CONVEX OPTIMISATION

MAT 220

Dr.C. Rajan (8113053359) Off: S109E

APPLICATIONS

Convex functions find applications in lots of fields:

1. digital signal processing
2. Optimisation problems
3. Circuit design
4. Communication networks
5. Data analytics
6. Economies, inventory control

DIMENSIONALITY REDUCTION

What is Dimensionality reduction?

Where is it used?

Why do we need it?

DIMENSIONALITY REDUCTION

Dimensionality reduction is a technique to reduce the data dimension

This is equivalent to reducing the number of variables

The most popular statistical techniques are principal component analysis (PCA), independent component analysis (ICA), factor analysis (FA) etc.

But optimization problems use another technique called sparsity inducing constraints to reduce the number of variables.

Large optimization problems have hundreds of variables. Our aim is to identify most important variables by inducing sparsity constraints.

SPARCITY INDUCING PENALTY FUNCTIONS

principle of parsimony is important in many optimization problems

There could be several variables or constraints that are redundant.

One way to reduce redundancy is using penalty functions, regularization by the L1-norm

Best sparse approximation may suffice in some problems over an exact solution that is computationally hard to find.

One example is variable selection in linear models. We could reduce the number of variables by 10 to 20 percent using proper penalty functions that penalize empirical risk or the log-likelihood

ADVANTAGES OF SPARSITY INDUCING

Sparse estimation problems can be cast as convex optimization problems

it leads to efficient estimation algorithms

it allows a fruitful theoretical analysis answering fundamental questions related to estimation consistency, prediction efficiency

Regularization by the L1 norm or other types of penalty function is the most common method to induce sparsity (norms which can be written as linear combinations of norms on subsets of variables can also be used)

structured parsimony is a natural extension, with applications to computer vision, bioinformatics, natural language processing

Consider a convex optimization problem

$$\min_{\boldsymbol{w}\in\mathbb{R}^p} f(\boldsymbol{w}) + \lambda\Omega(\boldsymbol{w}), \tag{1.1}$$

where $f : \mathbb{R}^p \to \mathbb{R}$ is a convex differentiable function and $\Omega : \mathbb{R}^p \to \mathbb{R}$ is a sparsity-inducing—typically nonsmooth and non-Euclidean—norm.

$\mathcal{X} = \mathbb{R}^p$. In this supervised setting, f generally corresponds to the empirical risk of a loss function $\ell : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}_+$. More precisely, given n pairs of data points $\{(\boldsymbol{x}^{(i)}, y^{(i)}) \in \mathbb{R}^p \times \mathcal{Y};\ i = 1, \ldots, n\}$, we have for linear models $f(\boldsymbol{w}) := \frac{1}{n}\sum_{i=1}^{n} \ell(y^{(i)}, \boldsymbol{w}^T\boldsymbol{x}^{(i)})$. Typical examples of loss functions are the square loss for least squares regression, that is, $\ell(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$ with y in $\mathbb{R}$, and the logistic loss $\ell(y, \hat{y}) = \log(1 + e^{-y\hat{y}})$ for logistic regression, with y

where y in {-1, +1}

When one knows a priori that the solutions $\boldsymbol{w}^\star$ of problem (2.1) have only a few non-zero coefficients, Ω is often chosen to be the ℓ_1-norm, that is, $\Omega(\boldsymbol{w}) = \sum_{j=1}^{p} |\boldsymbol{w}_j|$. This leads, for instance, to the Lasso

Regularizing by the ℓ_1-norm is known to induce sparsity in the sense that a number of coefficients of $\boldsymbol{w}^\star$, depending on the strength of the regularization, will be *exactly* equal to zero.

$$\Omega(\boldsymbol{w}) = \sum_{g \in \mathcal{G}} \|\boldsymbol{w}_g\|_q := \sum_{g \in \mathcal{G}} \Big\{ \sum_{j \in g} |\boldsymbol{w}_j|^q \Big\}^{1/q}.$$

This property makes it possible to control the sparsity patterns of $\boldsymbol{w}^\star$ by appropriately defining the groups in $\mathcal{G}$. This form of *structured sparsity* has proved to be useful notably in the context of hierarchical variable selection

As a simple example, let us consider the following optimization problem:

$$\min_{w \in \mathbb{R}} \frac{1}{2}(x - w)^2 + \lambda |w|.$$

Applying proposition 2.1 and noting that the subdifferential $\partial|\cdot|$ is $\{+1\}$ for $w > 0$, $\{-1\}$ for $w < 0$, and $[-1, 1]$ for $w = 0$, it is easy to show that the unique solution admits a closed form called the *soft-thresholding* operator, following a terminology introduced by Donoho and Johnstone (1995); it can be written

$$w^\star = \begin{cases} 0 & \text{if } |x| \leq \lambda \\ (1 - \frac{\lambda}{|x|})x & \text{otherwise.} \end{cases}$$

Overfitting: regularization

A regularizer is an additional criteria to the loss function to make sure that we don't overfit

It's called a regularizer since it tries to keep the parameters more normal/regular

It is a bias on the model forces the learning to prefer certain types of weights over others

$$\text{argmin}_{w,b} \sum_{i=1}^{n} loss(yy') + \lambda\ regularizer(w,b)$$

Regularizers

$$0 = b + \sum_{j=1}^{n} w_j f_j$$

Generally, we don't want huge weights

If weights are large, a small change in a feature can result in a large change in the prediction

Also gives too much weight to any one feature

Might also prefer weights of 0 for features that aren't useful

How do we encourage small weights? or penalize large weights?

Regularizers

$$0 = b + \sum_{j=1}^{n} w_j f_j$$

How do we encourage small weights? or penalize large weights?

$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} loss(yy') + \lambda$$

Common regularizers

sum of the weights

$$r(w,b) = \sum_{w_j} |w_j|$$

sum of the squared weights

$$r(w,b) = \sqrt{\sum_{w_j} |w_j|^2}$$

What’s the difference between these?

Common regularizers

sum of the weights

$$r(w,b) = \sum_{w_j} \left| w_j \right|$$

sum of the squared weights

$$r(w,b) = \sqrt{\sum_{w_j} \left| w_j \right|^2}$$

Squared weights penalizes large values more

Sum of weights will penalize small values more

p-norm

sum of the weights (1-norm) $$r(w,b) = \sum_{w_j} \left|w_j\right|$$

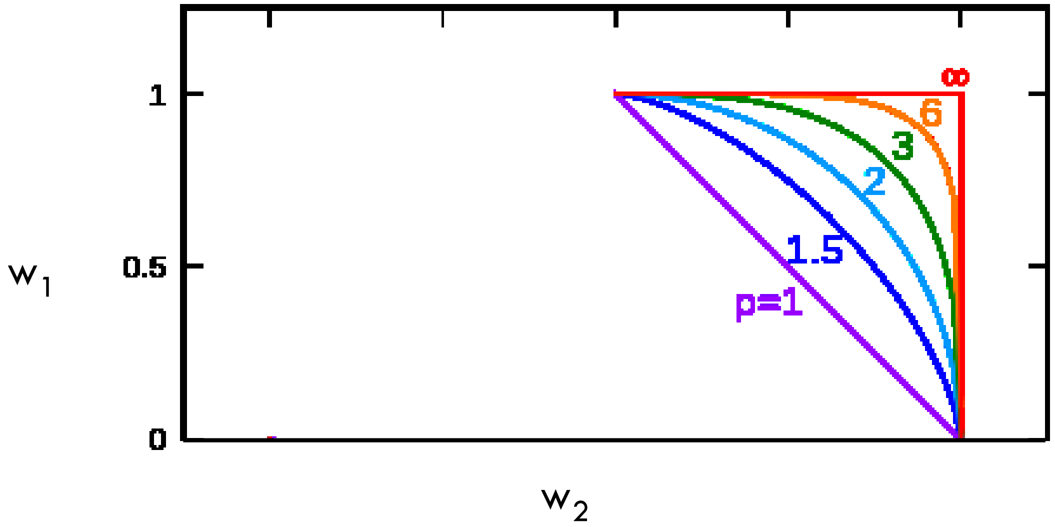
sum of the squared weights (2-norm) $$r(w,b) = \sqrt{\sum_{w_j} \left|w_j\right|^2}$$

p-norm $$r(w,b) = \sqrt[p]{\sum_{w_j} \left|w_j\right|^p} = \left\|w\right\|^p$$

Smaller values of p (p < 2) encourage sparser vectors

Larger values of p discourage large weights more

p-norms visualized

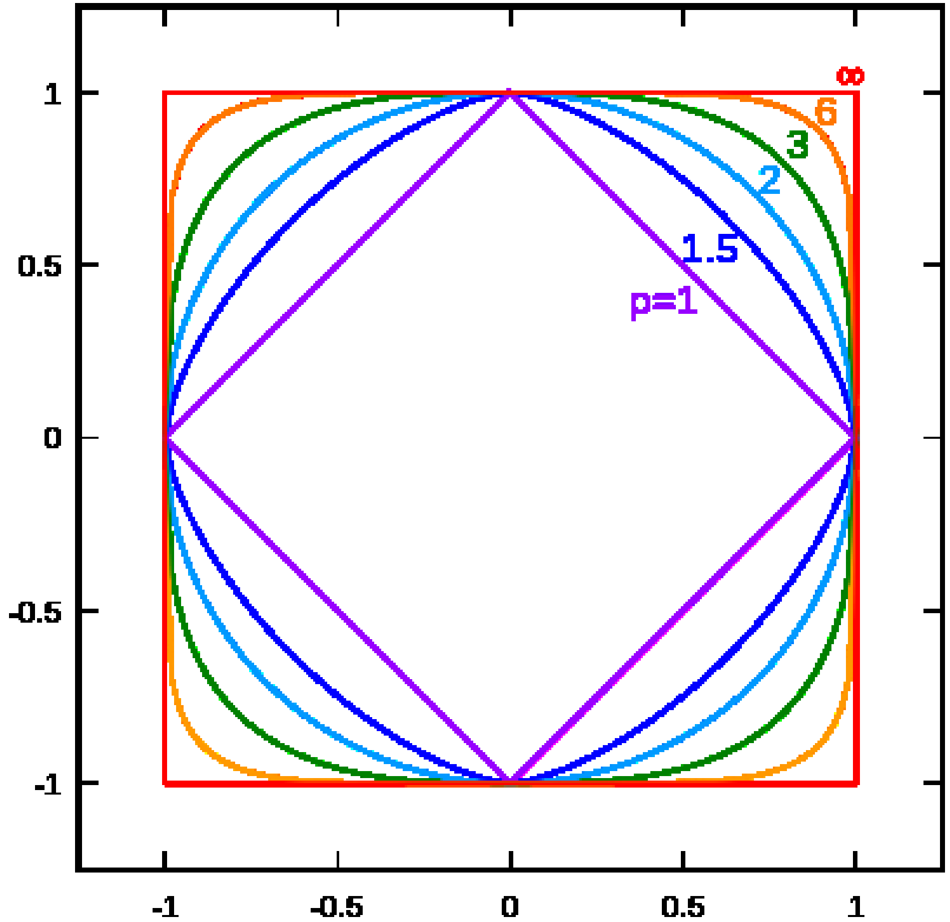


lines indicate penalty = 1

For example, if $w_1 = 0.5$

p	w_2
1	0.5
1.5	0.75
2	0.87
3	0.95
∞	1

p-norms visualized



all p-norms penalize larger weights

p < 2 tends to create sparse (i.e. lots of 0 weights)

p > 2 tends to like similar weights

Model-based machine learning

1. pick a model

$$0 = b + \sum_{j=1}^{n} w_j f_j$$

2. pick a criteria to optimize (aka objective function)

$$\sum_{i=1}^{n} loss(yy') + \lambda regularizer(w)$$

3. develop a learning algorithm

$$\text{argmin}_{w,b} \sum_{i=1}^{n} loss(yy') + \lambda regularizer(w)$$

Find w and b that minimize

Minimizing with a regularizer

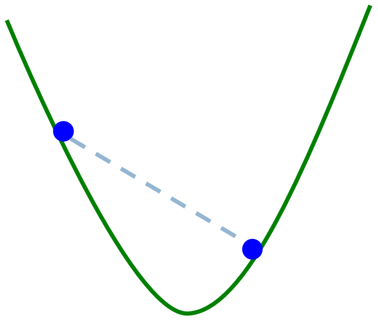
We know how to solve convex minimization problems using gradient descent:

$$\text{argmin}_{w,b} \sum_{i=1}^{n} loss(yy')$$

If we can ensure that the loss + regularizer is convex then we could still use gradient descent:

$$\text{argmin}_{w,b} \sum_{i=1}^{n} \underbrace{loss(yy') + \lambda regularizer(w)}_{\text{make convex}}$$

Convexity revisited



One definition: The line segment between any two points on the function is *above* the function

Mathematically, f is convex if for all x_1, x_2:

$$f(tx_1 + (1-t)x_2) \le tf(x_1) + (1-t)f(x_2) \quad \forall\ 0 < t < 1$$

the value of the function at some point between x_1 and x_2

the value at some point on the **line segment** between x_1 and x_2

Adding convex functions

Claim: If *f* and *g* are convex functions then so is the function z=*f*+*g*

Prove:

$$z(tx_1+(1-t)x_2)\le tz(x_1)+(1-t)z(x_2)\quad \forall\ 0<t<1$$

Mathematically, *f* is convex if for all x_1, x_2:

$$f(tx_1+(1-t)x_2)\le tf(x_1)+(1-t)f(x_2)\quad \forall\ 0<t<1$$

Adding convex functions

By definition of the sum of two functions:

$$z(tx_1+(1-t)x_2)=f(tx_1+(1-t)x_2)+g(tx_1+(1-t)x_2)$$

$$\begin{aligned}tz(x_1)+(1-t)z(x_2)&=tf(x_1)+tg(x_1)+(1-t)f(x_2)+(1-t)g(x_2)\\&=tf(x_1)+(1-t)f(x_2)+tg(x_1)+(1-t)g(x_2)\end{aligned}$$

Then, given that:

$$f(tx_1+(1-t)x_2)\le tf(x_1)+(1-t)f(x_2)$$

$$g(tx_1+(1-t)x_2)\le tg(x_1)+(1-t)g(x_2)$$

We know:

$$f(tx_1+(1-t)x_2)+g(tx_1+(1-t)x_2)\le tf(x_1)+(1-t)f(x_2)+tg(x_1)+(1-t)g(x_2)$$

So: $z(tx_1+(1-t)x_2)\le tz(x_1)+(1-t)z(x_2)$

Minimizing with a regularizer

We know how to solve convex minimization problems using gradient descent:

$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} loss(yy')$$

If we can ensure that the loss + regularizer is convex then we could still use gradient descent:

$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} \underbrace{loss(yy') + \lambda regularizer(w)}_{\text{convex as long as both loss and regularizer are convex}}$$

p-norms are convex

$$r(w,b) = \sqrt[p]{\sum_{w_j} |w_j|^p} = \|w\|^p$$

p-norms are convex for p >= 1

Model-based machine learning

1. pick a model

$$0 = b + \sum_{j=1}^{n} w_j f_j$$

2. pick a criteria to optimize (aka objective function)

$$\sum_{i=1}^{n} \exp(-y_i(w \cdot x_i + b)) + \frac{\lambda}{2}\|w\|^2$$

3. develop a learning algorithm

$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} \exp(-y_i(w \cdot x_i + b)) + \frac{\lambda}{2}\|w\|^2$$

Find w and b that minimize

Our optimization criterion

$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} \exp(-y_i(w \cdot x_i + b)) + \frac{\lambda}{2}\|w\|^2$$

Loss function: penalizes examples where the prediction is different than the label

Regularizer: penalizes large weights

Key: this function is convex allowing us to use gradient descent

Gradient descent

- pick a starting point (w)
- repeat until loss doesn't decrease in all dimensions:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

$$w_i = w_i - \eta \frac{d}{dw_i}(loss(w) + regularizer(w,b))$$

$$\text{argmin}_{w,b} \sum_{i=1}^{n} \exp(-y_i(w \cdot x_i + b)) + \frac{\lambda}{2}\|w\|^2$$

Some more maths

$$\frac{d}{dw_j} objective = \frac{d}{dw_j} \sum_{i=1}^{n} \exp(-y_i(w \cdot x_i + b)) + \frac{\lambda}{2} \|w\|^2$$

$\vdots$ (some math happens)

$$= -\sum_{i=1}^{n} y_i x_{ij} \exp(-y_i(w \cdot x_i + b)) + \lambda w_j$$

Gradient descent

- pick a starting point (w)
- repeat until loss doesn't decrease in all dimensions:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

$$w_i = w_i - \eta \frac{d}{dw_i}(loss(w) + regularizer(w,b))$$

$$w_j = w_j + \eta \sum_{i=1}^{n} y_i x_{ij} \exp(-y_i(w \cdot x_i + b)) - \eta \lambda w_j$$

The update

$$w_j = w_j + \eta y_i x_{ij} \exp(-y_i(w \cdot x_i + b))$$

learning rate

direction to update

constant: how far from wrong

regularization

What effect does the regularizer have?

The update

$$w_j = w_j + \eta y_i x_{ij} \exp(-y_i(w \cdot x_i + b))$$

learning rate

direction to update

constant: how far from wrong

regularization

If w_i is positive, reduces w_i

If w_i is negative, increases w_i

moves w_i towards 0

L1 regularization

$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} \exp(-y_i(w \cdot x_i + b)) + \|w\|$$

$$\frac{d}{dw_j} objective = \frac{d}{dw_j} \sum_{i=1}^{n} \exp(-y_i(w \cdot x_i + b)) + \lambda \|w\|$$

$$= -\sum_{i=1}^{n} y_i x_{ij} \exp(-y_i(w \cdot x_i + b)) + \lambda sign(w_j)$$

L1 regularization

$$w_j = w_j + \eta y_i x_{ij} \exp(-y_i(w \cdot x_i + b)) -$$

learning rate

direction to update

constant: how far from wrong

regularization

What effect does the regularizer have?

L1 regularization

$$w_j = w_j + \eta y_i x_{ij} \exp(-y_i(w \cdot x_i + b)) -$$

learning rate

direction to update

constant: how far from wrong

regularization

If w_i is positive, reduces by a constant
If w_i is negative, increases by a constant

moves w_i towards 0 ***regardless of magnitude***

Regularization with p-norms

L1:

$$w_j = w_j + \eta(loss\_correction - \lambda sign(w_j))$$

L2:

$$w_j = w_j + \eta(loss\_correction - \lambda w_j)$$

Lp:

$$w_j = w_j + \eta(loss\_correction - \lambda c w_j^{p-1})$$

How do higher order norms affect the weights?

Regularizers summarized

L1 is popular because it tends to result in sparse solutions (i.e. lots of zero weights)

- However, it is not differentiable, so it only works for gradient descent solvers

L2 is also popular because for some loss functions, it can be solved directly (no gradient descent required, though often iterative solvers still)

Lp is less popular since they don't tend to shrink the weights enough

The other loss functions

Without regularization, the generic update is:

$$w_j = w_j + \eta y_i x_{ij} c$$

where

$$c = \exp(-y_i(w \cdot x_i + b))$$ exponential

$$c = 1[yy' < 1]$$ hinge loss

$$w_j = w_j + \eta(y_i - (w \cdot x_i + b)x_{ij})$$ squared error

Regularizing by the L1-norm is known to induce sparsity in the sense that, a number of coefficients of coefficients $w^\star$, depending on the strength of the regularization, will be exactly equal to zero.