Linear Models II

CS771: Introduction to Machine Learning

Purushottam Kar

Recap of Last Lecture

Looked at SGD as a speedier way to implement GD

Mini-batch SGD as a more stable version of SGD

Dealing with constraints in optimization problems

Projected Gradient Descent: simple but only if projection is easy

Lagrangian Dual Problems: notions of dual variables/Lagrange multipliers, Lagrangian, primal/dual problems

Strong duality, complementary slackness

Looked at the dual problem for the CSVM (with/without bias/slack)

CSVM Dual Problem

If we have a bias b, then the dual problem looks like

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^n}{\text{argmax}} \left\{\sum_{i=1}^{n} \boldsymbol{\alpha}_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y^i y^j \left\langle \mathbf{x}^i, \mathbf{x}^j \right\rangle\right\}$$

s.t. $\boldsymbol{\alpha}_i \in [0, C]$, and $\sum_{i=1}^{n} \boldsymbol{\alpha}_i \, y^i = 0$

If we omit bias (or hide it inside the model vector itself) dual is

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^n}{\text{argmax}} \left\{\sum_{i=1}^{n} \boldsymbol{\alpha}_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y^i y^j \left\langle \mathbf{x}^i, \mathbf{x}^j \right\rangle\right\}$$

s.t. $\boldsymbol{\alpha}_i \in [0, C]$

Let us see a method to solve this problem

Coordinate Descent

Similar to GD except only one coordinate is changed in a single step

For example, let us take the problem $\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x})$ s.t. $\mathbf{x} \in \mathcal{C}$

Let $\nabla_j f(\mathbf{x}) = \frac{\partial f}{\partial \mathbf{x}_j}$ be the jth partial derivative

CCD: choose coordinate cyclically i.e. $j_t = 1,2,\dots,d,1,2,\dots,d,\dots$

SCD: choose j_t randomly

Block CD: choose a small set of coordinates at each t to update

Step length chosen as usual

(PROJECTED) COORDINATE DESCENT

1. For $t = 0,1,\dots$
    1. Select a coordinate $j_t \in [d]$
    2. Let $\mathbf{u}_{j_t}^{t+1} \leftarrow \mathbf{x}_{j_t}^{t} - \eta_t \cdot \nabla_{j_t} f(\mathbf{x}^t)$
    3. Let $\mathbf{u}_{j}^{t+1} \leftarrow \mathbf{x}_{j}^{t}$ for $j \neq j_t$
    4. Project $\mathbf{x}^{t+1} \leftarrow \Pi_{\mathcal{C}}(\mathbf{u}^{t+1})$
    5. Repeat until convergence

Coordinate Descent

Sometimes we are able to optimize completely along a given variable (even if constraints are there) – called *coordinate minimization (CM)*

Will see an example of this today

Coordinate descent popularly applied to dual problems (SDCA – stochastic dual coordinate ascent)

Recall: dual problem is a maximization problem – hence ascent not descent

Beneficial if $n \gg d$ *which is quite often the case*

CD/CA/CM applied to primal problems too if $d \gg n$ since in these cases, SGD takes $\mathcal{O}(d)$ time per step but SCD usually takes only $\mathcal{O}(n)$

Achieving this speedup can require some clever bookkeeping

SCD naively applied in these cases will take $\mathcal{O}(nd)$ *time per step*

SDCA for the CSVM Problem

Consider the version that does not have a bias

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^n}{\operatorname{argmax}} \left\{\sum_{i=1}^{n} \boldsymbol{\alpha}_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y^i y^j \left\langle \mathbf{x}^i, \mathbf{x}^j \right\rangle\right\}$$

s.t. $\boldsymbol{\alpha}_i \in [0, C]$ for all $i \in [n]$

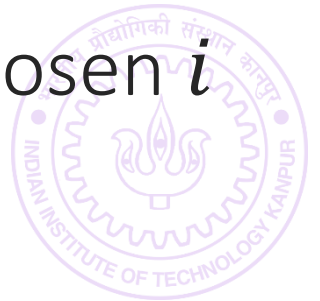
The version with bias has constraint $\sum_{i=1}^{n} \boldsymbol{\alpha}_i \, y^i = 0$ that links all $\boldsymbol{\alpha}_i$ together

Cannot update a single $\boldsymbol{\alpha}_i$ without disturbing all the others ☹

Need a more involved algorithm Sequential Minimal Optimization (SMO) by John Platt to solve the version with a bias – update two $\boldsymbol{\alpha}_i$ at a time!

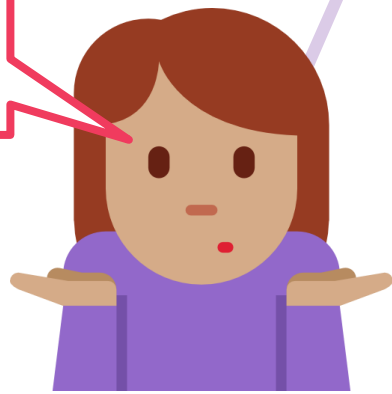
To apply SCA, at each step, we need to choose one $i \in [n]$

In this case, possible to completely maximize objective w.r.t. chosen i



SDCA for the CSVM P

Does this mean I need to choose one data point at each time step?

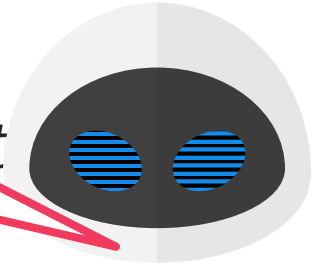


Consider the version that does not have a bias

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^n}{\text{argmax}} \left\{\sum_{i=1}^{n} \boldsymbol{\alpha}_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y^i y^j \left\langle \mathbf{x}^i, \mathbf{x}^j \right\rangle\right\}$$

s.t. $\boldsymbol{\alpha}_i \in [0, C]$ for all $i \in [n]$

Yes, coordinate ascent in the dual looks a lot like stochastic gradient descent in the primal! Both work with a single data point at a time



The ve … *t*

Cannot update a single $\boldsymbol{\alpha}_i$ *without disturbing all the others* ☹

Need a more involved algorithm Sequential Minimal Optimization (SMO) by John Platt to solve the version with a bias – update two $\boldsymbol{\alpha}_i$ *at a time!*

To apply SCA, at each step, we need to choose one $i \in [n]$

In this case, possible to completely maximize objective w.r.t. chosen i

SDCA for the CSVM Problem

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^n}{\operatorname{argmax}} \left\{\sum_{i=1}^{n} \boldsymbol{\alpha}_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y^i y^j \left\langle \mathbf{x}^i, \mathbf{x}^j \right\rangle\right\}$$
s.t. $\boldsymbol{\alpha}_i \in [0, C]$ for all $i \in [n]$

Concentrating on just the terms that involve $\boldsymbol{\alpha}_i$ we get

$$\underset{\boldsymbol{\alpha}_i\in\mathbb{R}}{\operatorname{argmax}} \left\{\boldsymbol{\alpha}_i - \frac{1}{2}\boldsymbol{\alpha}_i^2 \left\|\mathbf{x}^i\right\|_2^2 - \boldsymbol{\alpha}_i y^i \cdot \sum_{j\neq i} \boldsymbol{\alpha}_j y^j \left\langle \mathbf{x}^i, \mathbf{x}^j \right\rangle\right\}$$
s.t. $\boldsymbol{\alpha}_i \in [0, C]$

Renaming $x = \boldsymbol{\alpha}_i, q = \left\|\mathbf{x}^i\right\|_2^2, p = y^i \cdot \sum_{j\neq i} \boldsymbol{\alpha}_j y^j \left\langle \mathbf{x}^i, \mathbf{x}^j \right\rangle$, we get

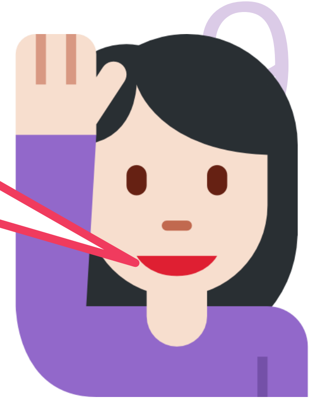
$$\arg\min_x \frac{1}{2} q x^2 - x(1-p) \text{ s.t. } x \in [0, C]$$

Solution is very simple: find unrestricted minimum i.e. $\tilde{x} = (1-p)/q$

If $\tilde{x} \in [0, C]$, solution is $\tilde{x}$ elif $\tilde{x} < 0$, solution is 0, else solution is C

SDCA for th

Warning: in general, finding an unconstrained solution and doing a projection step **does not** give a true solution



$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^n}{\text{argmax}} \left\{\sum_{i=1}^n \boldsymbol{\alpha}_i - \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \boldsymbol{\alpha}_i\boldsymbol{\alpha}_j y^i y^j \langle \mathbf{x}^i, \mathbf{x}^j\rangle\right\}$$

s.t. $\boldsymbol{\alpha}_i \in [0, C]$

Concentrating

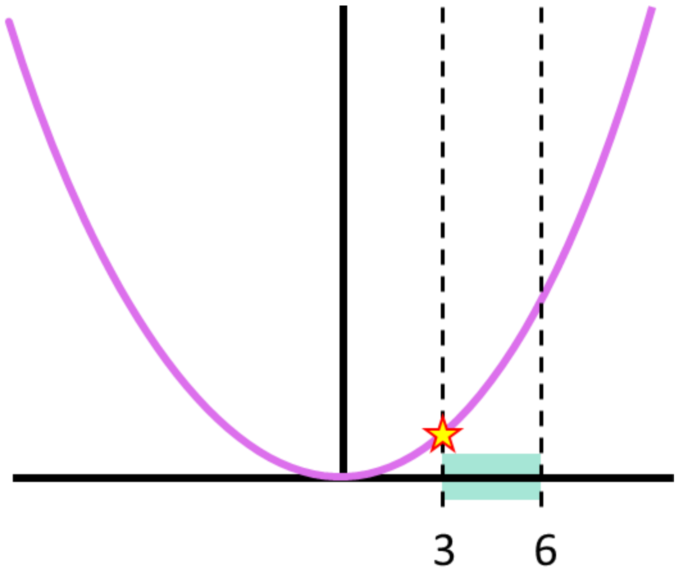
$$\underset{\boldsymbol{\alpha}_i\in\mathbb{R}}{\text{argmax}} \left\{\boldsymbol{\alpha}_i -\right.$$

s.t. $\boldsymbol{\alpha}_i \in [0, C]$

$$\min_x x^2$$

s.t. $x \geq 6$

and $x \leq 3$



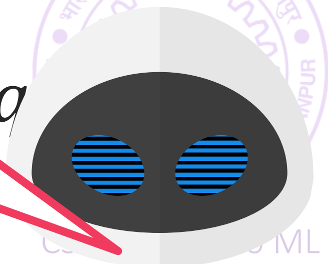
Renaming $x =$ ... $\mathbf{x}^j\rangle$, we get

$$\arg\min_x \frac{1}{2}qx^2 - x(1-p) \text{ s.t. } x \in [0, C]$$

Solution is very simple: fin ... $/q$

If $\tilde{x} \in [0, C]$, *solution is* $\tilde{x}$

Indeed! In this special case, our objective had a nice property called *unimodality* which is why this trick works – it won't work in general



Speeding up SDCA computations

All that is left is to find how to compute p, q four our chosen i

$q = \left\|\mathbf{x}^i\right\|_2^2$ can be easily precomputed for all data points

However, $p = y^i \cdot \sum_{j \neq i} \boldsymbol{\alpha}_j y^j \left\langle \mathbf{x}^i, \mathbf{x}^j \right\rangle$ needs $\mathcal{O}(nd)$ time to compute ☹

… only if done naively. Recall that we always have $\mathbf{w} = \sum_{i=1}^n \boldsymbol{\alpha}_i y^i \cdot \mathbf{x}^i$ for the CSVM (even if we have bias and slack variables)

Thus, $\sum_{j \neq i} \boldsymbol{\alpha}_j y^j \left\langle \mathbf{x}^i, \mathbf{x}^j \right\rangle = \mathbf{w}^\top \mathbf{x}^i - \boldsymbol{\alpha}_i y^i \left\|\mathbf{x}^i\right\|_2^2 = \mathbf{w}^\top \mathbf{x}^i - \boldsymbol{\alpha}_i y^i q$

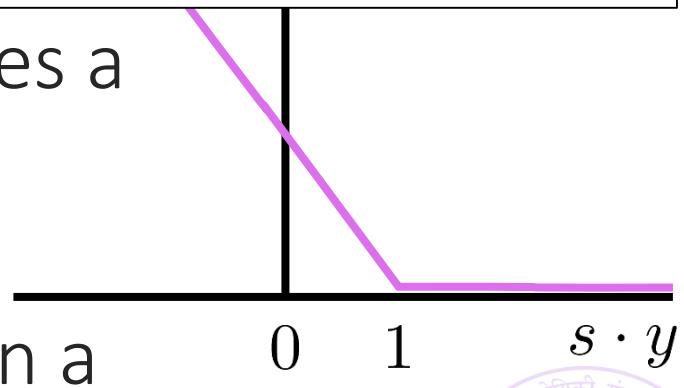
All we need to do is create (and update) the $\mathbf{w}$ vector in addition to the $\boldsymbol{\alpha}$ vector and we would be able to find p in just $\mathcal{O}(d)$ time ☺

Loss Functions

Loss functions are widely used in ML to encode what behaviour we desire from our model

Loss functions penalize undesirable behaviour and when we use an optimizer like SGD or SDCA, we (hopefully) end up with a model which has desirable behaviour

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



The hinge function is a loss function which penalizes a model if it either misclassifies a data point or else correctly classifies it with insufficient margin

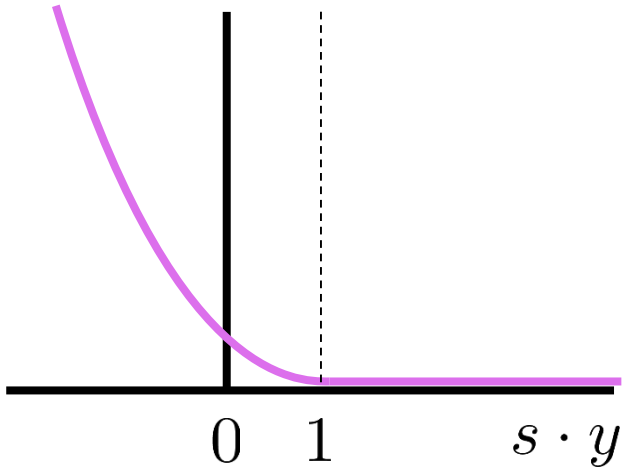
Note: hinge loss tells us how well is model doing on a single data point $\left(\mathbf{x}^i, y^i\right)$. We take sum/avg of these datapoint-wise loss values on the entire training set to see how well is model doing on the entire training set i.e. look at $\sum_{i=1}^{n} \ell_{\text{hinge}}\left(y^i \cdot \mathbf{w}^\top \mathbf{x}^i\right)$

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

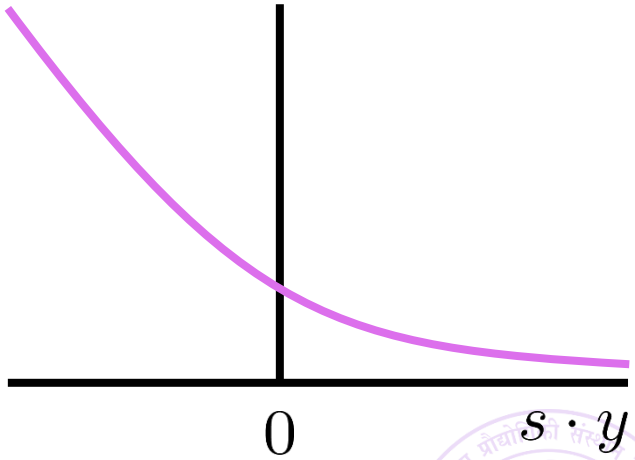


Logistic Loss Function:

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

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 has to be done a bit cleverly here since no constraints ☺

Will revisit logistic regression very soon and derive the loss function

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 $\mathbf{x}^i \in \mathbb{R}^d, y \in \mathbb{R}$

For example, predict price of a stock, predict *change* in price of a stock, predict test scores of a student 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

Need a different notion of "purity"

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

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* $\bar{y} = \frac{1}{\mathrm{n}} \sum_{\mathrm{i}=1}^{n} y^i$ *and use* $\sum_{i=1}^{n} \left| y^i - \bar{y} \right|$ *as purity*

Purity definition 4: *Let* $\bar{y} = \frac{1}{\mathrm{n}} \sum_{\mathrm{i}=1}^{n} y^i$ *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

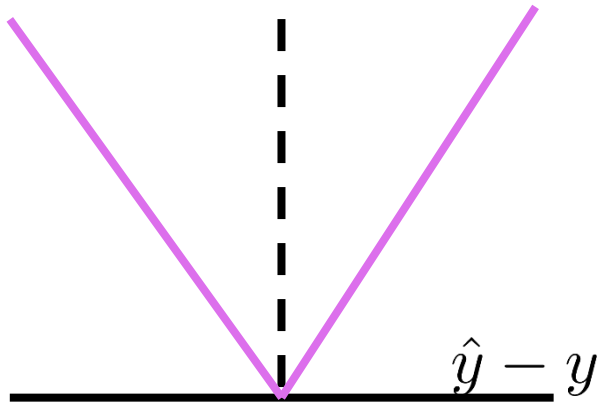
Loss Functions for Regression Problems

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

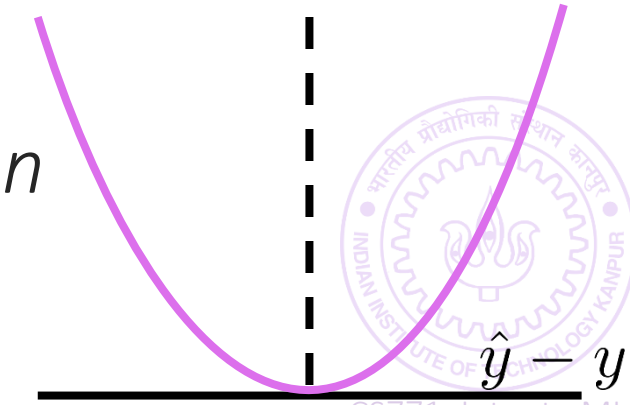
Absolute Loss: $\ell_{\text{abs}}(\hat{y}, y) = |\hat{y} - y|$

A model is doing badly if $\hat{y}$ is either much larger than y or much smaller than y



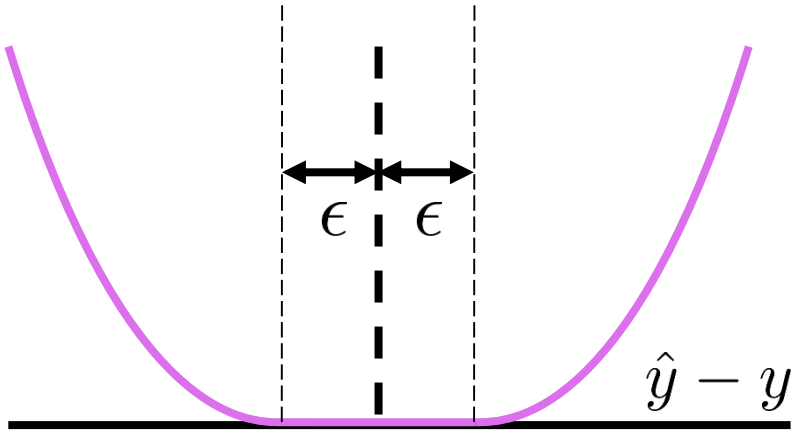
Squared Loss: $\ell_{\text{sq}}(\hat{y}, y) = (\hat{y} - y)^2$

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



Loss Functions for Regression Problems

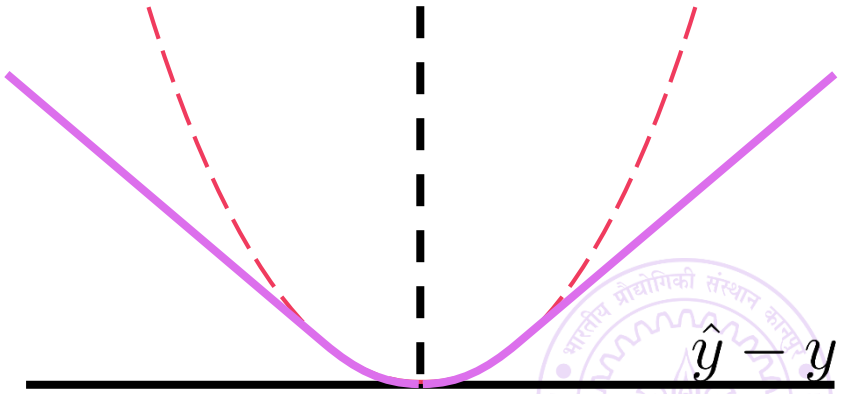
$$\ell_\epsilon(y, \hat{y}) = \begin{cases} (y - \hat{y} - \epsilon)^2 & \text{if } \hat{y} < y - \epsilon \\ 0 & \text{if } \hat{y} - y \in [-\epsilon, \epsilon] \\ (y - \hat{y} + \epsilon)^2 & \text{if } \hat{y} > y + \epsilon \end{cases}$$



Vapnik's ϵ-insensitive Loss:

If a model is doing slightly badly, don't penalize it at all, else penalize it as squared loss. Ensure a differentiable function

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

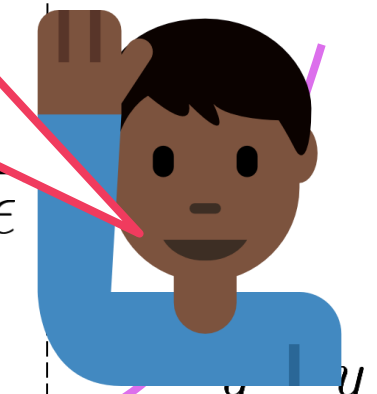


Huber Loss:

If a model is doing slightly badly, penalize it as squared loss, if doing very badly, penalize it as absolute loss. Ensure a differentiable function

Los s

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



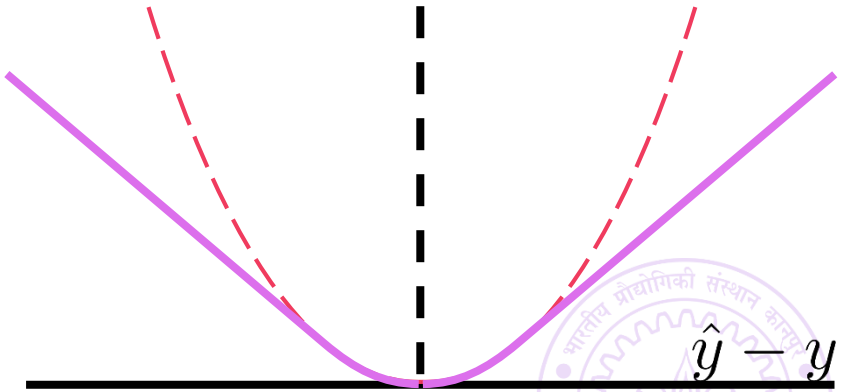
$\ell_\epsilon(y,$

$$\begin{cases} (y - \hat{y} + \epsilon)^2 & \text{if } \hat{y} > y + \epsilon \end{cases}$$

Vapnik's ϵ-insensitive Loss:

If a model is doing slightly badly, don't penalize it at all, else penalize it as squared loss. Ensure a differentiable function

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



Huber Loss:

If a model is doing slightly badly, penalize it as squared loss, if doing very badly, penalize it as absolute loss. Ensure a differentiable function

Least Squares Regression

Ignore the bias b for sake of notational simplicity

$$\arg\min_{\mathbf{w}\in\mathbb{R}^d} \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{n}\left(\mathbf{w}^\top\mathbf{x}^i - y^i\right)^2$$

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 $X \in \mathbb{R}^{n\times d}, \mathbf{y}\in\mathbb{R}^n$

Can apply first order optimality to obtain a solution (convex objective)

Gradient: $\lambda\cdot\mathbf{w} + 2X^\top(X\mathbf{w}-\mathbf{y})$ (recall: gradient must be $d\times 1$)

Gradient must vanish at minimum so we must have

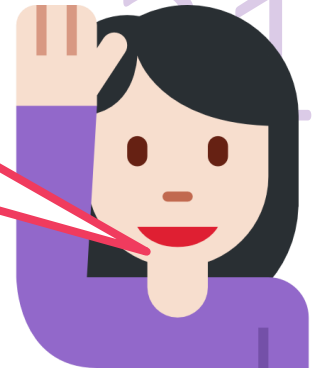
$$\mathbf{w} = (2X^\top X + \lambda\cdot I)^{-1}(2X^\top\mathbf{y})$$

Takes $\mathcal{O}(d^3)$ time to invert the matrix – may use faster methods (S)GD

Much faster methods available e.g. conjugate gradient method

Lea

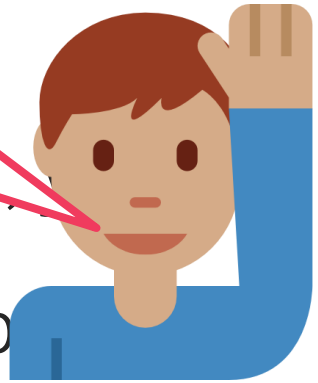
Could have used $\arg\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{w}\|_2^2 + C \cdot \sum_{i=1}^{n}\left(\mathbf{w}^\top \mathbf{x}^i - y^i\right)^2$ too but it is customary in regression to write the optimization problem a bit differently



Ignore the bias b for sake of notational simplicity

$\arg$

Even here, we may use dual methods like SDCA (liblinear etc do indeed use them) but need to carefully derive the dual since no constraints



Can

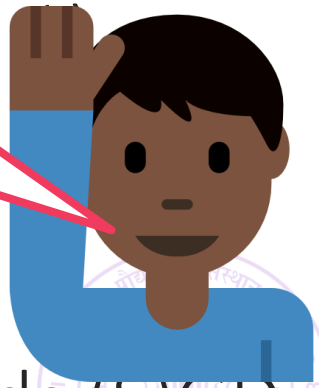
$\mathbf{w}\in\mathbb{R}^d$

Can apply first order optimality to obtain a solution (convex ob

Grad

Grad

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



$$\mathbf{w} = (2X^\top X + \lambda \cdot I)^{-1}(2X^\top \mathbf{y})$$

Takes $\mathcal{O}(d^3)$ time to invert the matrix – may use faster methods (S)GD

Much faster methods available e.g. conjugate gradient method

Behind the scenes in GD for Least Squares

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

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

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

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

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

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

If $\boldsymbol{w}$ does well on $(\boldsymbol{x}^1, y^1)$, say $\mathbf{w}^\top \mathbf{x}^1 = y^1$, then $\mathbf{w}^{\text{new}} = (1 - \eta\lambda) \cdot \mathbf{w}$

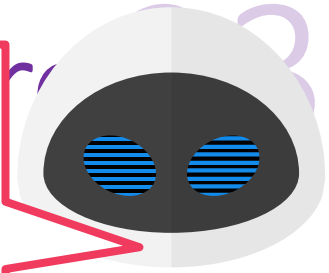
If $\boldsymbol{w}$ does badly on $(\boldsymbol{x}^1, y^1)$, say $\mathbf{w}^\top \mathbf{x}^1 \gg y^1$, then

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

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

Behind the scen

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



$$f(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{n}\left(\text{[illegible]}\right)$$

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

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

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

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