use cross-validation)
3. **Relevant distance metric** for your data type
4. **Feature selection** to remove noise

Summary & Key Takeaways

Implementation Optimizations:

- **KD-trees/Ball trees**: Fast neighbor search
- **Approximate methods**: Trade accuracy for speed

Simple yet powerful, but preprocessing and hyperparameter tuning are essential!

References

For SVM Pictures

RBF Kernel Explained: Mapping Data to Infinite Dimensions