Linear Models and Learning via Optimization

CS771: Introduction to Machine Learning

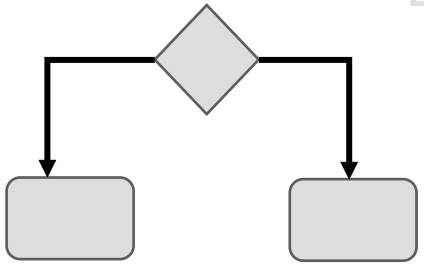
Piyush Rai

Announcements

- Quiz 1 postponed by a week (now on Aug 23)
  - Venue TBD, timing: 7pm, duration: 45 minutes
  - Syllabus will be everything up to Aug 21 class
- Homework 1 released by end of this week

Wrapping Up Decision Trees..

Avoiding Overfitting in DTs


A Decision Stump

- Desired: a DT that is not too big in size, yet fits the training data reasonably
- Note: An example of a very simple DT is "decision-stump"
  - A decision-stump only tests the value of a single feature (or a simple rule)
  - Not very powerful in itself but often used in a large ensemble of decision stumps
- Some ways to keep a DT simple enough:
  - Control its complexity while building the tree (stopping early)
  - Prune after building the tree (post-pruning)
- Criteria for judging which nodes could potentially be pruned (from already built complex DT)
  - Use a validation set (separate from the training set)
    - Prune each possible node that doesn't hurt the accuracy on the validation set
    - Greedily remove the node that improves the validation accuracy the most
    - Stop when the validation set accuracy starts worsening

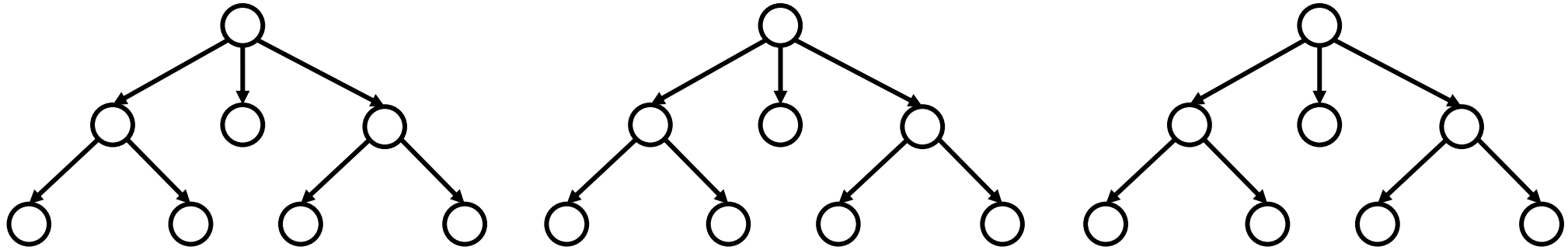
Ensemble of Trees

- Ensemble is a collection of models. Popular in ML
- Each model makes a prediction. Take their majority as the final prediction
- Ensemble of trees is a collection of simple DTs
  - Often preferred as compared to a single massive, complicated tree
- A popular example: Random Forest (RF)

All trees can be trained in parallel

Each tree is trained on a subset of the training inputs/features

An RF with 3 simple trees. The majority prediction will be the final prediction



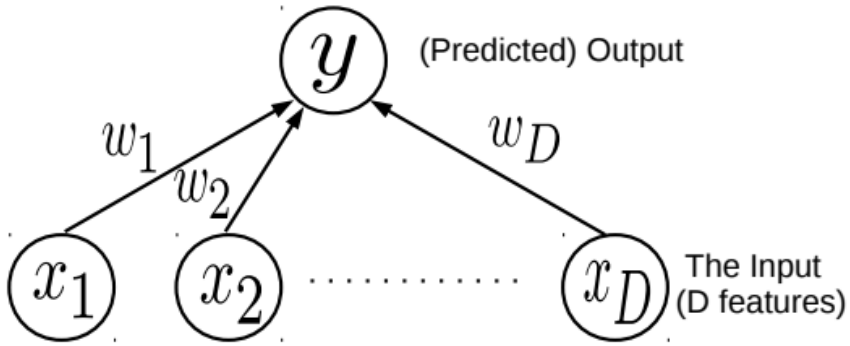
- XGBoost is another popular ensemble of trees
  - Based on the idea of "boosting" (will study boosting later) simple trees
  - Sequentially trains a set of trees with each correcting errors of previous ones

Linear Models and Learning via Optimization

Linear Models

- Suppose we want to learn to map inputs $\boldsymbol{x} \in \mathbb{R}^D$ to real-valued outputs $\boldsymbol{y} \in \mathbb{R}$
- Linear model: Assume output to be a linear weighted combination of the D input features

$$y = \sum_{d=1}^{D} w_d x_d = \boldsymbol{w}^\top \boldsymbol{x}$$

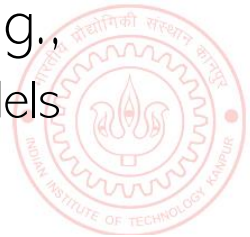


This defines a linear model with D parameters given by a "weight vector" $\boldsymbol{w} = [w_1, w_2, \ldots, w_D]$

Each of these weights have a simple interpretation: $\boldsymbol{w_d}$ is the "weight" or contribution of the $\boldsymbol{d^{th}}$ feature in making this prediction

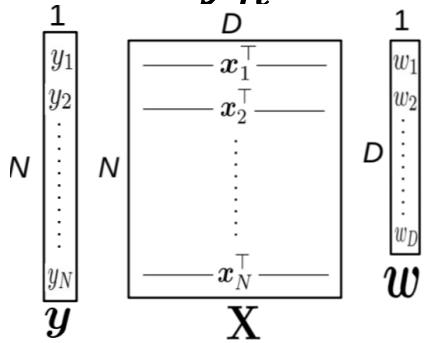
The "optimal" weights are unknown and have to be learned by solving an optimization problem, using some **training data**

- This simple model can be used for Linear Regression
- This simple model can also be used as a "building block" for more complex models, e.g.,
  - Classification (binary/multiclass/multi-output/multi-label) and various other ML/deep learning models
  - Unsupervised learning problems (e.g., dimensionality reduction models)



Linear Regression

- Given: Training data with N input-output pairs $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, $\boldsymbol{x}_n \in \mathbb{R}^D$, $y_n \in \mathbb{R}$



- Goal: Learn a model to predict the output for new test inputs

- Assume the function that approximates the I/O relationship to be a linear model

$$y_n \approx f(\boldsymbol{x}_n) = \boldsymbol{w}^\top \boldsymbol{x}_n \quad (n = 1,2,\ldots,N)$$

Can also write all of them compactly using matrix-vector notation as $\boldsymbol{y} \approx \boldsymbol{X}\boldsymbol{w}$

- Let's write the total error or "loss" of this model over the training data as

$$L(\boldsymbol{w}) = \sum_{n=1}^N \ell(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n)$$

Goal of learning is to find the $\boldsymbol{w}$ that minimizes this loss + does well on test data

Unlike models like KNN and DT, here we have an explicit problem-specific objective (loss function) that we wish to optimize for

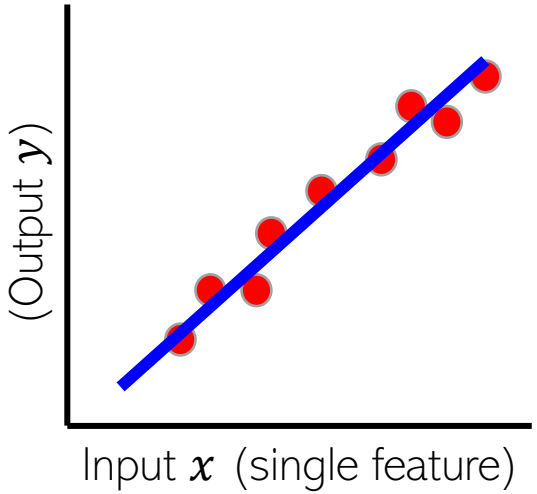
$\ell(y_n, \boldsymbol{w}^\top \boldsymbol{x}_n)$ measures the prediction error or "loss" or "deviation" of the model on a single training input $(\boldsymbol{x}_n, y_n)$

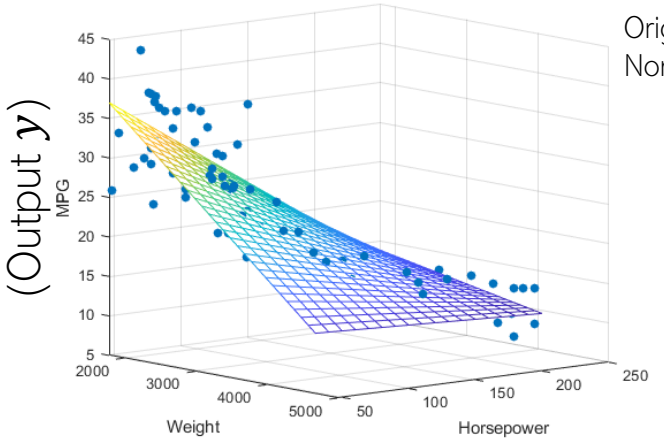
Linear Regression: Pictorially

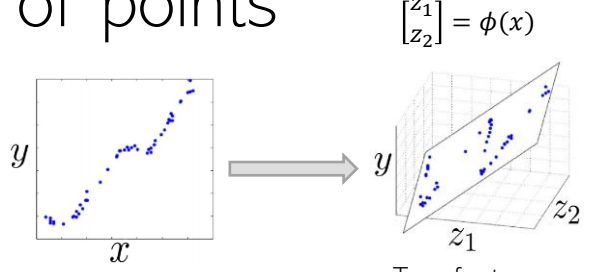
- Linear regression is like fitting a line or (hyper)plane to a set of points

What if a line/plane doesn't model the input-output relationship very well, e.g., if their relationship is better modeled by a nonlinear curve or curved surface?

Do linear models become useless in such cases?







Original (single) feature
Nonlinear curve needed

Two features
Can fit a plane (linear)

No. We can even fit a curve using a linear model after suitably transforming the inputs

$$y \approx \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})$$

The transformation $\boldsymbol{\phi}(.)$ can be predefined or learned (e.g., using kernel methods or a deep neural network based feature extractor). More on this later

- The line/plane must also predict outputs of the unseen (test) inputs well

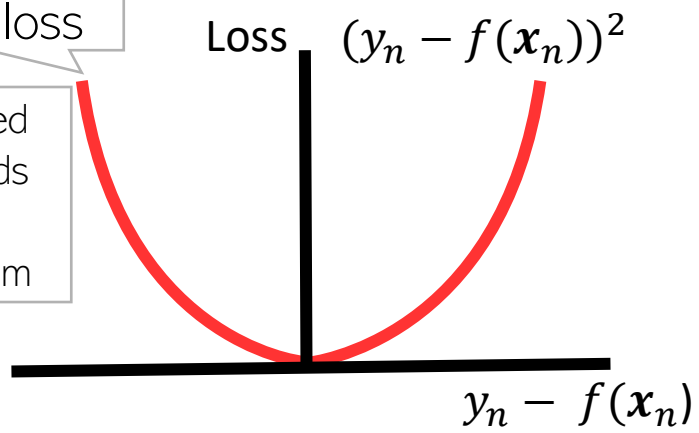
Loss Functions for Regression

- Many possible loss functions for regression problems

Choice of loss function usually depends on the nature of the data. Also, some loss functions result in easier optimization problem than others

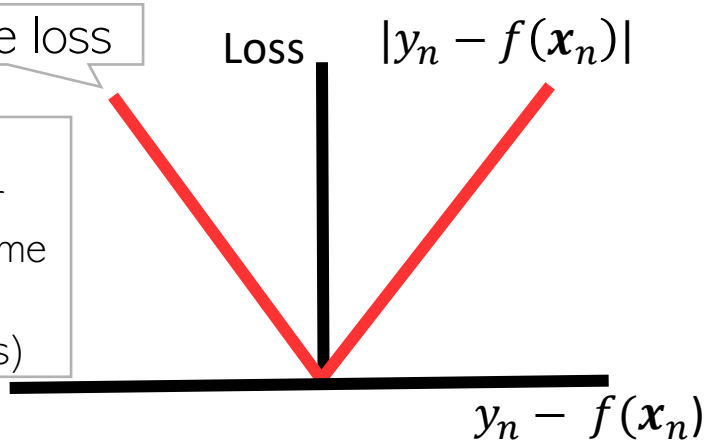
Squared loss

Very commonly used for regression. Leads to an easy-to-solve optimization problem



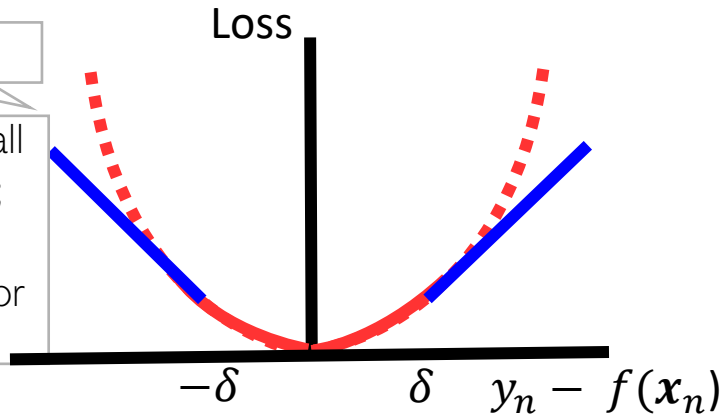
Absolute loss

Grows more slowly than squared loss. Thus better suited when data has some outliers (inputs on which model makes large errors)



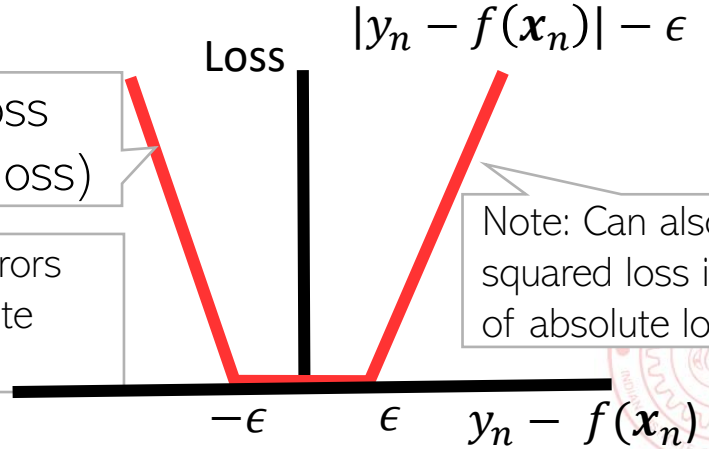
Huber loss

Squared loss for small errors (say up to δ); absolute loss for larger errors. Good for data with outliers



ϵ-insensitive loss (a.k.a. Vapnik loss)

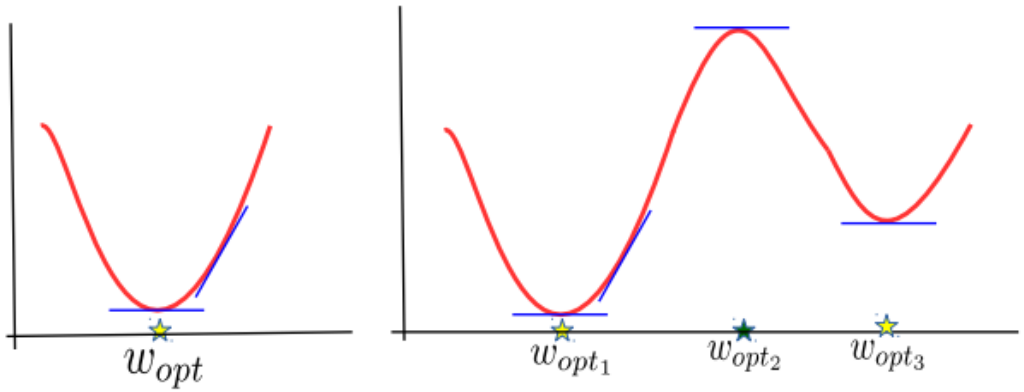
Zero loss for small errors (say up to ϵ); absolute loss for larger errors



Note: Can also use squared loss instead of absolute loss

Minimizing Loss Func using First-Order Optimality

- Use basic calculus to find the minima



Called "first order" since only gradient is used and gradient provides the first order info about the function being optimized

The approach works only for very simple problems where the objective is convex and there are no constraints on the values $\boldsymbol{w}$ can take

- First order optimality: The gradient $\boldsymbol{g}$ must be equal to zero at the optima

$$\boldsymbol{g} = \nabla_{\boldsymbol{w}}[L(\boldsymbol{w})] = \mathbf{0}$$

- Sometimes, setting $\boldsymbol{g} = \mathbf{0}$ and solving for $\boldsymbol{w}$ gives a closed form solution
- If closed form solution is not available, the gradient vector $\boldsymbol{g}$ can still be used in iterative optimization algos, like gradient descent (GD)
  - Note: Even if closed-form solution is possible, GD can sometimes be more efficient

Minimizing Loss Func. using Iterative Optimization

Can I used this approach to solve maximization problems?

Fact: Gradient gives the direction of steepest change in function's value

For max. problems we can use gradient ascent

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t \boldsymbol{g}^{(t)}$$

Will move in the direction of the gradient

Iterative since it requires several steps/iterations to find the optimal solution

For convex functions, GD will converge to the global minima

Good initialization needed for non-convex functions

Gradient Descent

- Initialize $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)}$
- For iteration $t = 0,1,2, \ldots$ (or until convergence)
  - Calculate the gradient $\boldsymbol{g}^{(t)}$ using the current iterates $\boldsymbol{w}^{(t)}$
  - Set the learning rate η_t
  - Move in the opposite direction of gradient

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$$

The learning rate very imp. Should be set carefully (fixed or chosen adaptively). Will discuss some strategies later

Sometimes may be tricky to to assess convergence. Will see some methods later

Linear Regression with Squared Loss

- In this case, the loss func will be

In matrix-vector notation, can write it compactly as $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})$

$$L(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

- Let us find the $\boldsymbol{w}$ that optimizes (minimizes) the above squared loss

The "least squares" (LS) problem Gauss-Legendre, 18th century)

- Let's use first order optimality
- The LS problem can be solved easily and has a closed form solution

Closed form solutions to ML problems are rare.

$$\boldsymbol{w}_{LS} = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

$$\boldsymbol{w}_{LS} = \left(\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top\right)^{-1}\left(\sum_{n=1}^{N} y_n \boldsymbol{x}_n\right) = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}$$

$D \times D$ matrix inversion – can be expensive. Ways to handle this. Will see later

Proof: A bit of calculus/optim. (more on this later)

- We wanted to find the minima of $L(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$

- Let us apply basic rule of calculus: Take first derivative of $L(\boldsymbol{w})$ and set to zero

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{\partial}{\partial \boldsymbol{w}} \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 = \sum_{n=1}^{N} 2(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n) \frac{\partial}{\partial \boldsymbol{w}}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n) = 0$$

Chain rule of calculus

Partial derivative of dot product w.r.t each element of $\boldsymbol{w}$

Result of this derivative is $\boldsymbol{x}_n$ - same size as $\boldsymbol{w}$

- Using the fact $\frac{\partial}{\partial \boldsymbol{w}} \boldsymbol{w}^\top \boldsymbol{x}_n = \boldsymbol{x}_n$, we get $\sum_{n=1}^{N} 2(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\boldsymbol{x}_n = 0$

- To separate $\boldsymbol{w}$ to get a solution, we write the above as

$$\sum_{n=1}^{N} 2\boldsymbol{x}_n(y_n - \boldsymbol{x}_n^\top \boldsymbol{w}) = 0 \quad \Rightarrow \quad \sum_{n=1}^{N} y_n \boldsymbol{x}_n - \boldsymbol{x}_n \boldsymbol{x}_n^\top \boldsymbol{w} = 0$$

$$\boldsymbol{w}_{LS} = \left(\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top\right)^{-1} \left(\sum_{n=1}^{N} y_n \boldsymbol{x}_n\right) = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

Problem(s) with the Solution!

- We minimized the objective $L(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$ w.r.t. $\boldsymbol{w}$ and got

$$\boldsymbol{w}_{LS} = (\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top)^{-1} (\sum_{n=1}^{N} y_n \boldsymbol{x}_n) = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

- Problem: The matrix $\boldsymbol{X}^\top \boldsymbol{X}$ may not be invertible
  - This may lead to non-unique solutions for $\boldsymbol{w}_{opt}$
- Problem: Overfitting since we only minimized loss defined on training data
  - Weights $\boldsymbol{w} = [w_1, w_2, \ldots, w_D]$ may become arbitrarily large to fit training data perfectly
  - Such weights may perform poorly on the test data however
- One Solution: Minimize a **regularized objective** $L(\boldsymbol{w}) + \lambda R(\boldsymbol{w})$
  - The reg. will prevent the elements of $\boldsymbol{w}$ from becoming too large
  - Reason: Now we are minimizing training error + magnitude of vector $\boldsymbol{w}$

$R(\boldsymbol{w})$ is called the Regularizer and measures the "magnitude" of $\boldsymbol{w}$

$\lambda \geq 0$ is the reg. hyperparam. Controls how much we wish to regularize (needs to be tuned via cross-validation)

Regularized Least Squares (a.k.a. Ridge Regression)

- Recall that the regularized objective is of the form $L_{reg}(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda R(\boldsymbol{w})$

- One possible/popular regularizer: the squared Euclidean (ℓ_2 squared) norm of $\boldsymbol{w}$

$$R(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2 = \boldsymbol{w}^\top \boldsymbol{w}$$

- With this regularizer, we have the regularized least squares problem as

$$\boldsymbol{w}_{ridge} = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

$$= \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

Look at the form of the solution. We are adding a small value λ to the diagonals of the DxD matrix $\boldsymbol{X}^\top \boldsymbol{X}$ (like adding a ridge/mountain to some land)

Why is the method called "ridge" regression

- Proceeding just like the LS case, we can find the optimal $\boldsymbol{w}$ which is given by

$$\boldsymbol{w}_{ridge} = \left(\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top + \lambda I_D\right)^{-1} \left(\sum_{n=1}^{N} y_n \boldsymbol{x}_n\right) = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda I_D)^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

Other Ways to Control Overfitting

- Use a regularizer $R(\boldsymbol{w})$ defined by other norms, e.g.,

ℓ_1 norm regularizer

$$\|\boldsymbol{w}\|_1 = \sum_{d=1}^{D} |w_d|$$

$$\|\boldsymbol{w}\|_0 = \#\text{nnz}(\boldsymbol{w})$$

ℓ_0 norm regularizer (counts number of nonzeros in $\boldsymbol{w}$

When should I used these regularizers instead of the ℓ_2 regularizer?

Automatic feature selection? Wow, cool!!! But how exactly?

Note that optimizing loss functions with such regularizers is usually harder than ridge reg. but several advanced techniques exist (we will see some of those later)

Use them if you have a very large number of features but many irrelevant features. These regularizers can help in automatic feature selection

Using such regularizers gives a sparse weight vector $\boldsymbol{w}$ as solution

sparse means many entries in $\boldsymbol{w}$ will be zero or near zero. Thus those features will be considered irrelevant by the model and will not influence prediction

- Use non-regularization based approaches
  - Early-stopping (stopping training just when we have a decent val. set accuracy)
  - Dropout (in each iteration, don't update some of the weights)
  - Injecting noise in the inputs

All of these are very popular ways to control overfitting in deep learning models. More on these later when we talk about deep learning

Gradient Descent for Linear/Ridge Regression

- Just use the GD algorithm with the gradient expressions we derived ☺
- Iterative updates for linear regression will be of the form

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$$

$$= \boldsymbol{w}^{(t)} + \eta_t \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\boldsymbol{x}_n$$

Unlike the closed form solution of least squares regression, here we have iterative updates but do not require the expensive matrix inversion of the $D \times D$ matrix $\boldsymbol{X}^\top \boldsymbol{X}$

- Similar updates for ridge regression as well (with the gradient expression being slightly different; left as an exercise)

- More on iterative optimization methods later