SMAI Quiz Notes

Lec 1-7

Recognition:

→ Identification of a pattern as member of known category

Problem formulations:

$y = f(X)$

- y ← Class Label (Predicted)
- f → Classifies
- X → Input feature vector

Pattern:

Opposite of chaos, vaguely defined

Pattern class:

→ Collection of similar objects

→ Class defined by samples of class

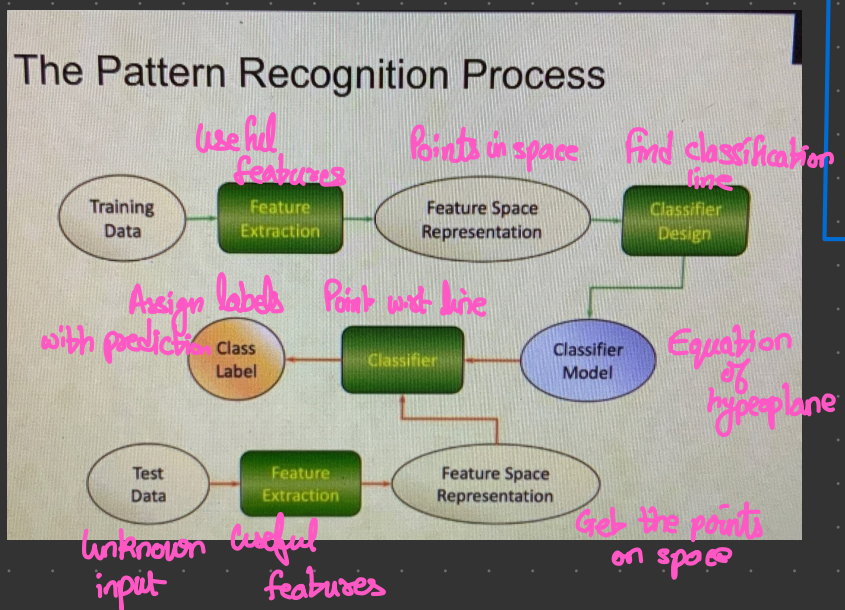
- Inter-class variability: Among groups
- Intra-class variability: Within groups

Examples: Trees

Pattern Classification

→ Remove irrelevant info

→ Feature extraction - Extracting useful info. from raw data



Pattern Recognition:

→ Learns to classify

→ Cognition: Forming new classes

Recognition: Known classes

Approaches to recognition:

Patterns represented in feature space

↳ Learn separating boundary b/w classes

↳ Learn stat. model for pattern generation in feature space

Nearest Neighbor Classifier:

→ Assign label of sample nearest (that's) to test sample

→

Voronoi Tesselation:

→ Boundary b/w every point
→ Then ignore those surrounded by same
→ Decision boundary is piecewise linear

Summary

Nearest Neighbor is a simple yet effective classifier

Error rate (with increasing number of training samples) is at-most twice that of the ideal classifier

Easy to implement, Training is trivial

Easy to interpret

Time complexity is high during testing
- Can be improved in different ways

K-Nearest Neighbor

→ Assign test sample label that is most frequent amongst K-Nearest neighbors
→ K-parameter that's user selectable
→ Output label can change with change of K

Weighted K-Nearest Neighbor: (read this part)

→ If there's a tie, randomly break it
→ Or assign weights

Summary

- Time complexity of Nearest Neighbor is high during testing
  - Can be improved in different ways
- Very popular classifier due to
  - Simplicity
  - Explainability
  - Robustness (fewer assumptions)
  - Ability to do continuous learning
  - Guarantees on Error Rate

Sample Pruning:

→ Remove samples that don't affect decision boundary
→ To identify samples to be pruned: (options)
  ↳ One per class
    ↳ Class means

↑
Nearest mean classifier
(Equivalent to 1-Nearest Neighbor)

Lecture 3

Linear decision boundary:

→ Straight line separating 2 classes

$y = mx + c$

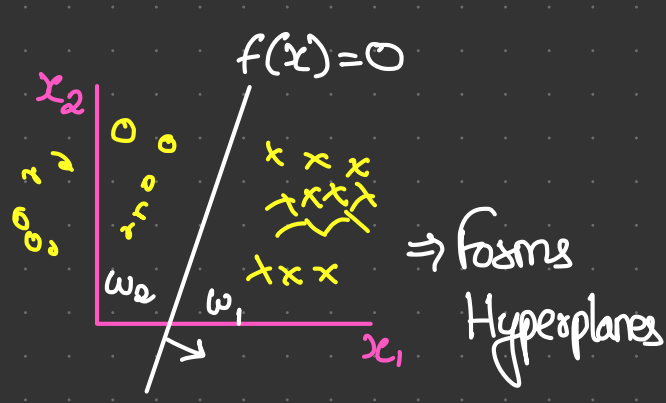
↑ Vector

→ Generalised eqn:

$w_0 + w_1 x_1 + w_2 x_2 \dots + w_d x_d = 0$

$\Rightarrow f(x) = 0$

-ve class ω_2 | +ve class ω_1



⇒ Forms Hyperplanes

Goal: Learn w_i's to draw decision rule

→ For 2-class,

$f(x) = w_0 + w_1 x_1 + w_2 x_2 = w \cdot x$ (or) $w^T x$

$w = [w_0\ w_1\ w_2]^T \quad x = [1\ x_1\ x_2]^T$

Decision boundary: $w^T x = 0$

Learn $d+1$ values of w_i

Summary:

→ Linear decision boundary are popular in ML due to:

→ Simplicity of model

→ Theoritical result

→ Simplicity of computation

Gradient Descent & Perceptron Learning Algorithm

To learn decision boundary:

→ Randomly initialise w_i's

→ Iteratively modify w_i so that each sample is correctly classified

To update automatically →

→ Define Loss function $J(w)$ which will be minimum when training samples are correctly classified

$\nabla J = \frac{\partial J}{\partial w}$ ← Gradient Vector

Gradient Descent Algorithm:

1. Randomly initialise elements of w: w^0
2. Compute gradient of J at w: $\partial J / \partial w$
3. $w^{t+1} = w^t - \eta \frac{\partial J}{\partial w}$
4. Repeat 2 & 3 steps until convergence ($\eta \frac{\partial J}{\partial w}$ becomes small)

Loss functions for GD:

1. No. of misclassified samples
   ↳ Not differentiable
2. $w^T x_i > 0$ for +ve class
   < 0 for −ve class
   $y_i (w^T x_i) > 0$ for all samples
   $J(x) = -y_i \cdot w^T x_i$
   $\nabla J = \frac{\partial J}{\partial w} = -y_i x_i$

→ Perceptron update rule

To compute J:
- → Per training sample ⇒ Single sample or Stochastic GD
- → Summed up over batch ⇒ Batch GD
- → In between ⇒ Mini-Batch GD

Batch perceptron rule:

$$J_p(x) = \sum_{x_i \in X} -y_i w^T x_i$$

↳ set of misclassified samples

→ Perceptron Loss Fn

$$\Rightarrow \nabla J_p = \sum_{x_i \in X} -y_i x_i \Rightarrow w^{t+1} = w^t + \eta \sum_{x_i \in X} y_i x_i$$

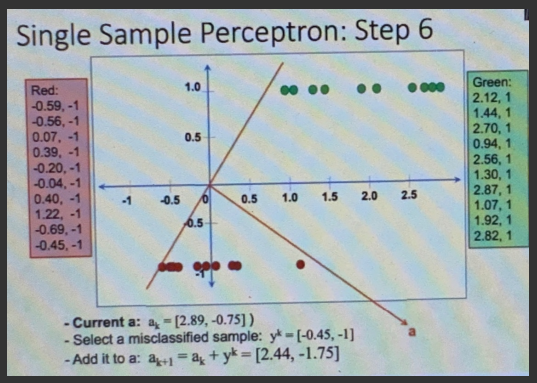
Summary

- Linear Decision boundaries are easily expressed as dot product with the co-efficient vector
- Gradient Descent allows us to learn the decision boundary from a set of labelled training samples
- Several variants of GD based on the loss function used
  - Perceptron learning algorithm is one such

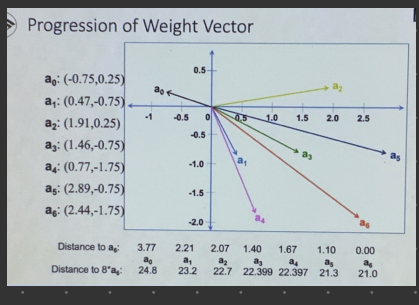
Lecture 4 (Revise perceptron loss fn)

Single Sample Perceptron Training

Steps: 1) Augment $[x, 1]$
2) Assign $[\pm x, \pm 1]$ based on label
3) Randomly initialise weight (a_k)
   ↳ Select misclassified sample (y^k)
   ↳ Add $a_{k+1} = a_k + y^k$
4) Finally $a_k = [a_1, a_2]$
   $a_1 x - a_2 = 0$ is solution



NOTE: Weight vector taken together when multiplied with particular constant is arranged in descending order



Relaxation Algorithms:

$$J_p(\omega) = \sum_{x_i \in X} (\omega^T x_i)^2$$ [Augmented & sign multiplied]

→ Smoother
→ Often too smooth to converge to boundary
→ Dominated by large vectors
→ Solutions: Norm. & add margin

$$J_q(\omega) = \frac{1}{2} \sum_{x_i \in X} \frac{(\omega^T x_i - b)^2}{\|x_i\|^2} \Rightarrow \nabla J_q(\omega) = \sum_{x_i \in X} \frac{\omega^T x_i - b}{\|x_i\|^2} x_i$$

X = Set of samples where $\omega^T x_i < b$

Summary

- Linear Decision boundaries are easily expressed as dot product with the co-efficient vector
- Gradient Descent allows us to learn the decision boundary from a set of labelled training samples
- Several variants of GD based on the loss function used
  - Perceptron learning algorithm is one such
  - Relaxation Methods
- GD Methods are generic and applicable to other problems as well
  - We will derive more of them later

Linear Regression

→ Find hyperplane that best approximates (y_i) in terms of (x_i)
→ Define loss function that's minimised when fit better
  ↳ Use GD
  ↳ Distance of samples from line
  ↳ Squared distance from line

Polynomial Regression

$$f(x; w) = \sum_{j=0}^{M} w_j x^j$$

$$J_{po}(\omega) = \frac{1}{2} \sum_i (f(x_i, w) - y_i)^2$$

$$\nabla J_{po}[j] = \sum_i (f(x_i, w) - y_i) x_i^j$$

Linear Regression with GD

$$J_{li}(\omega) = \frac{1}{2} \sum_i (\omega^T x_i - y_i)^2$$

$$\nabla J_{li} = \sum_i (\omega^T x_i - y_i) x_i$$

Use GD to minimise loss fn, update rule:

$$\omega^{t+1} = \omega^t + \eta \sum (y_i - \omega^T x_i) x_i$$

→ Closed form solutions exist
  ↳ Can do curve fitting as long as its linear in ω.

→ Basic Fn Models:

$$f(x_i, w) = \sum_{j=0}^{M} w_j \phi_j(x)$$

↳ Basis functions

Logistic Regression

→ $\sigma(a) = \frac{1}{1+e^{-a}}$ → Sigmoid fn

→ $\frac{\partial \sigma}{\partial a} = \sigma(1-\sigma)$

→ Uses logistic sigmoid of linear fn to approx. two class label: [0, 1]

$$J(w) = J_{LS}(w) = \frac{1}{2} \sum_i \left(\sigma(w^T x_i) - y_i\right)^2$$

$$\nabla_w J_{LS} = \sum_i \left(\sigma(w^T x_i) - y_i\right) \sigma($$

→ Not affected by outliers (Compared to GD)

Do this later

> **Summary**
>
> - Linear Regression is a generic tool to fit several types of curves to a set of data points
> - Multi-linear regression refers to approximating multiple output values for the same input
> - Logistic regression uses the regression formulation for 2-class classification
> - Will discuss non-linear regression problems later

MSE Procedures

→ Constrain $w^T x_i > 0$ to $w^T x_i = b$ for +ve b

→ n eq. with n training samples

$$X_{n \times \hat{d}} \, W_{\hat{d} \times 1} = b_{n \times 1} \Rightarrow Xw = b$$

↑ d+1

Multiply by X^{-1} both sides,

$$\boxed{w = X^{-1} b}$$ → Need not be invertible

Let $J_s(w)$ be diff b/w Xw & b:

$$J_s(w) = \|Xw - b\|^2 = \sum_{i=1}^{n} (w^T x_i - b_i)^2 \Rightarrow \nabla J_s = \sum_{i=1}^{n} 2(w^T x_i - b_i) x_i$$

↳ Can do GD to find optimal w [Widrow-Hoff]

To get closed form solution for MSE,

$$\nabla J_s = \sum_{i=1}^{n} 2x_i (x_i^T w - b_i)$$

In matrix,

$$\nabla J_s = 2X^T(Xw - b) \Rightarrow$$

At minimum $G = 0$,

$$X^T X w = X^T b$$

($X^T X$: $\hat{d} \times \hat{d}$)

$$\Rightarrow \boxed{w = (X^T X)^{-1} X^T b}$$

On comparison,

$$(X^T X)^{-1} X^T = X^{-1}$$

$$\boxed{X^+ = (X^T X)^{-1} X^T} \rightarrow \text{Pseudoinverse}$$

↳ Square & Non-singular $X^+ = X^{-1}$

↳ $X^+ = \lim_{\varepsilon \to 0} (X^T X + \varepsilon I)^{-1} X^T$

↳ $w = X^+ b$ is MSE solution to $Xw = b$

Ho-Kashyap Procedure

- If samples are linearly separable, there exists $\hat{a}$ and $\hat{b}$ such that: $Y\hat{a} = \hat{b}$, where $b_i > 0$ for all i.

$$J_s(a, b) = \|\mathbf{Xw} - \boldsymbol{b}\|^2$$

- As we do not know both $\hat{a}$ and $\hat{b}$, we will search for both.

$$\nabla_w J_s = 2\mathbf{X}^T(\mathbf{Xw} - \mathbf{b}), \text{ and}$$
$$\nabla_b J_s = -2(\mathbf{Xw} - \mathbf{b}).$$

- For any value of $\mathbf{b}$, we can set $\mathbf{w} = \mathbf{X}^+\mathbf{b}$, and if $e = (\mathbf{Xw} - \mathbf{b})$,

$$\mathbf{b}_{k+1} = \mathbf{b}_k + 2\eta e^+,$$

where e^+ is e with all its negative components set to 0.

Summary

- MSE procedures try to minimize the mean-squared error between the predicted value by the linear classifier and a set of constants, b.
- Depending on the choice of b, and the optimization procedure, we have different approaches.
- Each one has different numerical properties

NOTEs

→ Solution depends on b and need not on training samples

⇒ Computationally heavy

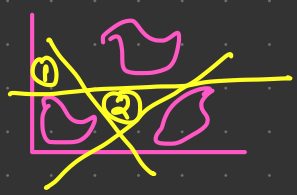
Linear to Multi Class (Lec 6)

Approach 1: One vs Rest

→ Compute 1 line per class (A vs(not A) etc)

→ Can lead to 2-way, 3-way ties

→ Use weighted voting as solution



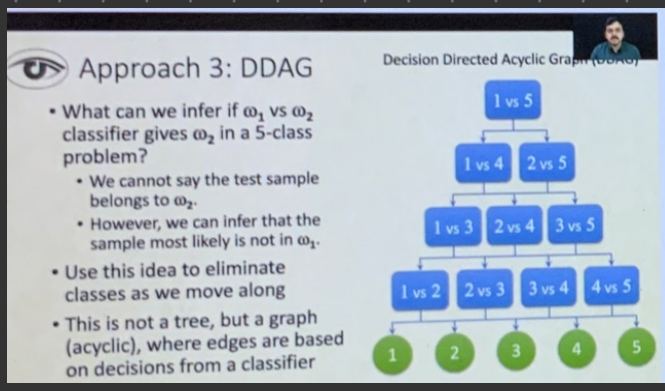
Approach 2: One vs One (Pairwise)

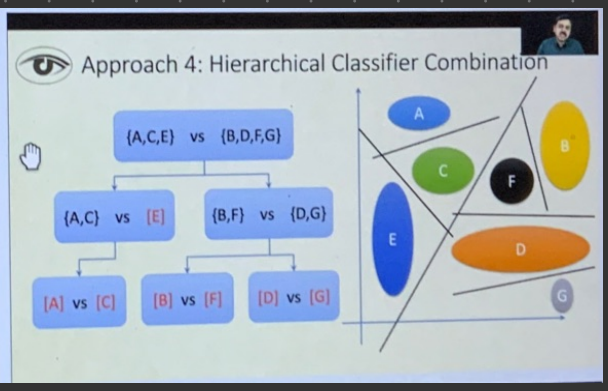
→ Compute 1 line per class pairs (A vs B etc)

⇒ Can lead to 3 way ties but lesser area relatively

Summary: Simple Approaches

- 1-vs-Rest:
  - Build n classifiers and vote (efficient)
  - Difficult to train as one-vs-rest may not be linearly separable
  - Several possibilities for indecision/tie
- 1-vs-1 (Pairwise):
  - Build ${}^nC_2$ classifiers and vote (compute heavy)
  - Easier to build as pairwise decision boundaries are simpler
  - Fewer possibilities for ties
- Weighted voting can break ties. Will explore ways to avoid it.





Linear Discriminant Fn's

- If $g_1(x), g_2(x)$ are linear fn's such that $g_1(x) \geq g_2(x)$, then x is more likely to belong to ω_1 else ω_2
- → Intersection of fn's becomes linear decision boundary $f(x) = g_1(x) - g_2(x)$

Summary: Efficient Approaches

- DDAG:
  - Build $n(n-1)/2$ classifiers and classify using $n-1$ of them
  - Compute heavy to build, but efficient in inference
  - Effective in classification (uses 1-v-1)
- Hierarchical
  - Build n classifiers by appropriate grouping of classes and pass through $\log(n)$
  - Far more efficient that 1-vs-1 (or DDAG)
  - Similar in accuracy to 1-v-1 and DDAG.
  - Difficult to find the right grouping of classes
- No Ties

→ For multiple classes, $\omega_i = \underset{i}{\text{argmax}}\, g_i(x)$

→ Partitions f-space using piecewise linear decision boundaries

→ Converts multiclass to two-class problems

Summary: Linear Machines

We have several methods to convert linear two-class classifiers into multi-class linear machines.

Some of the methods are simple and intuitive, others are more efficient and applicable in practice, and the linear discriminant functions provide mathematically elegant solutions for analysis and proofs.

The methods are applicable irrespective of the learning method used to learn **w** (also to support vector machines).

Note that the discriminant approach also generalizes to non-linear functions.

Lec 7

Dimensionality Reduction:

→ Reduce no. of features of feature vector
- → Less stuff to measure
- → Efficiency
- → Simpler boundaries
- → Ability to visualise

↳ How?
- → Select subset of features
- ↳ Define new features as linear function of original

Feature Selection:

→ Select subset (V) of original features (x)

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

Selected 2nd & 3rd feature

Sequential Search

Forward

→ Start with $V=\{\}$, $X=\{x_1 \dots x_d\}$
→ Select x_i which ↑ accuracy when added to V
→ Move x_i from X to V
→ Repeat 2 & 3 until satisfaction*

Backward

→ Start with $V=\{x_1, x_2 \dots x_d\}$, $X=\{\}$
→ Select x_i from V which ↑ accuracy or reduces drop in accuracy when removed from V
→ Move x_i from V to X
→ Repeat 2 & 3 until satisfaction*

* → Accuracy doesn't ↑ anymore
→ No. of features in V is desired

Feature Extraction:

→ Create new features as lin. combos of original
→ If highly correlated, can remove one (two case)

Feature space:

→ If confined close to a plane in 3D space
↳ Express them with 2 basis vectors (can get most info)
↳ Basis is orthogonal

↳ Find basis vectors, then shift plane to pass through origin, then shift points to be centered around origin