Loss and Regularization

CS771: Introduction to Machine Learning

Purushottam Kar

Topics to be Covered

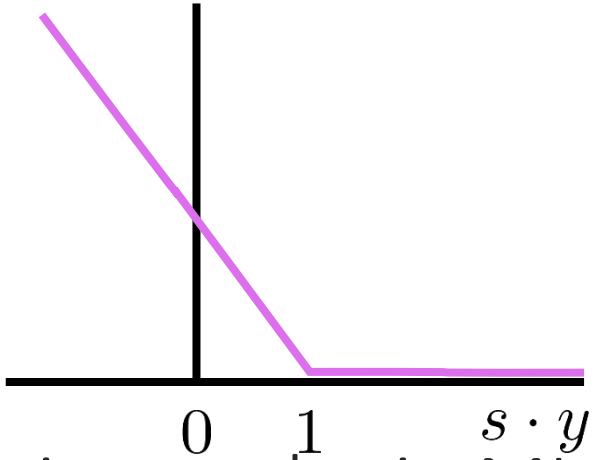
- Concept of loss function and regularization in ML
- Loss functions for binary and multi classification problems
- Loss functions for regression problems

Loss Functions

Loss functions are used in ML to ask the model to *behave* a certain way

Loss functions penalize undesirable beha[illegible] or SDCA then (hopefully) give us a model [illegible]

$$\ell_{\text{hinge}}\left(y^i \cdot \mathbf{w}^\top \mathbf{x}^i\right) = \left[1 - y^i \cdot \mathbf{w}^\top \mathbf{x}^i\right]_+$$



E.g. the hinge loss function which penalizes a model if it either misclassifies a data point or else classifies it correctly but with insufficient margin

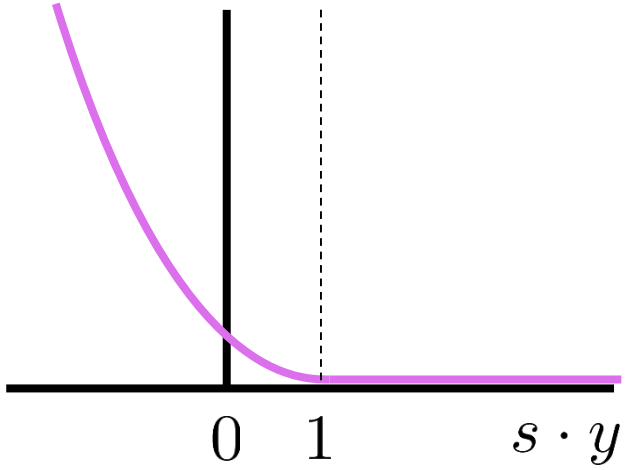
Note: hinge loss tells us how well is model doing on a single data point $\left(\mathbf{x}^i, y^i\right)$. Given such a loss function, it is popular in ML to then take the sum or average of these datapoint-wise loss values on the entire training set to see how well is model doing on the entire training set. For example, CSVM relies on $\sum_{i=1}^{n} \ell_{\text{hinge}}\left(y^i \cdot \mathbf{w}^\top \mathbf{x}^i\right)$

Other Classification Loss Functions

Squared Hinge loss Function:

$$\ell_{\text{sq-hinge}}\left(y^i \cdot \mathbf{w}^\top \mathbf{x}^i\right) = \left[1 - y^i \cdot \mathbf{w}^\top \mathbf{x}^i\right]_+^2$$

Popular since it is differentiable – no kinks

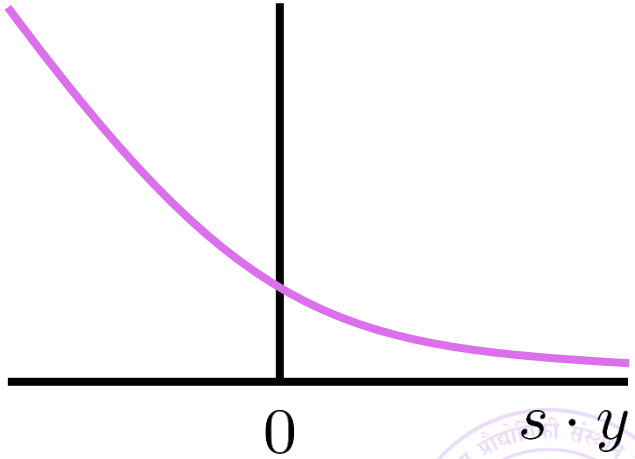


Logistic Loss Function:

$$\ell_{\text{logistic}}\left(y^i \cdot \mathbf{w}^\top \mathbf{x}^i\right) = \ln\left(1 + \exp\left(-y^i \cdot \mathbf{w}^\top \mathbf{x}^i\right)\right)$$

Popular, differentiable

Related to the cross-entropy loss function



Some loss functions e.g. hinge, can be derived in a geometric way, others e.g. logistic, can be derived probabilistically

However, some e.g. squared hinge, are directly proposed by ML experts as they have nice properties – no separate "intuition" for these ☺

Logistic Regression

If we obtain a model by solving the following problem

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2 + C \cdot \sum_{i=1}^{n} \ln\left(1 + \exp\left(-y^i \cdot \mathbf{w}^\top \mathbf{x}^i\right)\right)$$

then we would be doing *logistic regression*

Note that we simply replaced hinge loss with logistic loss here

Warning: logistic regression solves a binary classification problem, not a regression problem. The name is misleading

Can be solved in primal by using GD/SGD/MB

Can be solved in dual using SDCA as well

Dual derivation not that straightforward here since no constraints ☺

Will revisit logistic regression and derive the loss function very soon

Regression Problems

In binary classification, we have to predict one of two classes for each data point i.e. $\mathbf{x} \mapsto \{-1,1\}$

In regression, we have to predict a real value i.e. $\mathbf{x} \mapsto \mathbb{R}$

Training data looks like $\left\{\left(\mathbf{x}^i, y^i\right)\right\}_{i=1}^{n}$ where $\mathbf{x}^i \in \mathbb{R}^d, y \in \mathbb{R}$

Predicting price of a stock, predicting *change* in price of a stock, predicting test scores of a student, etc can be solved using regression

Let us look at a few ways to solve regression problem as well as loss functions for regression problems

Recall: logistic regression is not a way to perform regression, it is a way to perform binary classification

Solving Regression Problems via kNN

Store all training data as the "model"

For a given test data point $\mathbf{x}^t$, find its k nearest neighbours

May use Euclidean/learnt metric to define neighbours

Suppose neighbours are $\left(\mathbf{x}^{i_1}, y^{i_1}\right), \ldots, \left(\mathbf{x}^{i_k}, y^{i_k}\right)$

Predict the average score of neighbours i.e. $\hat{y}^t = \frac{1}{k}\sum_{j=1}^{k} y^{i_j}$

Solving Regression Problems via DT

We will study concepts of variance and absolute variance soon when we look at our probability refresher. Also, notice that purity definition 2 and 4 are the same definition but scaled differently. Can you show this yourself?

Nee

Hopeless to expect y values to repeat at a node and use old notion of

May call a node pure if y values in that node are "close" to each other

May define this in many ways: given $(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^n, y^n)$ at a node

Purity definition 1: $\sum_{i=1}^{n} \sum_{j=1}^{n} \left| y^i - y^j \right|$

Purity definition 2: $\sum_{i=1}^{n} \sum_{j=1}^{n} \left(y^i - y^j \right)^2$

Purity definition 3: *Let* $\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y^i$ *and use* $\sum_{i=1}^{n} \left| y^i - \bar{y} \right|$ *as purity*

Purity definition 4: *Let* $\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y^i$ *and use* $\sum_{i=1}^{n} \left(y^i - \bar{y} \right)^2$ *as purity*

Defn 4 is related to *variance*. Defn 3 is related to *absolute deviation*

Possible leaf action: predict average y for training points at that leaf

Loss Functions for Regression Problems

Can use linear models to solve regression problems too i.e. learn a $(\mathbf{w}, b)$ and predict score for test data point $\mathbf{x}^t$ as $\hat{y}^t = \mathbf{w}^\top \mathbf{x}^t + b$

Need loss functions that define what they think is bad behaviour

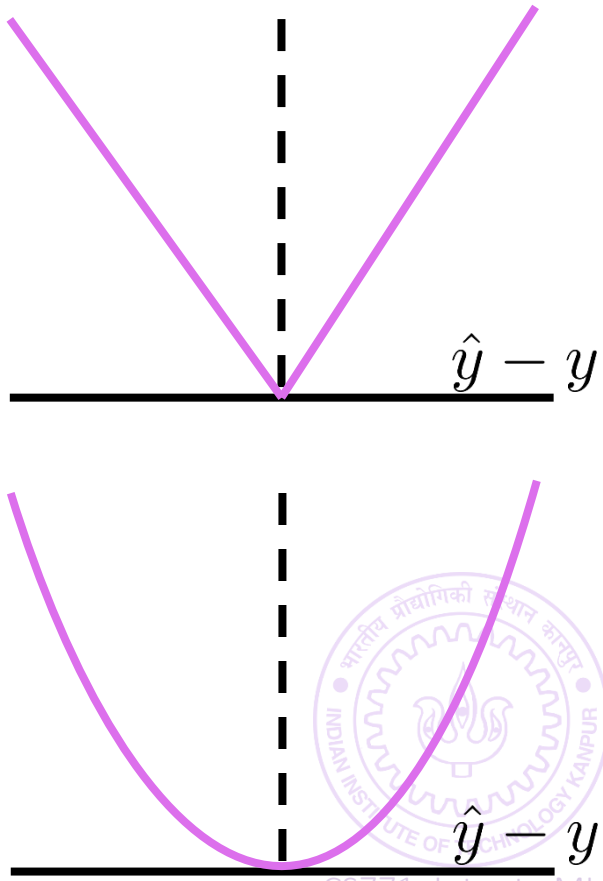
Absolute Loss: $\ell_{\text{abs}}(\hat{y}, y) = |\hat{y} - y|$

Intuition: *a model is doing badly if $\hat{y}$ is either much larger than y or much smaller than y*

Squared Loss: $\ell_{\text{sq}}(\hat{y}, y) = (\hat{y} - y)^2$

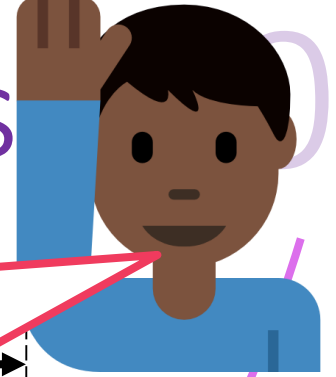
Intuition: *a model is doing badly if $\hat{y}$ is either much larger than y or much smaller than y. Also I want the loss fn to be differentiable so that I can take gradients etc*

Actually, even these loss fns can be derived from basic principles. We will see this soon.



Los

Loss functions have to be chosen according to needs of the problem and experience. Eg. Huber loss popular if some data points are corrupted. However, some loss functions are very popular in ML for various reasons e.g. hinge/cross entropy for binary classification, squared for regression



$\ell_\epsilon(y,$

Vapr

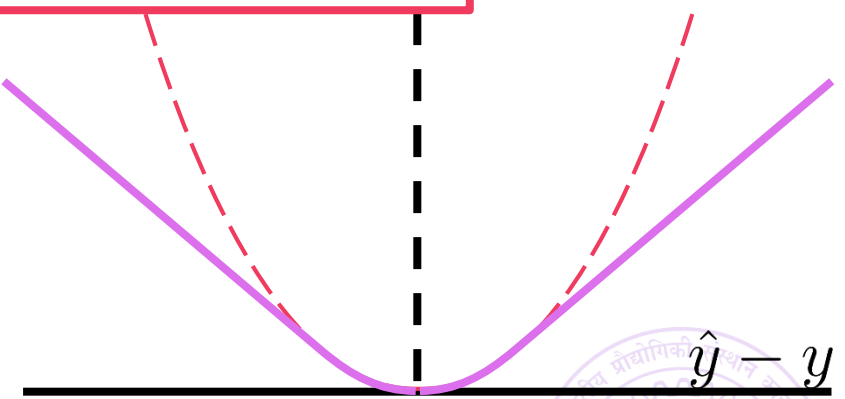
In

p

Indeed, notice that the Huber loss function penalizes large deviations less strictly that the squared loss function (the purple curve is much below the dotted red curve). Doing so may be a good idea in corrupted data settings since it tells the optimizer to not worry too much about corrupted points

uble fn

$$\ell_\delta(y, \hat{y}) = \begin{cases} (\hat{y} - y)^2 & \text{if } |\hat{y} - y| \leq \delta \\ \delta \cdot |y - \hat{y}| & \text{if } |\hat{y} - y| \geq \delta \end{cases}$$

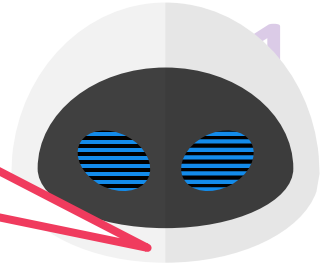


Huber Loss:

Intuition: if a model is doing slightly badly, penalize it as squared loss, if doing very badly, penalize it as absolute loss. Also, please ensure a differentiable fn

R

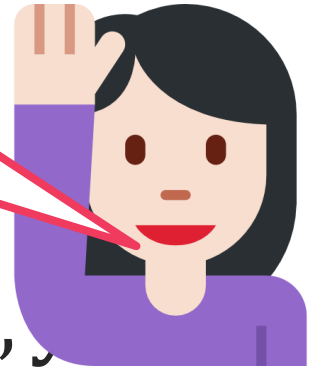
Ridge regression uses least squares loss and L_2 regularization. However the term "least squares regression" is often used to refer to ridge regression ☺



Ignore the bias b for sake of notational simplicity

$\arg$ r
w

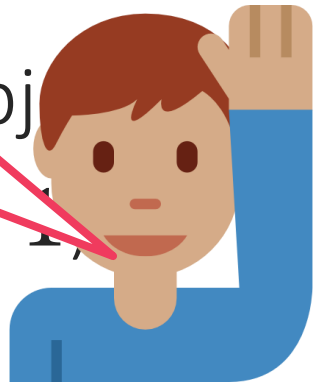
Could have used $\arg\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{w}\|_2^2 + C \cdot \sum_{i=1}^{n}\left(\mathbf{w}^\top \mathbf{x}^i - y^i\right)^2$ too but it is customary in regression to write the optimization problem a bit differently



Can rewrite as $\arg\min_{\mathbf{w}\in\mathbb{R}^d} \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \|X\mathbf{w} - \mathbf{y}\|_2^2$ with $X \in \mathbb{R}^{n\times d}$, y

Can a pj

Even here, we may use dual methods like SDCA (liblinear etc do indeed use them) but yet again, deriving the dual is not as straightforward here since there are no constraints in the original problem

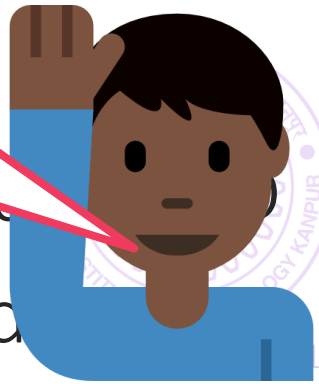


Grad 1,

Gradient must vanish at minimum so we must have

$\mathbf{w} =$

If we want to use fancier loss functions like Vapnik's loss function, then cannot apply first order optimality, need to use SGD, SDCA etc methods



Takes

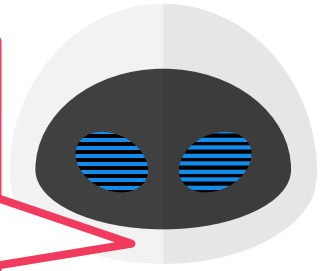
Much faster methods available e.g. conjugate gradient method

Behind the scenes: GD for Ridge Regression

$$f(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{n}($$

$$\nabla f(\mathbf{w}) = \lambda \cdot \mathbf{w} + 2\sum_{i=1}^{n}$$

GD and other optimization techniques merely try to obtain models that obey the rules of good behaviour as encoded in the loss function



$$\mathbf{w}^{\text{new}} = \mathbf{w} - \eta \cdot \nabla f(\mathbf{w}) = (1-\eta\lambda)\cdot\mathbf{w} - 2\eta\sum_{i=1}^{n}\left(\mathbf{w}^\top\mathbf{x}^i - y^i\right)\cdot\mathbf{x}^i$$

Assume $n = 1$ for a moment for sake of understa

$$\mathbf{w}^{\text{new}} = (1-\eta\lambda)\cdot\mathbf{w} - 2\eta(\mathbf{w}^\top\mathbf{x}^1 - y^1)\cdot\mathbf{x}^1$$

Small η: $(1-\eta\lambda)$ is large ⇒ do not change $\mathbf{w}$ too m

If $\boldsymbol{w}$ does well on $(\boldsymbol{x}^1, y^1)$, say $\mathbf{w}^\top\mathbf{x}^1 = y^1$, then $\mathbf{w}^{\text{n}}$

If $\boldsymbol{w}$ does badly on $(\boldsymbol{x}^1, y^1)$, say $\mathbf{w}^\top\mathbf{x}^1 \gg y^1$, then

$$\mathbf{w}^{\text{new}} = (1-\eta\lambda)\cdot\mathbf{w} - 2\eta g\cdot\mathbf{x}^1 \text{ where } g \gg 0$$

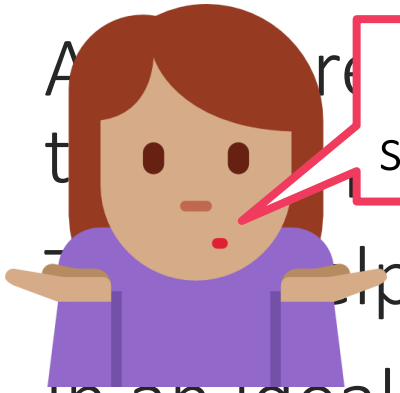
$$(\mathbf{w}^{\text{new}})^\top\cdot\mathbf{x}^1 = (1-\eta\lambda)\cdot\boldsymbol{w}^\top\boldsymbol{x}^1 - 2\eta g\cdot\|\mathbf{x}^1\|_2^2$$

No change to $\mathbf{w}$ due to the data point $(\boldsymbol{x}^1, y^1)$

$(\mathbf{w}^{\text{new}})^\top\cdot\mathbf{x}^1$ is smaller than $\mathbf{w}^\top\mathbf{x}^1$ i.e. may be closer to y^1

If $\mathbf{w}^\top\mathbf{x}^1 \ll y^1$, GD will try to increase the value

Regularization

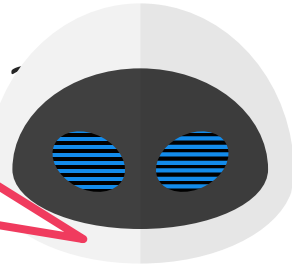


Makes sense since regularization is supposed to protect us from data issues

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

[illegible]lp ML algos offer stable behavi[illegible]

So regularization can be *somewhat* data dependent



In an ideal world where data is perfectly [illegible] data available, there would be no need for any regularization!

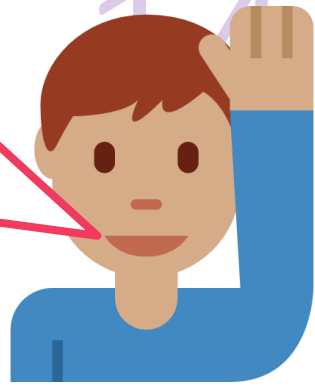
How to do regularization is often decided without looking at data

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

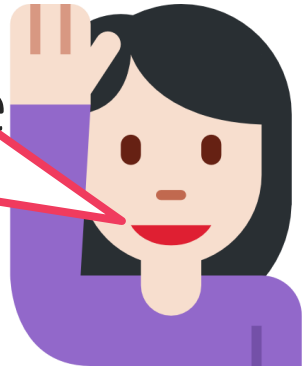
In general, regularization techniques prevent the model from just blindly doing well on data (since data cannot be trusted)

Frequently, regularization can also make the optimization problem well-posed and the solution unique – this helps optimizers (SGD etc) immensely

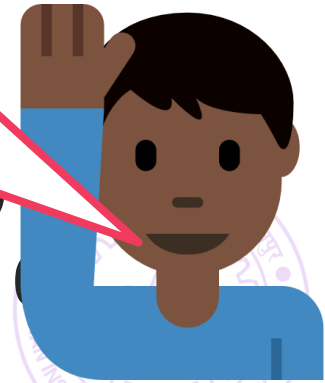
A regularizer essentially tells the optimizer to not just blindly return a model that does well on data (according to the loss function), but rather return a model that does well and is *simple*. The L2 regularizer defines *simplicity* using the L2 norm (or Euclidean length). A model is simple if it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



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 do well on data at all!

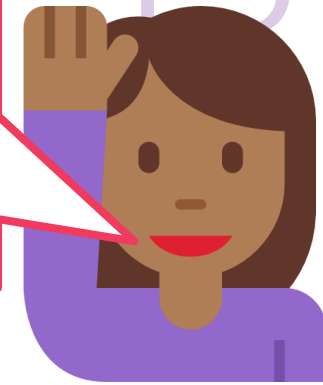


The key is moderation. Usually regularization constants are chosen using validation. Other important considerations include: how noisy do we expect data to be and how much data do we have. **Rule of thumb**: as you have more and more data, you can safely afford to regularize less and less



Othe

$\sum_{i=1}^{n} \boldsymbol{\alpha}_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y^i y^j \langle \mathbf{x}^i, \mathbf{x}^j \rangle$ 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: $f(\mathbf{w}) = \lambda\|\mathbf{w}\|_1 + \sum_{i=1}(\mathbf{w}^\top \mathbf{x}^i - y^i)^2$

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 although *proximal gradient descent* does much better in general

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 from not entire data, but a subset of the data that seems clean

Use a corruption-aware loss function like the Huber loss in robust regression

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 loss function

Multiclass Classification

Sometimes aka multiclassification – have seen kNN and DTs solve it

Can be solved using linear models too!

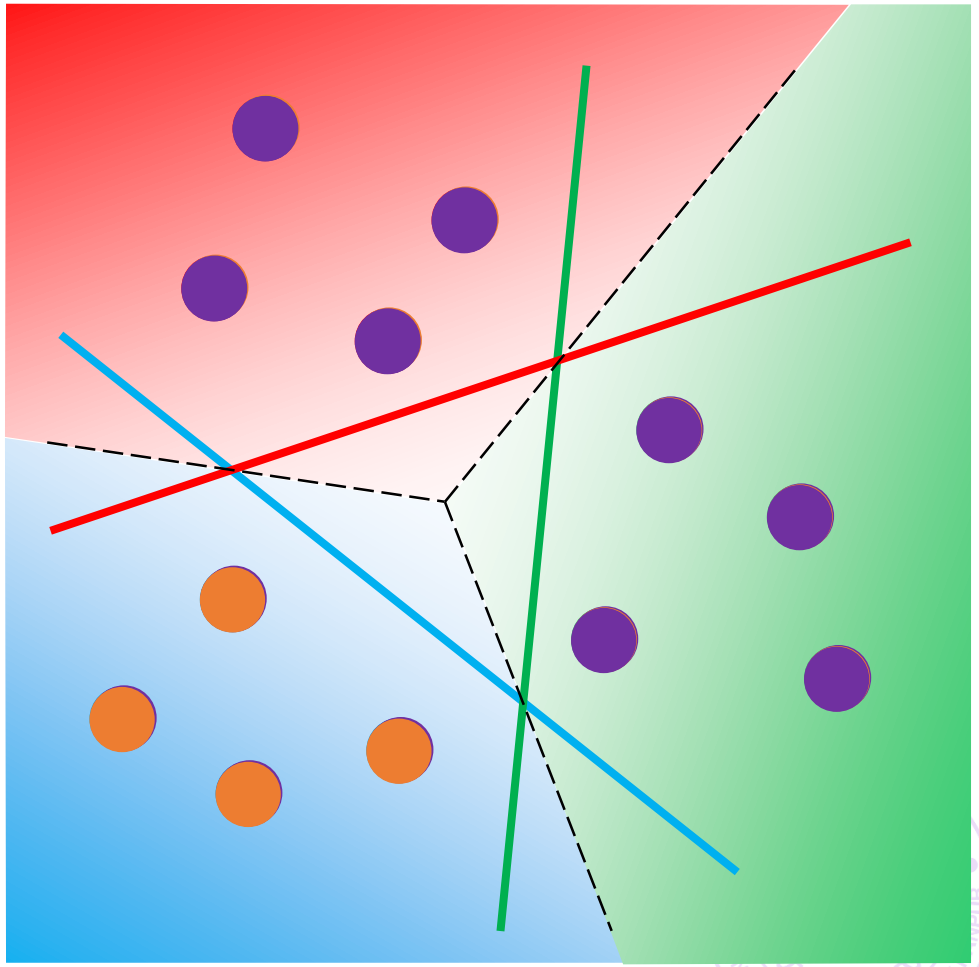
Trick is to reduce to binary classification

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. $\left\langle \mathbf{w}^{y^i}, \mathbf{x}^i \right\rangle \geq \left\langle \mathbf{w}^k, \mathbf{x}^i \right\rangle + 1 - \xi_i$ 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

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

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, especially in deep learning since this is a differentiable function (unlike Crammer-Singer) and so gradients can be taken

Multiclassification – popular techniques

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

OVA: convert multiclassification into several binary problems

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 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