Linear Models III

CS771: Introduction to Machine Learning

Purushottam Kar

Announcements

Quiz 1 marks released on Gradescope

- *Please submit any and all regrading requests by 25th August, 2019*
- *Will close regrading requests for quiz 1 after that*
- *Regrading requests entertained only if rubrics were applied incorrectly or else if the rubrics missed a way of **completely correctly** solving the problem*
- *No more "partial" mark rubric will be added now*

Assignment 1 released

- *Two part submission: Gradescope (.pdf) + website (.py)*
- *Deadline: 01 September, 11:59PM IST*
- *Gradescope will stop accepting submissions after this deadline*
- *Google form will also close down after this deadline*
- *We will run a script to download code files after this deadline – make sure your file is present at the location you submitted in the Google form*

Recap of Last Lecture

Coordinate Methods for speedy optimization

Coordinate Ascent/Descent as well as Coordinate Maximization/Minimization

Application to CSVM dual problem – superior performance

Loss functions for binary classification problems

Hinge loss, Squared hinge loss, Logistic loss

Regression problems: solutions using kNN, DT

Loss functions for regression problems

Absolute Loss, Squared Loss, Vapnik's ϵ-insensitive loss, Huber loss

$f(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{n}\left(\mathbf{w}^\top\mathbf{x}^i - y^i\right)^2$ (also called *ridge regression*)

Regularization

An umbrella term used in ML to describe a whole family of steps taken to prevent ML algos from suffering from problems in data

These help ML algos offer stable behaviour even if data misbehaves

In an ideal world where data is perfectly clean and there is plenty of data available, there would be no need for any regularization!

How to do regularization is often decided without looking at data

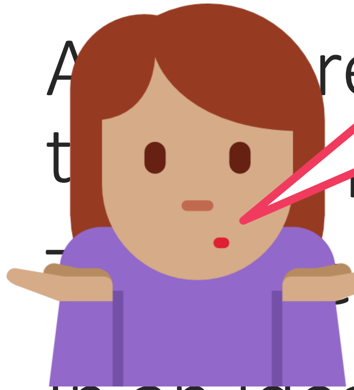
However, regularization usually involves its own hyperparameters that need to be tuned using data itself (using validation techniques)

In general, regularization techniques prevent the model from just blindly doing well on data (since data cannot be trusted)

Regularization

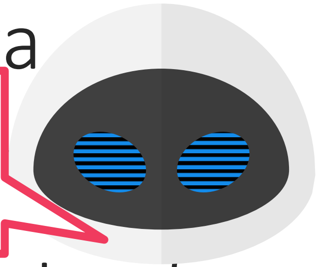
A re e a whole family of steps t from problems in data

T lp ML algos offer stable behavi



Makes sense since regularization is supposed to protect us from data issues

So regularization can be *somewhat* data dependent



In an ideal world where data is perfectly clean and there is plenty of data available, there would be no need for any regularization!

How to do regularization is often decided without looking at data

However, regularization usually involves its own hyperparameters that need to be tuned using data itself (using validation techniques)

In general, regularization techniques prevent the model from just blindly doing well on data (since data cannot be trusted)

Regularization by adding a regularizer

$f(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + C \cdot \sum_{i=1}^n \left[1 - y^i \cdot \mathbf{w}^\top \mathbf{x}^i\right]_+$

$f(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^n \left(\mathbf{w}^\top \mathbf{x}^i - y^i\right)^2$

The $\|\mathbf{w}\|_2^2$ term above is called the *L2 regularizer* (or the squared L2 regularizer if we want to be very specific)

In binary classification settings, we saw that this regularizer encourages a large margin

In regression settings it ensures uniqueness of solution

Recall that the closed form solution is $\mathbf{w} = (2X^\top X + \lambda \cdot I)^{-1}(2X^\top \mathbf{y})$

If $X^\top X$ *is non-invertible then we have infinitely many solutions if* $\lambda = 0$

Having $\lambda > 0$ *ensures unique solution no matter what the data*

R

A regularizer essentially tells the optimizer to not just blindly return a model that does well on data (according to the loss function), but rather return a model that does well and is *simple*. The L2 regularizer defines *simplicity* using the L2 norm (or Euclidean length). A model is simple if it has small L2 norm

f 2 $\|_2$ $\sum_{i=1}[$ $]_+$

$f(\mathbf{w}$

In binary classification, simple models also had large margins. However, be careful not to over regularize. If you use a very large value of regularization constant e.g. $\lambda \to \infty$ (or $C \to 0$ in SVM) then you may get a very useless model that does not fit data at all i.e. does not care to do well on data at all!

The e

regu

In binary classification settings, we saw that this regularizer

encou

In reg

The key is moderation. Usually regularization constants are chosen using validation. Other important considerations include: how noisy do we expect data to be and how much data do we have. **Rule of thumb**: as you have more and more data, you can safely afford to regularize less and less

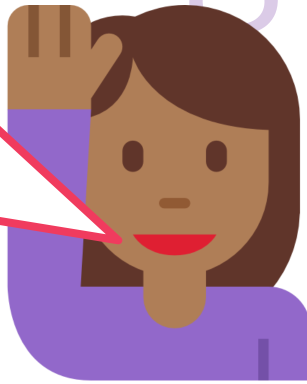
Rec)

If $X^\top X$ is non-invertible then we have infinitely many solutions if $\lambda =$ (

Having $\lambda > 0$ ensures unique solution no matter what the data

Othe

$\sum_{i=1}^{n} \boldsymbol{\alpha}_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{\alpha}_i\boldsymbol{\alpha}_j y^i y^j \left\langle \mathbf{x}^i, \mathbf{x}^j \right\rangle$ If you pay close attention, then the dual of CSVM also has L1 regularization on $\boldsymbol{\alpha}$. Since $\alpha_i \geq 0$, $\sum_{i=1}^{n} \boldsymbol{\alpha}_i = \|\boldsymbol{\alpha}\|_1$. Note that the dual does indeed have sparse solutions (i.e. not all $\boldsymbol{\alpha}_i \neq 0$) which means not every vector becomes a support vector!



The ot

LASSO: $f(\mathbf{w}) = \lambda\|\mathbf{w}\|_1 + \sum_{i=1}(\mathbf{w}^\top\mathbf{x}^i - y^i)$

L1-reg SVM: $f(\mathbf{w}) = \|\mathbf{w}\|_1 + C \cdot \sum_{i=1}^{n}\left[1 - y^i \cdot \mathbf{w}^\top\mathbf{x}^i\right]_+$

The L1 regularizer prefers model vectors that have lots of coordinates whose value is either 0 or close to 0 – called *sparse* vectors

Often, we make coordinates close to zero actually zero to save space

Sparse models are faster at test time, also consume less memory

Very popular in high dimensional problems e.g. $d \approx 1$ million

Since L1 norm is non-differentiable, need to use subgradient methods

Much better method: something called *proximal gradient descent*

Other popular regularizers

The other most popular regularizer is the L1 regularizer $\|\mathbf{w}\|_1$

LASSO: $f(\mathbf{w}) = \lambda\|\mathbf{w}\|_1 + \sum_{i=1}^n \left(\mathbf{w}^\top \mathbf{x}^i - y^i\right)^2$

L1-reg SVM: $f(\mathbf{w}) = \|\mathbf{w}\|_1 + C \cdot \sum_{i=1}^n \left[1 - y^i \cdot \mathbf{w}^\top \mathbf{x}^i\right]_+$

The L1 regularizer prefers model vectors that have lots of coordinates whose value is either 0 or close to 0 – called *sparse* vectors

Often, we make coordinates close to zero actually zero to save space

Sparse models are faster at test time, also consume less memory

Very popular in high dimensional problems e.g. $d \approx 1$ million

Since L1 norm is non-differentiable, need to use subgradient methods

Much better method: something called *proximal gradient descent*

Regularization by Early Stopping

Sometimes, ML practitioners stop an optimizer well before it has solved the optimization problem fully – sometimes due to timeout but sometimes deliberately

Note that this automatically prevents the model from fitting the data too closely (this is good if the data was noisy or had outliers)

This often happens implicitly with complex optimization problems e.g. training deep networks where the person training the network gets tired and gives up training – gets some regularization for free ☺

Be careful not to misuse this – if you stop too early, you may just get an overregularized model that does not fit data at all i.e. does not do well on data at all!

Regularization by adding noise

A slightly counter-intuitive way of regularization (considering that regularization is supposed to save us from noise in data)

Add controlled noise to data so that the model learns to perform well despite noise – note that it does not fit the data exactly here either

Most well-known instance of this technique is the practice of dropout in deep learning – randomly make features go missing

Related methods: learn not from entire data, but a subset of the data

Of the d features present, choose only those that are informative, non-noisy

*Called **sparse recovery**: e.g. LASSO does sparse recovery for least squares*

Of the n data points, choose only those data points which look clean

*Called **robust learning**: e.g. Huber loss used for robust regression*

Multiclass Classification

Sometimes aka multiclassification – have seen kNN and DTs solve it

Can be solved using linear models too!

Trick is to reduce to binary classification

If there are C classes, then train C linear models, each trained to identify one class

E.g. $C = 3$ {dog, horse, fish}. Train model1 to say yes to dog images but no to horse and no to fish images, similarly model2, 3

Called the OVA method (one-vs-all)

At test time, just ask all three models and hope that only one of them says yes ☺

Multiclass Classification

Sometimes aka multiclassification – have seen kNN and DTs solve it

Can be solved using linear models too!

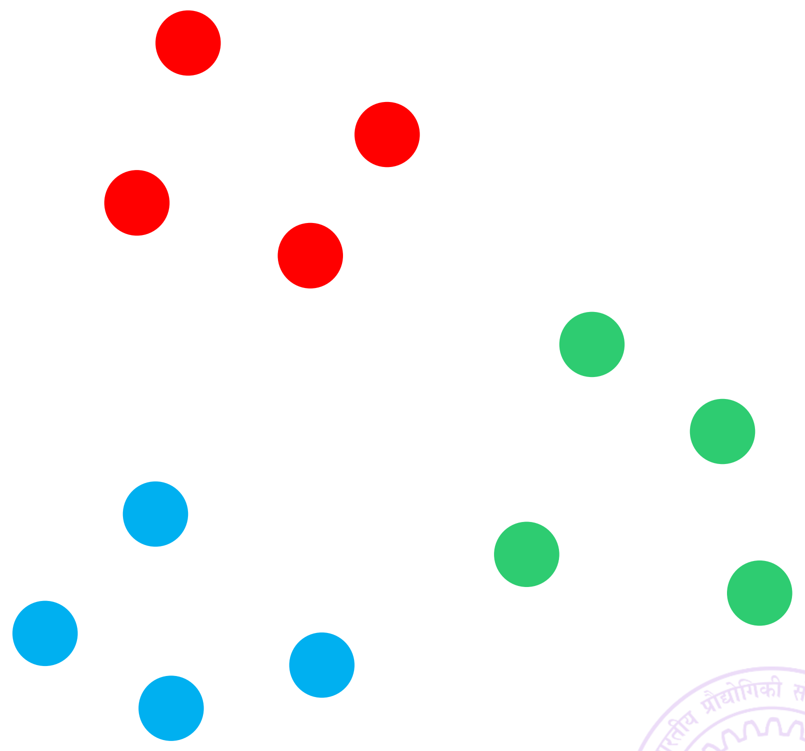
Trick is to reduce to binary classification

If there are C classes, then train C linear models, each trained to identify one class

E.g. $C = 3$ {dog, horse, fish}. Train model1 to say yes to dog images but no to horse and no to fish images, similarly model2, 3

Called the OVA method (one-vs-all)

At test time, just ask all three models and hope that only one of them says yes ☺



Multiclass Classification

Sometimes aka multiclassification – have seen kNN and DTs solve it

Can be solved using linear models too!

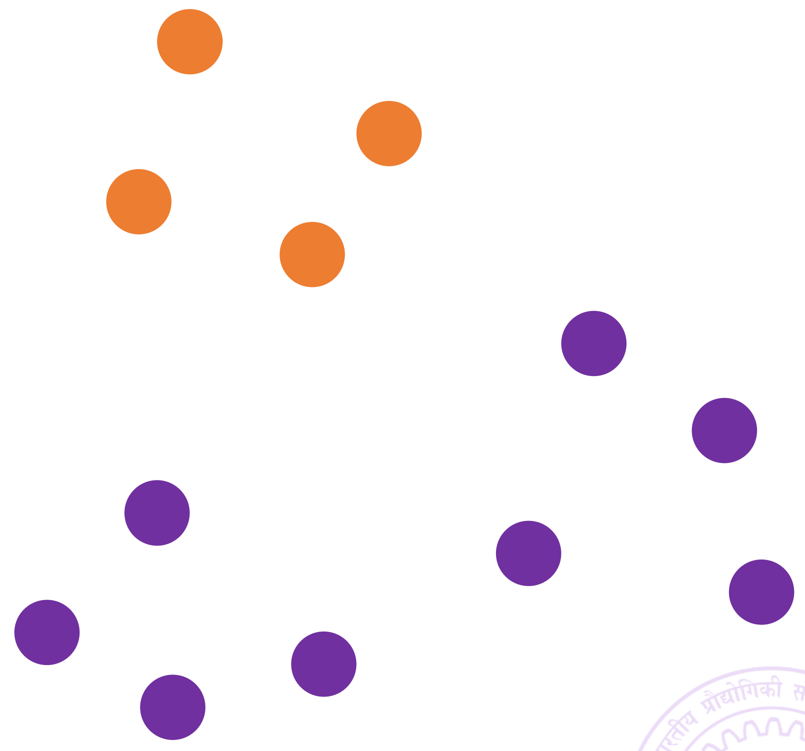
Trick is to reduce to binary classification

If there are C classes, then train C linear models, each trained to identify one class

E.g. $C = 3$ {dog, horse, fish}. Train model1 to say yes to dog images but no to horse and no to fish images, similarly model2, 3

Called the OVA method (one-vs-all)

At test time, just ask all three models and hope that only one of them says yes ☺



Multiclass Classification

Sometimes aka multiclassification – have seen kNN and DTs solve it

Can be solved using linear models too!

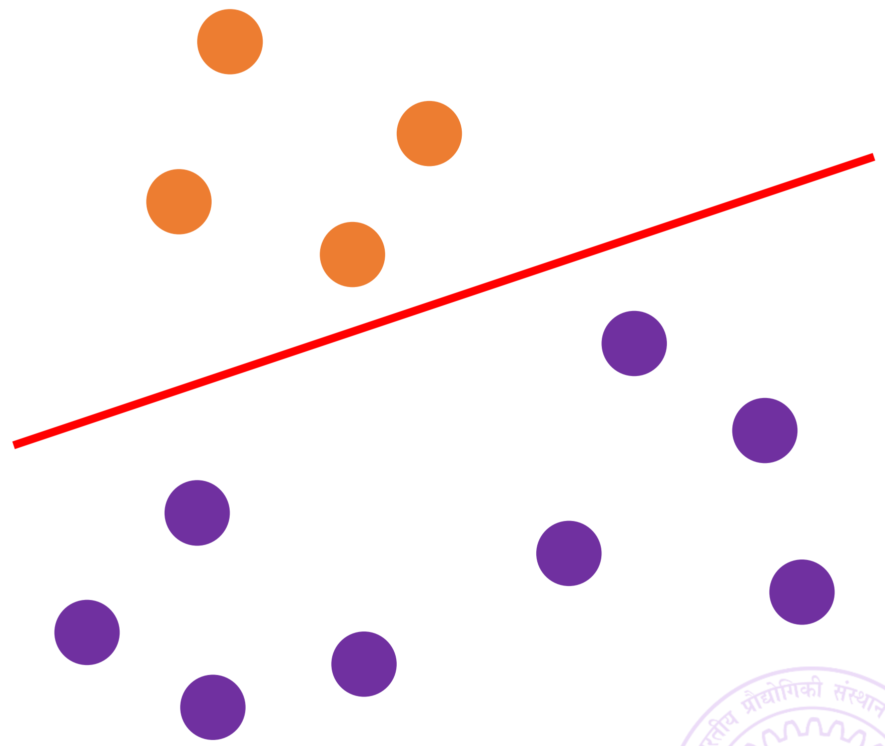
Trick is to reduce to binary classification

If there are C classes, then train C linear models, each trained to identify one class

E.g. $C = 3$ {dog, horse, fish}. Train model1 to say yes to dog images but no to horse and no to fish images, similarly model2, 3

Called the OVA method (one-vs-all)

At test time, just ask all three models and hope that only one of them says yes ☺



Multiclass Classification

Sometimes aka multiclassification – have seen kNN and DTs solve it

Can be solved using linear models too!

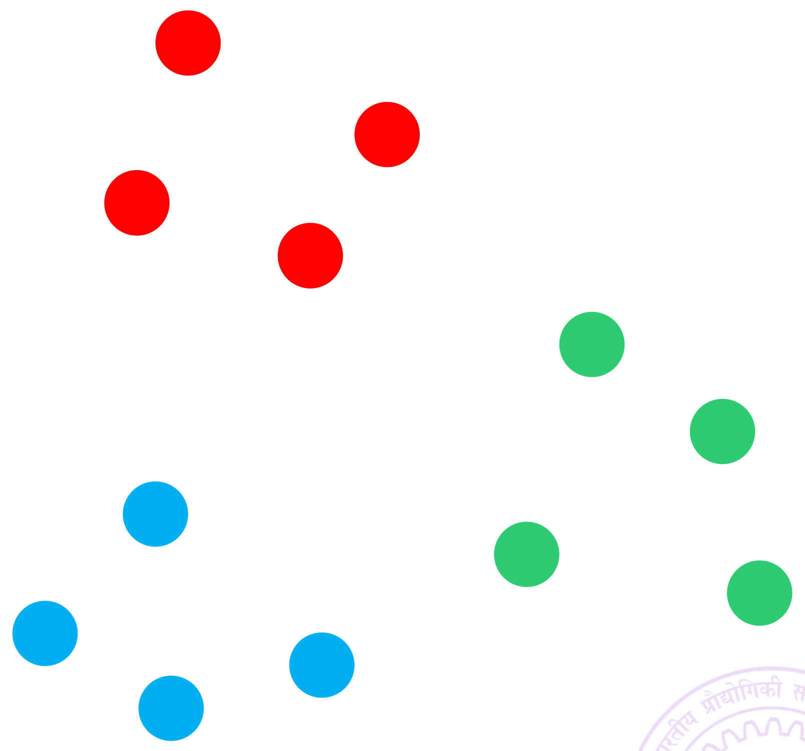
Trick is to reduce to binary classification

If there are C classes, then train C linear models, each trained to identify one class

E.g. $C = 3$ {dog, horse, fish}. Train model1 to say yes to dog images but no to horse and no to fish images, similarly model2, 3

Called the OVA method (one-vs-all)

At test time, just ask all three models and hope that only one of them says yes ☺



Multiclass Classification

Sometimes aka multiclassification – have seen kNN and DTs solve it

Can be solved using linear models too!

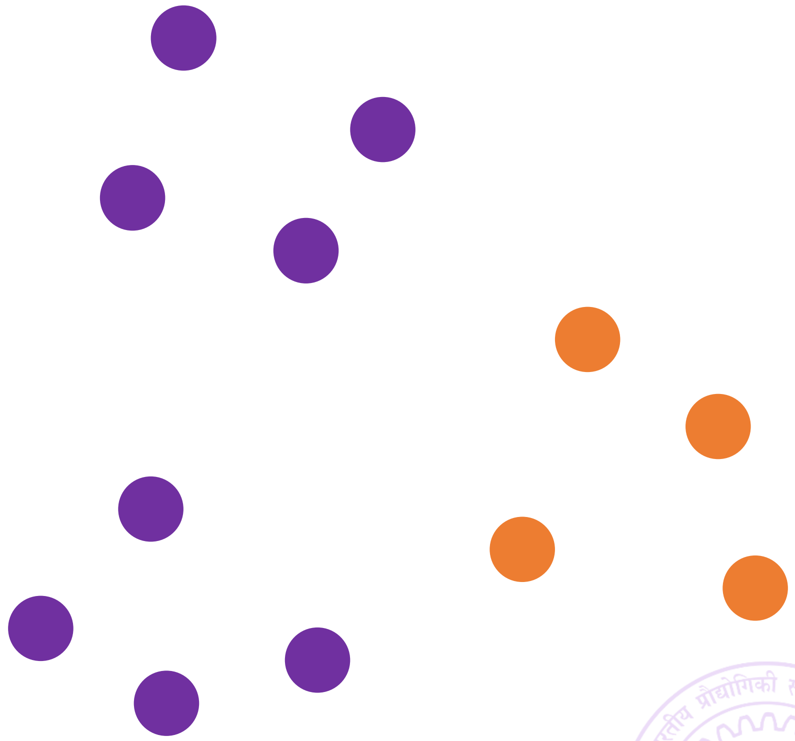
Trick is to reduce to binary classification

If there are C classes, then train C linear models, each trained to identify one class

E.g. $C = 3$ {dog, horse, fish}. Train model1 to say yes to dog images but no to horse and no to fish images, similarly model2, 3

Called the OVA method (one-vs-all)

At test time, just ask all three models and hope that only one of them says yes ☺



Multiclass Classification

Sometimes aka multiclassification – have seen kNN and DTs solve it

Can be solved using linear models too!

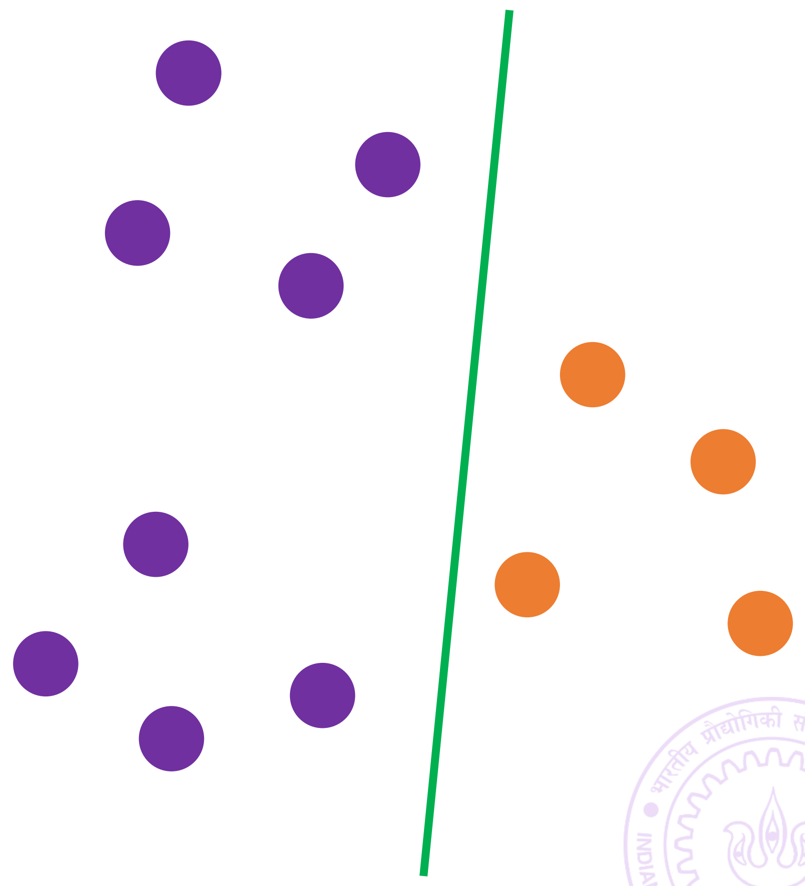
Trick is to reduce to binary classification

If there are C classes, then train C linear models, each trained to identify one class

E.g. $C = 3$ {dog, horse, fish}. Train model1 to say yes to dog images but no to horse and no to fish images, similarly model2, 3

Called the OVA method (one-vs-all)

At test time, just ask all three models and hope that only one of them says yes ☺



Multiclass Classification

Sometimes aka multiclassification – have seen kNN and DTs solve it

Can be solved using linear models too!

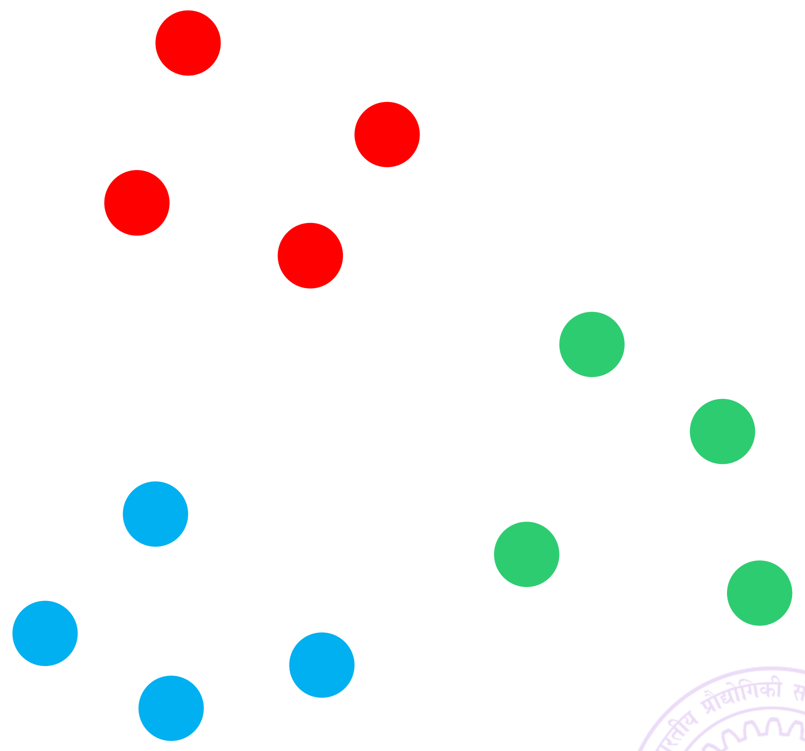
Trick is to reduce to binary classification

If there are C classes, then train C linear models, each trained to identify one class

E.g. $C = 3$ {dog, horse, fish}. Train model1 to say yes to dog images but no to horse and no to fish images, similarly model2, 3

Called the OVA method (one-vs-all)

At test time, just ask all three models and hope that only one of them says yes ☺



Multiclass Classification

Sometimes aka multiclassification – have seen kNN and DTs solve it

Can be solved using linear models too!

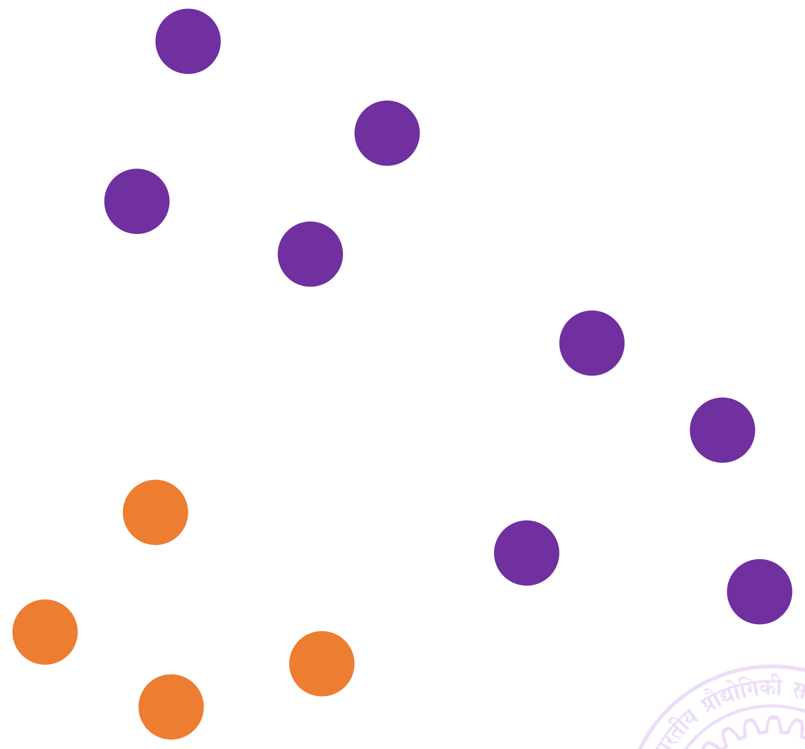
Trick is to reduce to binary classification

If there are C classes, then train C linear models, each trained to identify one class

E.g. $C = 3$ {dog, horse, fish}. Train model1 to say yes to dog images but no to horse and no to fish images, similarly model2, 3

Called the OVA method (one-vs-all)

At test time, just ask all three models and hope that only one of them says yes ☺



Multiclass Classification

Sometimes aka multiclassification – have seen kNN and DTs solve it

Can be solved using linear models too!

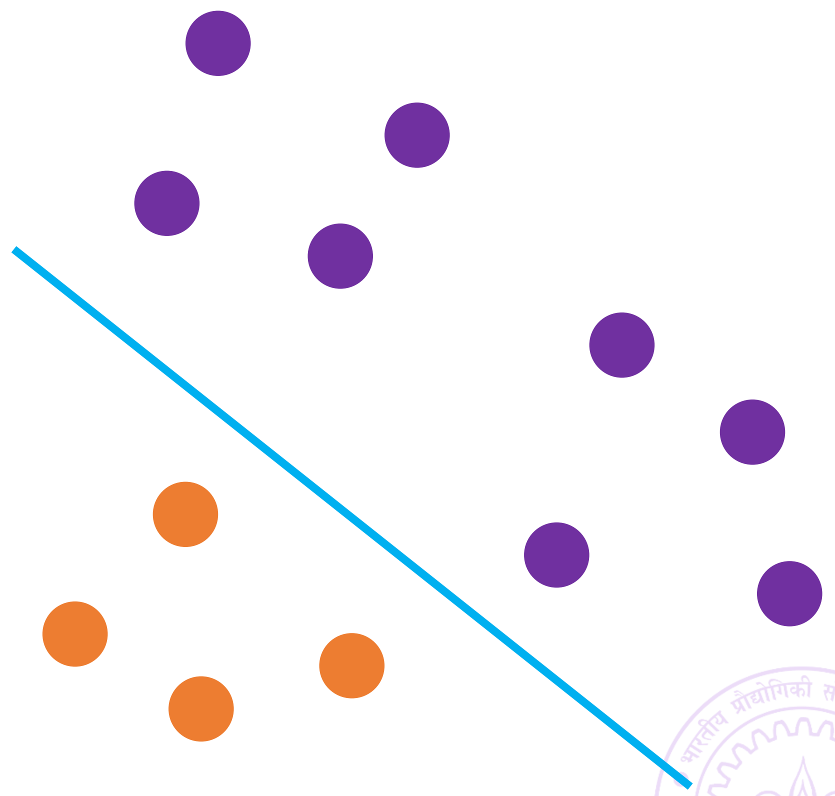
Trick is to reduce to binary classification

If there are C classes, then train C linear models, each trained to identify one class

E.g. $C = 3$ {dog, horse, fish}. Train model1 to say yes to dog images but no to horse and no to fish images, similarly model2, 3

Called the OVA method (one-vs-all)

At test time, just ask all three models and hope that only one of them says yes ☺



Multiclass Classification

Sometimes aka multiclassification – have seen kNN and DTs solve it

Can be solved using linear models too!

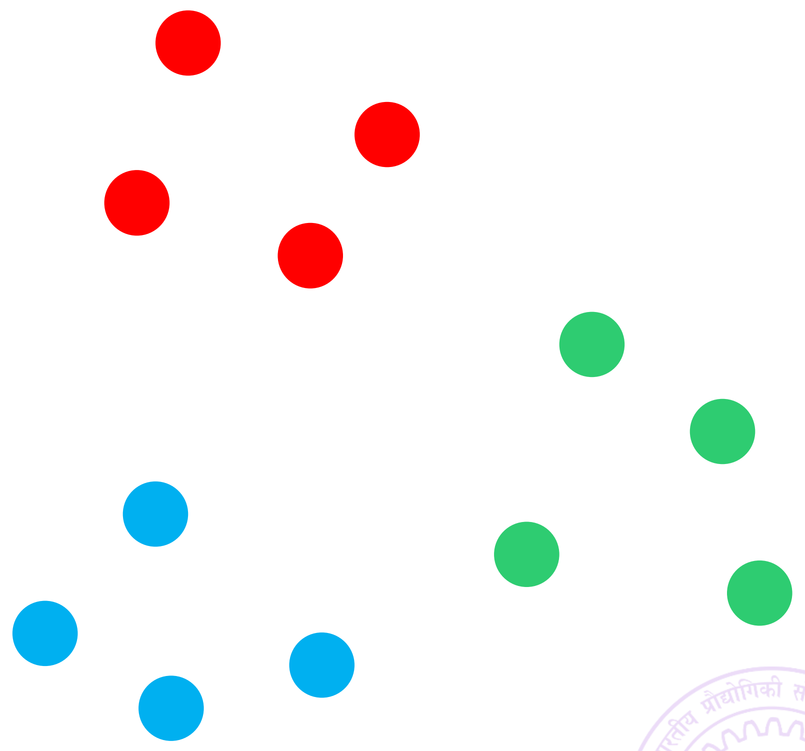
Trick is to reduce to binary classification

If there are C classes, then train C linear models, each trained to identify one class

E.g. $C = 3$ {dog, horse, fish}. Train model1 to say yes to dog images but no to horse and no to fish images, similarly model2, 3

Called the OVA method (one-vs-all)

At test time, just ask all three models and hope that only one of them says yes ☺



Multiclass Classification

Sometimes aka multiclassification – have seen kNN and DTs solve it

Can be solved using linear models too!

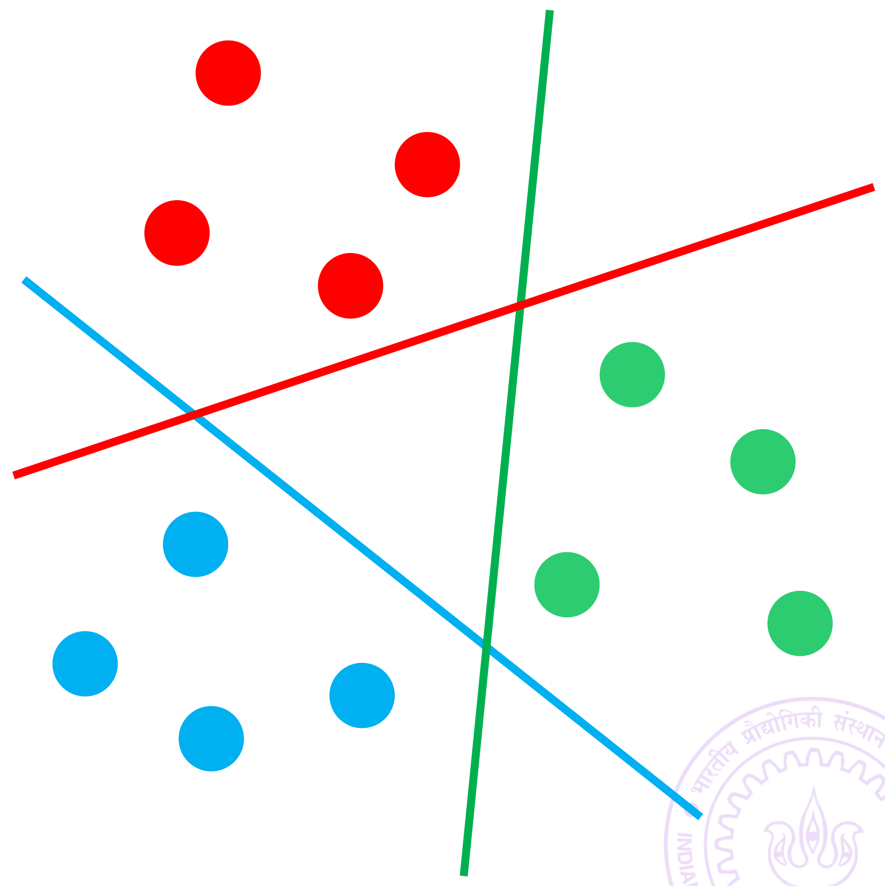
Trick is to reduce to binary classification

If there are C classes, then train C linear models, each trained to identify one class

E.g. $C = 3$ {dog, horse, fish}. Train model1 to say yes to dog images but no to horse and no to fish images, similarly model2, 3

Called the OVA method (one-vs-all)

At test time, just ask all three models and hope that only one of them says yes ☺



Multiclass Classification

Sometimes aka multiclassification – have seen kNN and DTs solve it

Can be solved using linear models too!

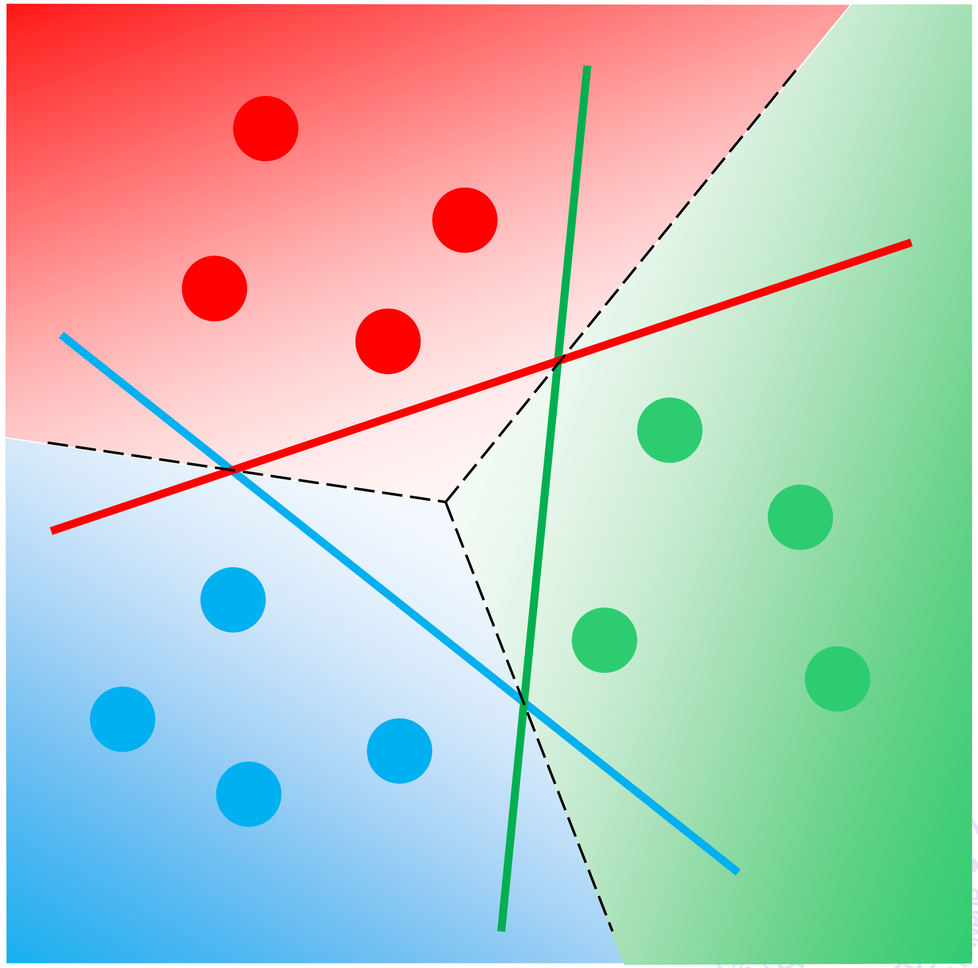
Trick is to reduce to binary classification

If there are C classes, then train C linear models, each trained to identify one class

E.g. $C = 3$ {dog, horse, fish}. Train model1 to say yes to dog images but no to horse and no to fish images, similarly model2, 3

Called the OVA method (one-vs-all)

At test time, just ask all three models and hope that only one of them says yes ☺



OVA – Reduce to C binary problems

Create C binary classification datasets

For each $c \in [C]$, create a dataset where points in class c are labelled positive and points of all other classes labelled negative

$$y^{i,(c)} = \begin{cases} 1 & ; y^i = c \\ -1 & ; y^i \neq c \end{cases}$$

Learn a model to distinguish data points in class c from those not in class c

$$\hat{\mathbf{w}}^c = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \ell(y^{i,(c)}, \langle \mathbf{w}, \mathbf{x}^i \rangle)$$

At test time, predict the class whose model gives the test point the highest score!

$$\hat{y}^t = \arg\max_{c \in [C]} \langle \hat{\mathbf{w}}^c, \mathbf{x}^t \rangle$$

OVA – Learn the C models together

Can introduce the concept of margin here as well

Demand that if the true class of a data point $\mathbf{x}$ is c^*, then we must have $\langle \mathbf{w}^{c^*}, \mathbf{x} \rangle \geq \langle \mathbf{w}^{c}, \mathbf{x} \rangle + 1$ for all $c \neq c^*$

Introducing slack as before allows us to form an optimization problem

$$\min_{\{\mathbf{w}^c\},\{\xi_i\}} \frac{1}{2} \cdot \sum_{c=1}^{C} \|\mathbf{w}^c\|_2^2 + K \sum_{i=1}^{n} \xi_i$$

s.t. $\left\langle \mathbf{w}^{y^i}, \mathbf{x}^i \right\rangle \geq \langle \mathbf{w}^k, \mathbf{x}^i \rangle + 1 - \xi_i$ for all $k \neq y^i$ and $\xi_i \geq 0$ for all i

Can rewrite this in terms of the *Crammer-Singer Loss* (let $\eta_c = \langle \mathbf{w}^c, \mathbf{x} \rangle$)

$$\min_{\{\mathbf{w}^c\}} \frac{1}{2} \cdot \sum_{c=1}^{C} \|\mathbf{w}^c\|_2^2 + K \sum_{i=1}^{n} \ell_{\text{cs}}\left(y^i, \{\langle \mathbf{w}^c, \mathbf{x}^i \rangle\}_{c=1}^{C}\right)$$

$$\ell_{\text{cs}}\left(y, \{\eta_c\}_{c=1}^{C}\right) = \left[1 + \max_{c \neq y} \eta_c - \eta_y\right]$$

OVA via Softmax

Just as hinge loss becomes Crammer-Singer loss when looking at multiclassification, logistic loss becomes the softmax loss

$$\ell_{\text{SM}}\left(y, \{\eta_c\}_{c=1}^C\right) = -\ln\left(\frac{\exp(\eta_y)}{\sum_{c=1}^C \exp(\eta_c)}\right)$$

where we have $\eta_c = \langle \mathbf{w}^c, \mathbf{x} \rangle$

Note that this loss also encourages η_y to be the largest of all η_c

The loss approaches zero only if η_y is enormously larger than all η_c

This ensures $\sum_{c=1}^C \exp(\eta_c) \approx \exp(\eta_y)$ *and use* $\ln(x) \to 0$ *as* $x \to 1$

Popular since this is differentiable and so gradients can be taken

Multiclassification – popular techniques

Decision Trees: very popular especially if number of classes $C \gg 1$

OVA: can be slow if $C \gg 1$ but ways exist to speed things up

Crammer Singer present in liblinear, sklearn. Softmax popular in deep learning

Output Codes: convert multiclassification into a bunch of regression problems.

Represent each class $c \in [C]$ using a k-dim vector $\mathbf{v}_c \in \mathbb{R}^k$.

Solve k regression problems on the data, essentially trying to predict $\mathbf{v}_1$ for data points that belong to class 1, $\mathbf{v}_2$ for data points that belong to class 2 etc

At test time, predict k numbers for the test point, think of this as a k-dim vector and see if this vector is closest to $\mathbf{v}_1$ or $\mathbf{v}_2$ or etc …

Want k to be small for sake of speed but cannot have very small k. The whole purpose of having $k > 1$ is to account for regression mistakes

How to grade an exam

How to grade an exam

gradescope

How to grade an exam

gradescope

Q1. $\int x = ?$

5 marks

How to grade an exam

gradescope

Q1. $\int x = ?$

5 marks

$\frac{x^2}{2} + b$

$x^2 + d$

$x^2/2$

$x^2/2$

$x^2 + C$

$\frac{x^2}{2} + C$

$x^2/2$

$x^2/2 + a$

$x^2 + \varepsilon$

How to grade an exam

gradescope

Q1. $\int x = ?$

5 marks

$x^2/2$

$x^2/2$ $X^2/2$

$\frac{X^2}{2} + a$

$\frac{x^2}{2} + b$ $\frac{x^2}{2} + c$

$x^2 + c$ $X^2 + d$

$X^2 + \varepsilon$

How to grade an exam

gradescope

Q1. $\int x = ?$

5 marks

$x^2/2$ $x^2/2$ $X^2/2$

$\frac{X^2}{2} + a$ $\frac{x^2}{2} + b$ $\frac{x^2}{2} + c$

$x^2 + c$ $X^2 + d$ $X^2 + e$

How to grade an exam

gradescope

Q1. $\int x = ?$

5 marks

$\frac{x^2}{2}$ $\frac{x^2}{2}$ $\frac{x^2}{2}$

4/5

$\frac{x^2}{2} + a$

$\frac{x^2}{2} + b$ $\frac{x^2}{2} + c$

5/5

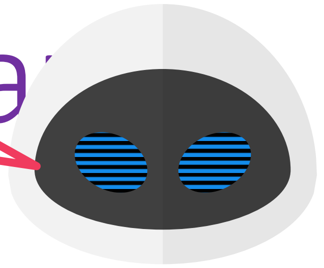
$x^2 + c$ $x^2 + d$

$x^2 + e$

2/5

Ho… …xa…

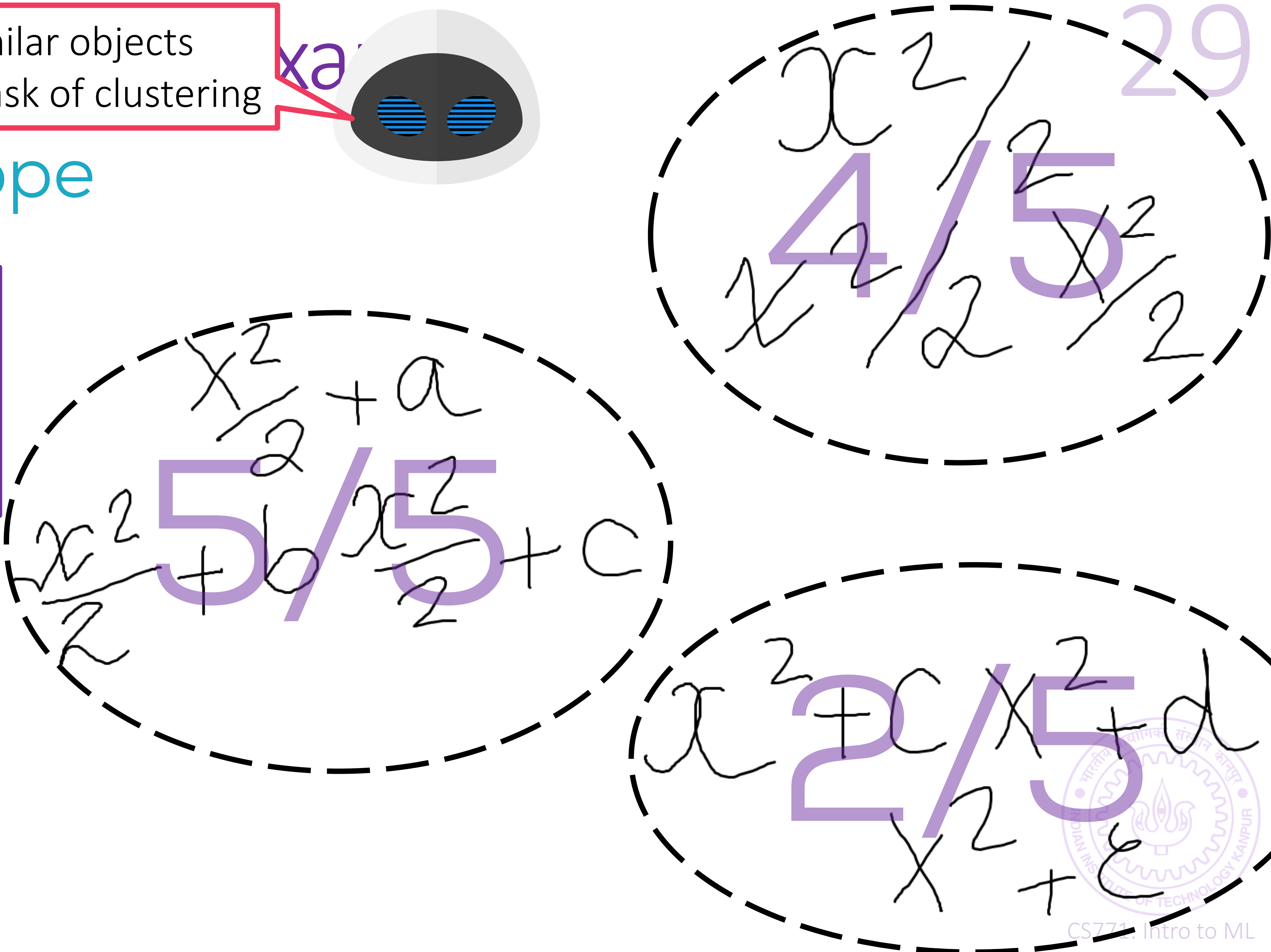
Grouping similar objects together is the task of clustering



gradescope

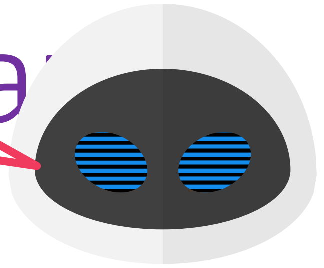
Q1. $\int x = ?$

5 marks



Ho xa

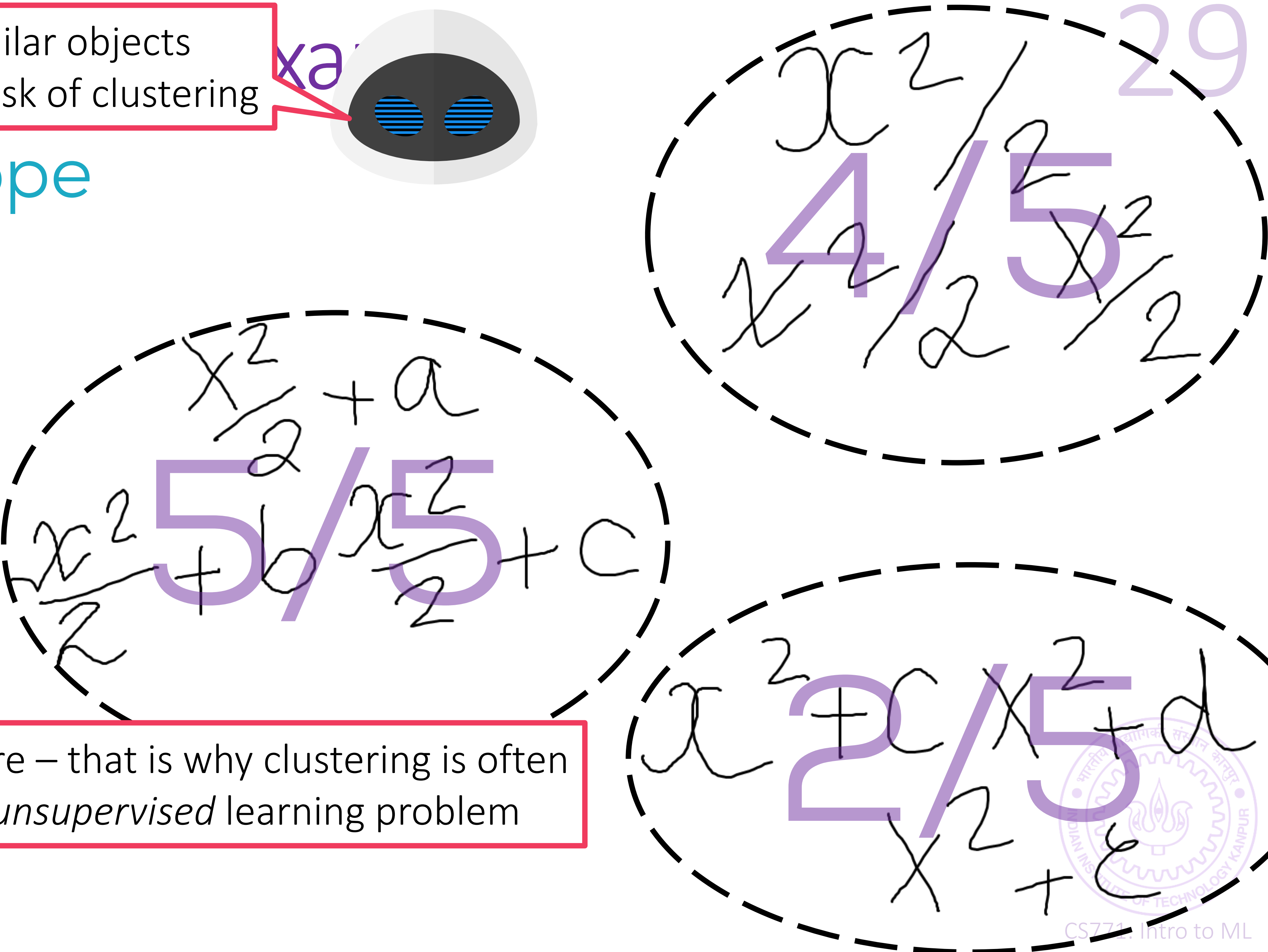
Grouping similar objects together is the task of clustering

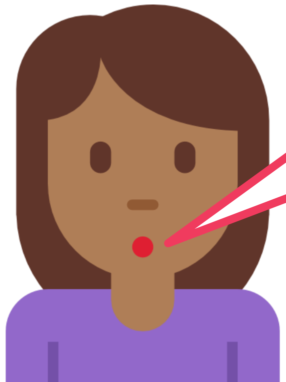


gradescope

Q1. $\int x = ?$

5 marks





No labels here – that is why clustering is often called an *unsupervised* learning problem

Clustering

Given a set S of n data points $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^n \in \mathbb{R}^d$

Split this set into C disjoint *clusters* $S_1, \ldots S_C$ i.e.

Assign every data point $\boldsymbol{i}$ *to one of the subsets, say* $z_i \in [C]$ *(note that every data point is assigned to exactly one cluster) so that*

Data points assigned to the same subset are "similar" to each other, e.g.

If $z_i = z_j = c$ *for some* $c \in [C]$ *then* $\left\|\mathbf{x}^i - \mathbf{x}^j\right\|_2$ *is small*

The K-means problem asks this problem a bit differently

Split S into C *clusters* $S_1, \ldots S_C$ *and find a prototype for each cluster i.e.* $\boldsymbol{\mu}^c \in \mathbb{R}^d$ *s.t. if* $\mathbf{x}^i$ *is assigned to cluster* $\boldsymbol{c}$ *i.e.* $z_i = c$, *then* $\left\|\mathbf{x}^i - \boldsymbol{\mu}^c\right\|_2^2$ is small i.e. $\mathbf{x}^i$ *is close to prototype of its cluster*

K-means clustering

$$\min_{\{\boldsymbol{\mu}^c \in \mathbb{R}^d\},\{z_i \in [C]\}} \sum_{c=1}^{C} \sum_{i:z_i=c} \left\| \mathbf{x}^i - \boldsymbol{\mu}^c \right\|_2^2$$

This optimization problem is NP hard to solve ☹

Popular heuristic: *Lloyd's algorithm* (often called *k-means algorithm*)

Uses a technique called alternating minimization

Observation 1*: if we fix all $\boldsymbol{\mu}^c$, obtaining optimal assignments z_i is very simple*

Assign each data point to the cluster whose prototype is closest!

Observation 2*: if we fix all assignments z_i, obtaining optimal prototypes simple*

$\min_{\{\boldsymbol{\mu}^c \in \mathbb{R}^d\}} \sum_{c=1}^{C} \sum_{i:z_i=c} \left\| \mathbf{x}^i - \boldsymbol{\mu}^c \right\|_2^2$ -- *all that is needed is* $\min_{\boldsymbol{\mu}^c} \sum_{i:z_i=c} \left\| \mathbf{x}^i - \boldsymbol{\mu}^c \right\|_2^2$

Apply first order optimality to deduce that optimal value of $\boldsymbol{\mu}^c$ is simply the average of all data points assigned to the cluster c – repeat for all $c \in [C]$

Keep repeating these two steps again and again

Looks a bit like coordinate minimization where we fix all but one coordinate and update that one coordinate to its optimal value

$\min_{\{\boldsymbol{\mu}^c \in \mathbb{R}^d\}, \{z_i \in [C$

This optimizati

Popular heuristi *ns algorithm)*

Uses a techniqu

Observation 1: *s z_i is very simple*

Assign each da *est!*

Observation 2: *prototypes simple*

$\min_{\{\boldsymbol{\mu}^c \in \mathbb{R}^d\}} \sum_{c=1}^{C} \sum_{i}$ $_{i:z_i=c} \left\|\mathbf{x}^i - \boldsymbol{\mu}^c\right\|_2^2$

Apply first orde *$\boldsymbol{\mu}^c$ is simply the average of all data points assigned to the cluster c – repeat for all c*

Keep repeating these t

K-MEANS/LLOYD'S ALGORITHM

1. Initialize means $\{\boldsymbol{\mu}^c\}_{c=1\ldots C}$
2. For $i \in [n]$, update z_i using $\{\boldsymbol{\mu}^c\}$
   1. Let $z_i = \arg\min_c \left\|\mathbf{x}^i - \boldsymbol{\mu}^c\right\|_2^2$
3. Let $n_c = \#$ points assigned to c
4. Update $\boldsymbol{\mu}^c = \frac{1}{n_c} \sum_{i:z_i=c} \mathbf{x}^i$
5. Repeat until convergence

True, coordinate minimization can be thought of as a special case of alternating optimization ☺